ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПЕРСОНАЛЬНЫХ ЭВМ

Справочное пособие

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПЕРСОНАЛЬНЫХ ЭВМ

Справочное пособие

Под редакцией
члена-корреспондента АН СССР
А. А. СТОГНИЯ

КИЕВ
НАУКОВА ДУМКА
1989

Авторы

А. А. СТОГНИЙ, С. А. АНАНЬЕВСКИЙ, Я. И. БАРСУК,
В. В. КАРПЕНКО, А. Н. МЕНДЕЛЕВ, А. Ю. ФЕДОРОВСКИЙ,
Л. И. ШОЛМОВ

УДК 681.3

Программное обеспечение персональных ЭВМ / Стогний А. А., Ананьевский С. А., Барсук Я. И. и др.; Под ред. Стогния А. А.— Киев: Наук. думка, 1989.— 368 с.— ISBN 5-12-000487-3.

В справочном пособии приведено описание и сравнительный анализ функций и технических характеристик наиболее распространенных интегрированных инструментальных пакетов программ обработки данных для микроЭВМ, рассмотрены образцы их типового применения и даны рекомендации по их использованию.

Для научных и инженерно-технических работников, преподавателей и студентов; всех лиц, использующих вычислительную технику.

Ил. 22. Табл. 18. Библиогр.: с. 361 (17 назв.)

Печатается по решению редакционной коллегии
справочной литературы АН УССР

Редакция справочной литературы
Заведующий редакцией *В. В. Панюков*
Редактор *Р. И. Гусячая*

П $\frac{2404090000-186}{M221(04)-89}$ 557-89

ISBN 5-12-000487-3

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

В первой половине 80-х гг. в развитии персональных ЭВМ (ПЭВМ) как отдельного класса микропроцессорной техники произошли качественные изменения, вызванные массовым выпуском нового типа 16-разрядных профессиональных ПЭВМ фирмы IBM, быстро нашедших широкое применение. Этот тип ЭВМ зафиксировал также новый качественный уровень программного обеспечения, отражающий новые идеи организации диалога человек — машина.

В отличие от программного обеспечения предыдущих поколений и типов ЭВМ, которое являлось результатом решений, принимаемых профессиональными программистами, программное обеспечение ПЭВМ получило оценку массового пользователя и вызвало новую, весьма плодотворную обратную связь, повлиявшую на появление фактически новых принципов и технологий производства программного обеспечения. Основной характеристикой прикладного программного обеспечения ПЭВМ стал служить реализованный интерфейс конечного пользователя.

С учетом того что в ближайшем будущем основными техническими средствами отечественного производства для автоматизации учрежденческих работ станут 16-разрядные ПЭВМ типа ЕС 1841, аналогичные IBM PC, перечислим кратко основные классы программных изделий для этих ПЭВМ, использующих такие возможности аппаратуры, как программируемая клавиатура, манипуляторы и видеосканеры для ввода, точечная графическая печать, цветные видеомониторы и синтезаторы звука для вывода, дисковая внешняя память с фиксированными дисками емкостью более 10 Мбайт и сменными дискетами емкостью более 300 Кбайт, системные блоки с 16-разрядными основными микропроцессорами и сопроцессорами и оперативной памятью более 500 Кбайт.

Можно выделить следующие классы программных изделий для ПЭВМ: базовая операционная среда и комплекс обслуживающих программ (утилит); базовые системы программирования (компиляторы языков программирования с соответствующими библиотеками, сервисным обеспечением и др.); инструментальные пакеты, которые можно разделить на специализированные и интегрированные; прикладные пакеты по различным областям конкретных приложений. Общее количество программных изделий за рубежом оценивается в несколько тысяч.

Основной базовой операционной средой 16-разрядных ПЭВМ является операционная система MS DOS фирмы «Microsoft», пришедшая на смену операционной системе CP/M фирмы «Digital Research» для 8-разрядных ПЭВМ [1].

Базовые системы программирования представлены семейством компиляторов языков СИ, ПАСКАЛЬ, ФОРТРАН фирмы «Micro-

soft» [2], имеющих унифицированный интерфейс объектного уровня. Особо представлен язык BASIC, который имеет как программную, так и аппаратную реализации. Для перечисленных языков интенсивно развиваются библиотечные окружения, превращающие эти системы программирования в своеобразные инструментальные пакеты.

Особо следует отметить реализации языков фирмой «Borland» с префиксным названием TURBO (TURBO PASCAL, TURBO C, TURBO PROLOG, TURBO BASIC) [3], в которых удачно сочетаются традиционные возможности компиляторов и средства быстрой отладки в режиме исполнения без фиксации объектного кода.

К специализированным инструментальным пакетам можно отнести подклассы, соответствующие классическим приложениям автоматизации учрежденческих работ (office automation) за рубежом. Среди них семейство текстовых и графических редакторов для подготовки и выпуска разнообразных документов, например текстовые редакторы WordStar (в разных версиях для 8-разрядных ПЭВМ в среде CP/M и 16-разрядных ПЭВМ) фирмы «MicroPro», MS Word фирмы «Microsoft», графические редакторы Grafix Partner фирмы «Brihtbill Roberts», PC Paintbrush фирмы «Zsoft» и др. [4,5]. К этой группе можно также отнести пакеты поддержки некоторых внешних устройств, например пакет EyeStar фирмы «Microtek International», обеспечивающий сканирование изображений устройствами видеоввода с последующей их обработкой.

Отдельную группу специализированных пакетов представляют процессоры электронных таблиц (spreadsheet). Данная группа интенсивно использовалась на 8-разрядных ПЭВМ, а для 16-разрядных машин в основном погрузилась в качестве компонента интегрированных пакетов, однако и для 16-разрядных ПЭВМ она представлена пакетами Multiplan фирмы «Microsoft» и SuperCalc-3 фирмы «Sorsim» (развитие соответствующих пакетов 8-разрядных ПЭВМ).

К специализированным инструментальным пакетам отнесем также СУБД, среди которых ведущее место принадлежит dBASE III Plus фирмы «Ashton Tate» [6] (развитие СУБД dBASE II для 8-разрядных ПЭВМ) и совместимому компилятору пакетной части входного языка этой СУБД — Clipper фирмы «Nantucket» [7]. СУБД представлены также такими пакетами, как R : base System V фирмы «Microrim» и Paradox фирмы «Ansa Software». Указанные СУБД являются законченными инструментами, использующими собственную исполнительную среду. Для ПЭВМ имеются также СУБД, ориентированные на использование других языков манипуляции данными. Характерным представителем их является СУБД Vista фирмы «Raiva», ориентированная на использование языка СИ.

Основными среди интегрированных пакетов являются пакеты Symphony фирмы «Lotus». Knowledgeman фирмы «MDBS» и Framework II фирмы «Ashton Tate» [8, 9]. Возможности интегрированных пакетов реализованы и в различных версиях объектно-ориентированной системы Smalltalk фирмы «Digitalk», хотя ее идеология весьма отлична от традиционных программных систем [10].

Термин «интегрированный» для данного класса инструментальных средств означает, что они объединили в рамках некоторого универсального интерфейса пользователя такие основные приложения, как обработка текстов и табличной информации, ведение реляционных баз данных и получение деловой графики. Языковый интерфейс ориентирован на непрофессионального в области программирования англоязычного конечного пользователя, однако в состав интегрированных пакетов включен алгоритмически полный набор языковых

средств, позволяющий осуществлять профессиональные программистские приложения и расширения.

Отдельный класс программной продукции ПЭВМ составляют прикладные пакеты для различных областей приложений — от простых обучающих и игровых программ до пакетов автоматизированного проектирования (например, пакет AutoCADD) и экспертных систем (например, пакет GURU), однако при анализе применимости пакетов следует учитывать среду реализации, которая в основном определяется использованным инструментарием (так, пакет GURU использует среду Knowledgeman и т. п.).

Следует отметить существенное влияние национальных языковых особенностей в организации интерфейса конечного пользователя и ПЭВМ, реализованного в соответствующих программных изделиях. Особенности национального языка в подобных пакетах прикладных программ касаются не только вопросов алфавита, но и лексики, морфологии и даже синтаксиса естественного языка, что отражается на структуре и входном языке пакета, а также влияет на другие компоненты (обеспечивающие режимы помощи (HELP), проверки орфографии (SPELLING) и т. п.), не говоря уже о системах, взаимодействующих на языках, близких к естественным, и с речевым вводом — выводом.

В данном справочном пособии рассмотрены основные характеристики следующих программных изделий для ПЭВМ, ориентированных на использование в приложениях, связанных с автоматизацией учрежденческих работ: СУБД dBASE III Plus с компилятором Clipper, представляющие некоторую инструментальную систему создания и сопровождения баз данных; интегрированного пакета Knowledgeman и развивающейся объектно-ориентированной системы Smalltak-80. Основанием подобного выбора послужило то, что данные изделия в значительной степени отражают основные концепции нового поколения программных средств для ПЭВМ.

Необходимо отметить определенные терминологические трудности при изложении данного материала. По мнению авторов, нет большого вреда от прямой транслитерации ряда английских терминов, которые хотя и имеют естественные русские эквиваленты, но либо звучат неточно, либо требуют каждый раз дополнительных пояснений. Книга адресуется в первую очередь читателям, которые имеют или будут иметь доступ к ПЭВМ и, следовательно, столкнутся с обилием англоязычных пакетов программ ПЭВМ, которые необходимо изучать в оригиналах, чтобы в короткие сроки выйти на передовой уровень аналогичных отечественных разработок (а тогда будет сформирована и соответствующая отечественная терминология).

Авторы

Часть I

ИНСТРУМЕНТАЛЬНЫЙ КОМПЛЕКС СОЗДАНИЯ БАЗ ДАННЫХ dBASE III PLUS, CLIPPER

Глава 1

АРХИТЕКТУРА dBASE-МАШИНЫ

1.1. Общая характеристика

dBASE III Plus представляет собой реляционную СУБД со встроенным специализированным языком программирования, позволяющим в диалоговом и пакетном (программном) режимах создавать и обслуживать персональные базы данных, включая программную обработку информации из баз. Пользователь СУБД фактически работает с некоторой dBASE-машиной (неймановского типа), архитектура которой представлена на рис. 1.

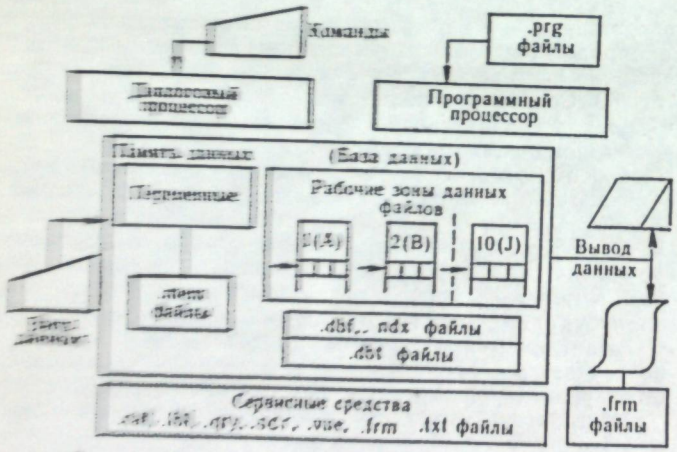


Рис. 1. Архитектура dBASE III Plus машины.

Основными компонентами dBASE-машины являются структуры данных, процессоры и средства ввода — вывода. Память dBASE-машины используется для хранения информации, процессоры выполняют входные команды (операторы), представляющие входной язык dBASE III Plus, а средства ввода — вывода обеспечивают обмен информацией между пользователем и базой данных.

Основными базой данных являются файлы данных, имена которых в дисковой системе MS DOS имеют стандартное расширение .dbf (например, NAME.dbf, ORDER7.dbf).

Файлы данных представляют собой прямоугольные таблицы следующего вида:

	Поле 1	Поле 2	. . .	Поле N
Запись 1				
Запись 2				
Запись M				

Строки этих таблиц называются записями, а столбцы представляют поля записей. Все записи имеют одни и те же типы полей. Например, список сотрудников, характеризуемых рядом реквизитов:

Ф. И. О.	Должность	Год рождения
1. Иванов И. И.	Нач. отд.	1935
2. Петров П. П.	Вед. инж.	1940
3. Сидоров С. С.	Инж.	1955

В терминах реляционных СУБД подобные таблицы называются отношениями, их записи представляют кортежи отношений, а поля называются атрибутами отношений.

Конкретная база данных состоит из одного или нескольких подобных файлов.

Память dBASE-машины состоит из некоторой области оперативной памяти, которая содержит информацию, доступную для обработки процессором, а также служит буфером операций ввода—вывода (содержимое оперативной памяти сохраняется только в сеансе работы) и внешней памяти, в которой сохраняются данные при выключениях ПЭВМ. Основной механизм доступа к данным следующий.

Файлы данных находятся во внешней памяти, а доступ к ним осуществляется в рабочих зонах оперативной памяти, причем одновременный доступ разрешается не более чем к десяти .dbf файлам. Доступ обеспечивается путем назначения данному .dbf файлу зоны (команда SELECT ⟨номер зоны⟩) и его открытия (команда USE ⟨имя .dbf файла⟩). Для открытого файла в операциях обработки доступна одна, так называемая текущая, запись (после открытия файла — первая), на которую ссылается специальный указатель.

Для того чтобы осуществлять доступ к другим записям, необходимо изменять указатель текущей записи, что достигается различными средствами языка dBASE. По окончании работы с данными .dbf файла его закрывают (команда USE), что сопровождается отсоединением его от выделенной зоны оперативной памяти и обеспечивает гарантию сохранения информации в состоянии, соответствующем последним изменениям.

В состав собственно базы данных наряду с .dbf файлами входят также .dbt и .ndx файлы, назначение которых будет рассмотрено ниже.

Оперативная память dBASE-машины может использоваться также для хранения промежуточных данных и некоторых неструктурированных (скалярных) элементов, которые представлены понятием «переменная». Для сохранения во внешней памяти и последующего восстановления текущих значений переменных используется специальный тип файла со стандартным расширением .mem (дамп памяти).

В состав сервисных средств, позволяющих задавать различные пользовательские форматы организации обработки данных, а также форматы вывода и ввода информации и ряд других сервисных функций, входит ряд служебных файлов, имеющих стандартные расширения .cat, .lbl, .qry, .scr, .vue, .fmt, .txt, .frm.

dBASE-машину представляют два процессора: диалоговый и программный.

Диалоговый процессор выполняет команды языка dBASE в диалоговом режиме и имеет два подрежима: ввод и исполнение команд и меню-ориентированный режим помощи.

В первом подрежиме пользователь вводит команды в соответствии с правилами входного языка и наблюдает реакцию dBASE-машины.

Во втором подрежиме на экране дисплея появляется подсказывающее меню для определенного подмножества наиболее употребительных команд, и формирование команд с необходимыми параметрами (опциями) производится путем их выбора с помощью клавиш управления курсором. Вызов меню-ориентированного подрежима производится командами assist, set и т. д.

Программный процессор выполняет программы, заранее составленные на языке dBASE III Plus как на обычном языке программирования и помещенные в программные файлы со стандартным расширением .prg. Эти файлы являются текстовыми и исполняются в режиме интерпретации последовательности записанных в них команд. Отметим, что набор команд (операторов) для диалогового процессора не совпадает полностью с набором, используемым программным процессором, поскольку в состав последнего входят средства программирования разветвляющихся циклических программ и подпрограмм, а некоторые операторы ориентированы только на диалоговый режим.

В общем случае процессоры имеют много различных состояний (режимов), которые определяют особенности выполнения команд. Установка этих состояний производится специальной группой команд (set-команды).

В качестве средств ввода и вывода информации в dBASE-машине используются группы команд, обеспечивающие ввод потоков символов, образующих некоторые последовательности таких единиц языка, как числа, строки, тексты и т. п., из некоторого входного логического буфера (входной поток) и вывод аналогичного потока символов в выходной буфер (выходной поток). Как правило, в качестве физического устройства, соответствующего входному потоку, используется клавиатура, а в качестве устройства вывода — дисплей. При необходимости выходной поток может назначаться и на устройство печати. Обычно вводимая с клавиатуры в память информация одновременно отображается на экране дисплея.

Существенным преимуществом СУБД dBASE III Plus является наличие специального компилятора Clipper, с помощью которого можно получать законченные приложения, написанные на языке Clipper. Входной язык Clipper является подмножеством языка dBASE III Plus, что позволяет после удобной диалоговой отладки программ в интерпретирующей среде dBASE III Plus скомпилировать программы и получить исполнительный программный модуль для исполнительной среды MS DOS, независимый от dBASE III Plus интерпретатора. Скорость выполнения программ при этом существенно возрастает.

1.2. Структуры данных и элементарные операции

В рассмотренной модели основной единицей языка, управляющей работой dBASE-машины, является команда (оператор), которая имеет следующий общий вид:

⟨глагол⟩ [⟨зона действия⟩] [⟨список выражений⟩]
[⟨условие выполнения⟩] [⟨ограничение⟩]

где ⟨глагол⟩ представляет некоторое ключевое слово (английское), определяющее вид действия, осуществляемого командой, а другие компоненты, заключенные в угловые скобки, называются опциями. Опции уточняют:

зону базы данных, на которую распространяется действие команды (опция ⟨зона действия⟩ имеет возможные значения: ALL — вся база, NEXT ⟨N⟩ — следующие N записей, REST — остаток базы данных, RECORD ⟨N⟩ — запись с заданным номером N);

объекты, используемые при выполнении команды (опция ⟨список выражений⟩);

условие выполнения команды, имеющее вид FOR ⟨логическое выражение⟩ (опция ⟨условие выполнения⟩);

ограничение на выполнение команды, имеющее вид WHILE ⟨логическое выражение⟩ (опция ⟨ограничение⟩).

Опции в общем виде команды заключены в квадратные скобки, что означает возможность умолчания любой из них. Например, команда

LIST ALL a, b, c FOR a<b WHILE c = 10

означает, что из некоторой существующей и доступной в данный момент базы данных (.dbf файла) необходимо вывести (LIST) значения полей a, b, c для тех записей, которые удовлетворяют условию a < b. Команда работает до тех пор, пока c = 10. Поиск записей проводится по всему файлу. В данном примере ни одна из опций команды не умалчивается.

Рассмотрим базовые элементы языка, описывающие элементарные компоненты данных и операций над ними и образующие одно из центральных понятий языка — «выражение».

Элементарной единицей данных является скалярная величина, которая может иметь следующие основные типы значений:

C — символьное или строчное (Character);
N — числовое (Numeric);
L — логическое (Logical);
D — дата (Date).

Указанные элементарные данные называют первичными. Они представляются в языке в трех основных конструкциях: константах, переменных и полях записей .dbf файлов.

Переменные и поля записей обозначаются идентификаторами (именами). Имена конструируются из букв, цифр и знака подчеркивания «—» длиной не более 10 символов и начинаются с буквы. Для полей записей допускается также еще один специальный тип значений M — текст (memory), который будет рассмотрен отдельно.

Типы значений имен определяются при задании структуры .dbf файлов (для полей) или при присваивании значений (для переменных). Переменные и поля записей в процессе работы принимают некоторые значения, которые могут изменяться. Константы представляют собой самостоятельные элементы команд (в составе выражений); которые определяют фиксированные и неизменяемые значения соответствующих типов.

Примеры.

0, 15, 173.77 — числовые константы;
'string' [this is строчная константа] — символьные константы;
.T., .F., Y, N — логические константы (истина, ложь).

Константы типа дата могут быть заданы как символьные константы — аргументы специальных функций преобразования, например ctod ('13/01/88').

Над первичными элементами допускаются операции в соответствии с их типом. Операция над элементами одних типов может порождать результат того же или другого типа.

Следующие арифметические операции применяются для числовых типов и вырабатывают числовые значения:

\+ — сложение;
— — вычитание;
* — умножение;
/ — деление;
** или ^ — возведение в степень.

Арифметические операции сложения и вычитания допускаются также, если один из операндов имеет тип дата, что соответствует прибавлению (вычитанию) некоторого количества дней к (от) заданной дате, при этом результат имеет тип дата. Вычитание допускается для двух операндов типа дата. В этом случае вычисляется интервал между датами в днях и результат является числовым значением.

Следующие операции отношения применяются к операндам одного типа и вырабатывают логический тип результата (.T.— истина, .F.— ложь):

< — меньше;
> — больше;
= — равно;
< > или # — не равно;
<= — меньше или равно;
>= — больше или равно;
$ — вхождение подстроки (применяется для символьных операндов: A $ B равно истине, если подстрока A находится в строке B).

Над логическими типами допускаются логические операции, вырабатывающие результаты логического типа:

.NOT. — отрицание (логическое НЕ);
.AND. — конъюнкция (логическое умножение И);
.OR. — дизъюнкция (логическое сложение ИЛИ).

Над символьными типами разрешены операции конкатенации (сцепления), результатом которых является строка, образованная сцеплением строк операндов:

\+ — сцепление двух строчных операндов;
— — сцепление двух строчных операндов, при котором заключительные символы пробелов первого операнда помещаются в конец сцепленной строки.

Первичные элементы (константы, переменные и поля записей), возможно, связанные рассмотренными знаками операций, образуют выражения. Выражения вычисляются машиной dBASE III Plus в порядке старшинства операций (сначала арифметические от возведения в степень до сложения и вычитания над первичными числовыми данными или конкатенации над первичными символьными данными, затем операции отношений и, наконец, логические). Однородные операции выполняются слева направо. При необходимости изменения порядка выполнения операций используются алгебраические круглые скобки (,), выражение в которых вычисляется в первую очередь. Приведем некоторые примеры выражений:

(2 + 2) * x1/x2 — арифметическое (по типу результата) выражение, в котором x1 и x2 (переменные или поля записей) должны иметь числовой тип;

a1 + a2 < 15.5.AND. a1 + a2>15.1 — логическое выражение, равное истине, если сумма числовых значений a1 + a2 находится в пределах 15.1 и 15.5;

fname + sname — выражение, которое интерпретируется в зависимости от типов операндов fname и sname; например, если они числовые, то результат — арифметическая сумма, если они символьные, то результат — сцепленная строка.

Операции над разнотипными операндами (кроме указанных выше) не допускаются и вызывают сообщения об ошибках.

В языке имеется еще один важный вид первичных компонентов выражений — функции. Вычисление функции приводит к получению некоторого результата — значения определенного типа. Функции могут иметь аргументы и дополняют вычисления, производимые машиной dBASE III Plus, такими возможностями, как вычисления элементарных математических функций квадратного корня (sqrt()), абсолютного значения (abs ( )) и др., а также многими другими, например получения текущего времени и даты (time ( ), date ( )), преобразования одних типов значений в другие. Аргументы записываются в круглых скобках, которые обязательны, даже если аргументы отсутствуют. Полный список функций приведен в приложении 1.

В заключение данного раздела рассмотрим некоторые команды языка dBASE, которые позволяют вычислять произвольные выражения и выводить вычисленные значения на экран дисплея, т. е. обеспечивать работу мощного калькулятора.

Оператор присваивания значения переменным имеет следующую форму:

⟨переменная⟩ = ⟨выражение⟩

Например, rad = 17.882 вызовет присваивание переменной rad значения 17.882. Последующий оператор rad = (rad — 1) * 2 вычислит значение (17.882—1) * 2 и присвоит полученное значение 33.764 в качестве нового значения переменной rad. Отметим, что процессор dBASE-машины, выполняя данный оператор, может находиться в состоянии, при котором вычисление и присваивание значений сопровождается выводом результата на экран. В это состояние процессор переводится командой установки SET TALK ON. По команде SET TALK OFF процессор выполняет вычисление без сопровождающего вывода результатов на экран. В этом случае для вывода результатов может использоваться оператор вывода

? ⟨список выражений⟩

где ⟨список выражений⟩ представляет последовательность выражений, разделенных запятой, например

? 2 * 3.141 * rad, 3.141 * rad ** 2

В данном примере предполагается, что значения переменной rad определены предыдущими вычислениями. После выполнения данной команды на экран будут выведены вычисленные значения длины окружности и площади круга радиусом rad.

1.3. Основные классы диалоговых команд

В диалоговом режиме используются следующие классы команд:

команды создания, уничтожения и организации доступа к файлам базы данных;

команды манипулирования (изменения значений) данными;

команды ввода и вывода значений;

служебные команды.

К первой группе относятся команды, с помощью которых пользователь организует на машине dBASE III Plus определенную информационную среду, превращая эту машину в некоторую конкретную систему обработки данных.

Вторая группа команд включает действия, производимые над данными, отображаемыми в памяти машины dBASE III Plus.

Третья группа команд обеспечивает действия по вводу данных из внешней среды и выводу результатов на внешние носители. В этой группе будут рассмотрены также команды, которые редко используются в прямом режиме диалога, а полезны лишь при составлении программ, однако они отнесены сюда из соображений целостности представления машины dBASE III Plus.

В четвертую группу отнесены команды служебного назначения по управлению машиной dBASE III Plus.

Рассмотрим кратко основные команды (полный список команд приведен в приложении 2).

1.3.1. Команды создания, уничтожения и организации доступа к файлам базы данных

Для создания .dbf файла используется команда CREATE. По этой команде задается имя файла и формируется структура записи файла путем указания имени полей, типов их значений с указанием длин полей в байтах. Формирование производится в диалоговом режиме. Созданный файл можно открыть (т. е. предоставить ему

доступную для обработки записей область оперативной памяти dBASE-машины) командой USE ⟨имя файла⟩ в заданной рабочей зоне, если последняя явно определена командой SELECT ⟨номер рабочей зоны⟩; в противном случае файл открывается в первой зоне. Файл закрывается командами USE (без указания имени, что соответствует закрытию файла в текущей активной рабочей зоне) или CLOSE. Последняя команда имеет различные опции, позволяющие использовать ее при работе и с другими типами файлов. Файлы базы данных могут быть закрыты также командой CLEAR ALL, однако при этом производятся дополнительные действия по очистке памяти.

В общем случае, если в данной рабочей зоне открыт файл, новая команда USE ⟨имя файла⟩ закрывает открытый файл и открывает новый.

Для открытого файла можно осуществлять операции добавления и исключения записей. Добавление производится командой APPEND, имеющей различные опции, или командой INSERT, а исключение — командой DELETE, имеющей перечисленные в начале раздела опции зоны действия, условия исключения и ограничения. Команда APPEND добавляет записи в конец файла, а команда INSERT — за текущей записью или в виде INSERT BEFORE — перед текущей записью.

Как правило, организация базы данных предполагает различное упорядочение записей в файлах данных. В dBASE-машине предусмотрено два основных способа упорядочения (сортировки) данных по возрастанию или убыванию значений некоторых полей. В первом случае записи сортируются физически командой SORT, опции которой определяют имя нового (отсортированного) файла, ключ сортировки (имена полей) и порядок сортировки (убывающий или возрастающий). Если в качестве ключей задается несколько полей, то сортировка производится сначала по первому ключу, а затем внутри соседних записей, имеющих одинаковое значение этого ключа, производится дополнительное упорядочение по следующему ключу и т. д.

Второй способ упорядочения заключается в построении дополнительных так называемых индексных файлов к данному .dbf файлу. Имена индексных файлов имеют стандартные расширения .ndx.

Индексные файлы создаются командой INDEX, в которой указываются имя индексного файла и некоторое выражение, использующее имена полей индексируемого .dbf файла и являющееся ключом упорядочения по возрастанию (убывание легко задать, определив обратное выражение) аналогично сортировке. При индексировании основной файл физически не переупорядочивается, однако доступ к его записям осуществляется через индексный файл таким образом, что последовательность соответствующих обращений производится как к отсортированному файлу. При работе с индексными файлами необходимо при открытии файла данных указать используемые индексные файлы (USE ⟨имя файла данных⟩ INDEX ⟨список индексных файлов⟩). Если для данного файла данных построено несколько индексных файлов, то это означает возможность одновременного использования нескольких копий по-разному отсортированных файлов данных и содержимого основного файла.

Использование индексных файлов требует определенной осторожности. Если с основным файлом производятся манипуляции, связанные с изменением содержимого полей, входящих в ключи упорядочивания, в том числе при добавлениях или исключениях записей, то файл должен быть открыт со всеми своими индексными

файлами для оперативного внесения изменений. В противном случае индексные файлы необходимо строить заново. Однако определенные операции (например, переход к следующей записи) допускаются только тогда, когда файл открыт с не более чем одним индексом, и, следовательно, необходимо оперативно отслеживать возможные изменения файлов, что усложняет работу, снижает эффективность сопровождения базы данных и может приводить к достаточно тонким ошибкам.

К рассматриваемой группе команд необходимо отнести команды управления указателем текущей записи. Значение указателя текущей записи может быть получено с помощью функции recno ( ). Данная функция дает значение номера текущей записи основного .dbf файла, которое в случае использования индексирования может не соответствовать истинной последовательности просмотра файла. С помощью функции recount ( ) можно вычислить текущее количество записей dbf файла. Имеется также важная логическая функция eof ( ) (end of file — конец файла). Данная функция возвращает значение истина (.T.), если указатель текущей записи установлен на некоторую действительную запись файла, и возвращает значение ложь (.F.), если в процессе просмотра файла достигнут конец файла.

Изменение указателя текущей записи (после открытия файла — первой) производится двумя основными способами.

Первый способ использует команды SKIP [⟨выражение⟩] и GOTO с параметрами следующего вида: TOP, BOTTOM или ⟨выражение⟩. Команда SKIP изменяет текущее значение указателя на величину заданного выражения, а при отсутствии его — на +1. Команда GOTO TOP устанавливает указатель на первую запись, команда GOTO BOTTOM —на последнюю запись файла, команда GOTO ⟨выражение⟩ — на запись, номер которой определяется заданным выражением.

Второй способ изменяет указатель текущей записи неявно в связи с поиском записей по некоторым условиям. Основным средством поиска записей файлов является команда LOCATE, имеющая общую стандартную форму

LOCATE [⟨зона действия⟩] [FOR ⟨условие⟩] [WHILE ⟨ограничение⟩]

По этой команде в заданной зоне действия (по умолчанию во всем файле) и независимо от текущего значения указателя записи, начиная с первой записи, производится поиск записи, удовлетворяющей заданному условию поиска (в виде выражения, имеющего логический тип значений) до тех пор, пока заданное в ограничении логическое выражение истинно. Например, создадим базу данных, хранящую некоторые сведения об улицах города:

CREATE streets

После ввода команды в диалоге (т. е. по предлагаемой dBASE-машиной схеме) сформируем следующую структуру записи файла streets:

Имя поля	Тип	Длина	Точность (для дробных чисел)
1. region	C	15	(район)
2. street	C	20	(улица)
3. length	N	5	(протяженность)
4. surface	C	12	(покрытие)
5. termpr	D	8	(дата ремонта)

Далее, командой APPEND введем значения соответствующих полей для конкретных улиц города. Ввод осуществляется в диалоге, при котором dBASE-машина выводит на экран структурную заготовку пустой записи с наименованием полей с возможностью редактирования (исправления) вводимых значений.

Например:

APPEND

Информация, выводимая dBASE-машиной	Информация, вводимая пользователем
1. region	Дарницкий
2. street	Энтузиастов
3. length	2500
4. surface	асфальт
5. termpr	15/06/87

Команда APPEND позволяет вводить несколько записей в одном сеансе, запрашивая после ввода очередной записи подтверждение на продолжение ввода.

После создания подобной базы данных команда

LOCATE FOR region='Московский' .AND. length)1000

установит указатель текущей записи на запись о первой (по нахождению в файле streets .dbf) улице Московского района, длина которой больше 1 км. Наименование улицы и дату ремонта можно вывести на экран с помощью известной команды

? street, termpr

Если запись, удовлетворяющая этому условию, отсутствует, то достигается конец файла, что может быть обнаружено функцией eof ( ):

? eof ( )

Если в базе данных имеется несколько записей, удовлетворяющих заданному условию, то для поиска следующей после найденной записи используется команда CONTINUE.

Отметим, что выполнение команды LOCATE и последующих CONTINUE связано с последовательным просмотром базы данных. Физически это означает чтение с внешнего носителя небольшими порциями записей в оперативную память и последующий анализ условий поиска, что для больших баз данных может занимать существенное время. Ускорение поиска требуемой записи является еще одной важной функцией индексированных файлов данных. При наличии индекса поиск может быть осуществлен командой FIND с параметром в виде некоторой строки (символьной константы). По этой команде ищется запись, индексное поле которой совпадает с указанной в параметре-строке. Например, последовательность команд

USE streets
INDEX ON street TO indstr

обеспечит построение индекса, упорядочивающего рассмотренный файл streets .dbf в алфавитном порядке наименований улиц.

Для рассматриваемого вида поиска возможен следующий набор команд:

```
USE streets INDEX indstr
FIND 'Крещатик'
? region+' район',' длина '+str (length)+' м'
```

Реакцией dBASE-машины на последнюю команду является вывод на экран следующего сообщения (функция str ( ) преобразует числовой тип в символьный, что необходимо для выполнения строчной операции конкатенации (+)):

Ленинский район длина 1500 м

Значительное ускорение поиска в рассмотренном примере достигается с помощью специальных алгоритмов сравнения параметра команды FIND с упорядоченными значениями индекса в файле indstr .ndx без анализа содержимого основного файла streets .dbf. Параметром команды FIND может быть числовая константа, если индексирование велось по числовому полю.

Поиск записи в индексированном файле производится также более универсальной командой SEEK, параметром которой является произвольное выражение. Выражение вычисляется и служит ключом поиска, что может обеспечить большую гибкость поиска.

Следует учитывать особенности использования группы команд, изменяющих указатель текущей записи файла, при работе в нескольких рабочих зонах. Такие команды, как GOTO, SKIP, CONTINUE, LOCATE, FIND и SEEK, применяются к файлам, открытым в активной рабочей зоне, т. е. имеющей номер, заданный последней выполненной командой SELECT. Поэтому необходимо правильно активизировать требуемые рабочие зоны с помощью своевременного использования команд SELECT.

В заключение обзора группы команд организации базы данных рассмотрим случай реорганизации существующей базы данных, когда требуется изменить структуру записи с сохранением содержимого базы для немодифицируемых полей (такая возможность характерна для реляционных СУБД, к которым относится dBASE III Plus). Изменение структуры производится оператором MODIFY STRUCTURE, при выполнении которого dBASE-машина начинает диалог аналогично диалогу при выполнении оператора CREATE, выводя шаблон имеющейся структуры и допуская ее изменение (добавление и исключение полей или корректировку их описаний). Однако все имеющиеся записи при этом уничтожаются. Поэтому рассматриваемую реорганизацию технологически необходимо проводить за несколько шагов: сохранение текущего файла путем копирования его в некоторый рабочий буферный файл оператором COPY, изменение структуры оператором MODIFY STRUCTURE, перепись оператором APPEND FROM предыдущего содержимого из рабочего файла в реконструированный, исключение временного рабочего файла оператором ERASE.

Например, для рассмотренного файла улиц можно добавить поле 'площадь покрытия' с помощью последовательности действий:

действие 1 — вводится последовательность команд

```
USE streets
COPY TO workfile
MODIFY STRUCTURE
```

действие 2 — в ответ dBASE-машина выведет на экран существующую структуру

Имя поля	Тип	Длина	Точность (для дробных чисел)	
1. region	C	15		(район)
2. street	C	20		(улица)
3. length	N	5		(протяженность)
4. surface	C	12		(покрытие)
5. termpr	D	8		(дата ремонта)
6. square	N	10	2,	

действие 3 — пользователь добавляет поле 6

действие 4 — вводится последовательность команд

```
APPEND FROM workfile
ERASE workfile
```

Оператор APPEND FROM переносит записи из заданного файла в открытый по тем полям, которые имеют одинаковые наименования у обоих файлов. При этом оператор может быть дополнен опцией условия переноса FOR ⟨логическое выражение⟩. Возможны и другие технологии реорганизации структур, например путем полного или частичного копирования структуры оператором COPY STRUCTURE с опцией FIELDS ⟨список полей⟩ в новый .dbf файл с последующими изменениями скопированной структуры и переписью информации из старого файла в новый оператором APPEND FROM и т. д.

1.3.2. Команды манипулирования данными

К этой группе команд следует отнести уже рассмотренный оператор присваивания значений вида ⟨переменная⟩ = ⟨выражение⟩ и его разновидность

STORE ⟨выражение⟩ TO ⟨список переменных⟩

осуществляющую присваивание значения выражения нескольким переменным. Отметим, что в состав компонентов вычисляемого выражения могут входить наименования полей записей открытых файлов данных, и в этом случае при вычислениях используются значения указанных полей для текущих (т. е. доступных) записей. Для устранения возможных неоднозначностей, вызванных одинаковым наименованием полей разных файлов (а возможно, и переменных памяти), имена могут дополнительно специфицироваться префиксом рабочей зоны, в которой открыт файл. Префикс указывается латинской буквой (строчной либо прописной), соответствующей порядковому номеру рабочей зоны (A — 1, B — 2 и т. д.) (см. рис. 1), и отделяется от имени стрелкой. Префикс M означает, что данное наименование относится к переменной памяти. Например, пусть база данных включает два файла file 1 и file 2, имеющие поле с одинаковым наименованием name. Тогда в последовательности

```
SELECT 1
USE file 1
SELECT 2
USE file 2
? a —> name, b —> name
```

последним оператором будут выведены значения полей name первого и второго файлов. Оператор

```
? name, name
```

вызвал бы вывод дважды значения поля name второго файла.

Изменение значений полей записей производится различными способами. Для диалоговой корректировки используются возможности так называемого экранного редактирования с помощью оператора EDIT. На экране появляется заданная структура записи с ее текущим содержимым, а корректировка значений производится с клавиатуры. Общая форма этого оператора следующая:

```
EDIT [⟨зона замены⟩] [FIELDS ⟨список имен полей⟩]
     [⟨условие замены значения⟩]
     [⟨ограничение действия оператора⟩]
```

где ⟨зона замены⟩ определяет область файла, на которую распространяется присваивание значений полям записей (ALL, NEXT ⟨N⟩, REST, RECORD ⟨N⟩), а при отсутствии этой опции замена будет производиться для полей текущей записи. В операторе могут указываться опции условия замены вида FOR ⟨логическое выражение⟩, которое обеспечивает замену значений только для записей, удовлетворяющих условию, и ограничения вида WHILE ⟨логическое выражение⟩, которые обеспечивают замену значений, пока логическое выражение равно истине. В режиме экранного редактирования команды редактирования типа передвижения курсора, перехода к следующей или предыдущей записям, окончания редактирования и т. п. вызываются определенными комбинациями клавиш. При этом на экран выводится соответствующая подсказка для пользователя. В состав языка входит эквивалентный по структуре и функциям оператор CHANGE.

Для диалогового меню-ориентированного экранного редактирования используется также оператор BROWSE, имеющий следующий вид:

```
BROWSE [FIELDS ⟨список полей⟩] [LOCK ⟨числовое выражение⟩]
       [FREEZE ⟨поле⟩] [NOFOLLOW] [NOMENU]
       [NOAPPEND] [WIDTH ⟨числовое выражение⟩]
```

Этим оператором вызывается для редактирования до 17 смежных записей, располагаемых построчно. Список полей задает поля, выводимые для корректировки. Опция LOCK задает число смежных полей списка, которые фиксируются слева на экране и не участвуют в горизонтальном скролинге (перемещении), который обеспечивается для полей, не помещающихся в строке экрана. Опция FREEZE задает единственное поле, для которого разрешено редактирование. Опция NOFOLLOW используется при корректировке индексированного файла и обеспечивает сохранение порядка визуализируемых данных даже при изменении ключевого (участвующего в индексе) поля. В противном случае порядок вызванных для редактирования записей изменяется. Опция NOMENU позволяет отключить подсказывающее меню редактирования (для увеличения полезной площади экрана). Опция NOAPPEND блокирует возможность добавления новых записей при выполнении редактирования. Опция WIDTH задает ширину видимого изображения значения каждого поля. Если этот размер меньше ширины поля, то допускается горизонтальный скролинг значения поля.

Для автоматического изменения значений полей записи используется специальный оператор REPLACE следующего вида:


```
REPLACE [⟨зона замены⟩] ⟨имя поля 1⟩ WITH ⟨выражение 1⟩
        [, ⟨имя поля 2⟩ WITH ⟨выражение 2⟩ . . .]
        [ ⟨условие замены значения⟩ ]
        [ ⟨ограничение действия оператора⟩]
```

где ⟨зона замены⟩ определяет область файла, на которую распространяется присваивание значений полям записей. Далее в операторе следует список конструкций вида ⟨имя поля⟩ WITH ⟨выражение⟩, разделяемых запятой. В этих конструкциях указываются выражения, значения которых присваиваются указанным полям. В операторе также могут указываться опции условия замены вида FOR ⟨логическое выражение⟩ и ограничения вида WHILE ⟨логическое выражение⟩, например

```
REPLACE ALL surface WITH 'асфальт', termpr WITH ctod;
        ('20/06/87') FOR street = 'Садовая' .AND. region;
        ='Дарницкий'
```

В данной записи символ точка с запятой указывает на продолжение оператора на следующей строке. Этот оператор присваивает полю surface значение 'асфальт', а полю termpr значение даты '20/06/87' для записи с информацией об улице Садовой в Дарницком районе.

В п. 1.3.1 упоминалась команда исключения записи DELETE. В общем случае ее можно также рассматривать как команду манипулирования данными. Эта команда может иметь общие опции зоны действия, условия исключения и ограничения вида WHILE, при отсутствии которых команда исключает текущую запись. Отметим некоторые особенности манипулирования данными, связанные с организацией базы данных. По команде DELETE производится специальная отметка исключаемой записи, но физического удаления записи не происходит. При дальнейшей работе с файлом, имеющим исключенные таким образом записи, их «видимость» для различных средств визуализации записей определяется состоянием процессора. Оператор SET DELETED ON устанавливает процессор в состояние, при котором отмеченные записи доступны, как и неотмеченные. Оператор SET DELETED OFF делает отмеченные записи недоступными для обработки. Отмеченные записи можно восстановить оператором RECALL, имеющим структуру, аналогичную DELETE. В процессе работы можно использовать функцию deleted ( ), которая возвращает значение .T., если текущая запись отмечена как исключенная, и .F. в противном случае. При необходимости физического удаления исключенных записей из базы данных используется оператор PACK (без опций).

Если манипуляции такого вида производятся над индексированными файлами, то необходимо учитывать, что автоматические изменения индексных файлов, связанных с данным .dbf файлом, при добавлении, исключении или изменении ключевых полей записей производятся только в том случае, если файл данных открыт вместе со всеми индексными файлами. Как уже указывалось, это не всегда допустимо и, кроме того, может вызвать значительные затраты времени на реорганизацию индексных файлов, которая будет производиться после каждой изменяющей индексы корректировки данных. Для ускорения процесса можно использовать оператор REINDEX (без опций), который перестраивает открытые индексные файлы. В этом случае удобно произвести необходимые корректировки данных на файле данных, открытом без индексов, а затем переоткрыть файл вместе со всеми его индексами и применить оператор REINDEX.

В составе языка dBASE имеются удобные средства вычисления некоторых агрегатных функций для открытой базы данных. К ним относятся операторы AVERAGE, SUM и COUNT. Оператор AVERAGE служит для вычисления среднего арифметического некоторых выражений над полями записей по всему файлу с возможностью задания условий отбора суммируемых записей. Оператор имеет следующий вид:

AVERAGE ⟨список выражений⟩ [⟨зона вычисления⟩]
[⟨условие⟩] [⟨ограничение⟩]
[TO ⟨список переменных⟩]

Список выражений определяет поля (включая выражения над ними), по которым независимо ведется подсчет среднего значения. Зона вычисления, условие типа FOR ⟨. . .⟩ и ограничения типа WHILE ⟨. . .⟩ имеют уже известный смысл управления отбором записей файла данных для выполнения рассматриваемой операции, а список переменных используется для сохранения вычисленных средних значений соответственно порядку следования выражений.

Оператор SUM имеет такую же форму, но вычисляет сумму соответствующих выражений.

Оператор COUNT подсчитывает количество записей в файле данных и имеет следующую форму:

COUNT [⟨зона вычисления⟩] [⟨условие⟩]
[⟨ограничение⟩] [TO ⟨переменная⟩]

Значение количества записей, удовлетворяющих используемым опциям, присваивается указанной переменной.

1.3.3. Команды ввода и вывода значений

dBASE-машина оперирует понятиями логических внешних устройств в виде входного и выходного потоков данных, которые на практике назначаются на традиционные устройства ввода с клавиатуры и вывода на экран дисплея и по желанию на печатающее устройство. Поэтому будем отождествлять эти устройства с устройствами ввода — вывода dBASE-машины.

При планировании вывода на экран будем считать, что экран имеет 24 строки (с координатами 0—23) и 80 столбцов (позиций) (с координатами 0—79) по строке. dBASE-машина может использовать 0-ю строку для отображения состояния некоторых важных клавиш клавиатуры. Чтобы освободить эту строку для пользовательского вывода, необходимо изменить состояние машины оператором SET SCOREBOARD OFF. Оператор SET SCOREBOARD ON восстанавливает статус нулевой строки. 22-я строка экрана используется dBASE-машиной в качестве специальной статусной строки, в которой во время работы отображается информация об имени открытого файла, количестве записей, номере текущей записи и др. Для освобождения этой строки для пользовательского вывода необходимо изменить состояние машины оператором SET STATUS OFF Оператор SET STATUS ON восстанавливает статус 22-й строки.

Если используется цветной дисплей, то оператором SET COLOR ON устанавливается возможность работы в цвете. Установка цвета для различных объектов, визуализируемых на экране, производится оператором SET COLOR TO ⟨. . .⟩.

Вывод на экран может обеспечиваться одновременно с выводом на печать, если dBASE-машину перевести в режим печати оператором SET PRINT ON. SET PRINT OFF отключает печатающее устройство от выводного потока. Для вывода на печать используются и другие средства, списываемые ниже.

Для очистки экрана используется оператор CLEAR (без дополнительных опций). Курсор устанавливается в начало экрана (т. е. в нулевую позицию первой доступной строки).

Простейший оператор вывода вида

? ⟨список выражений⟩

был рассмотрен ранее. Такой оператор выводит значения в строке экрана, следующей за строкой предыдущего вывода, а если это последняя строка, то вывод вызывает смещение предыдущих строк на одну строку вверх (так называемый вертикальный скролинг). Другая форма этого оператора

?? ⟨список выражений⟩

вызывает вывод в текущей строке.

Рассмотрим наиболее универсальный оператор ввода— вывода языка. Он имеет следующую форму:

@⟨строка⟩, ⟨столбец⟩ [[SAY ⟨выражение⟩ [PICTURE ⟨шаблон⟩]] [GET ⟨имя переменной или поля записи⟩ [PICTURE ⟨шаблон⟩] [RANGE ⟨минимальное значение⟩, ⟨максимальное значение⟩]]] /CLEAR

Этот оператор в простейшей форме @ ⟨строка⟩, ⟨столбец⟩ переводит курсор на указанную позицию и очищает строку с позиции, начиная с указанного столбца.

Другая простая форма @ ⟨строка⟩, ⟨столбец⟩ CLEAR очищает правую нижнюю часть экрана начиная с указанных координат.

Координаты номера строки и столбца могут задаваться произвольными численными выражениями. Имеются следующие функции определения текущей позиции курсора на экране:

row ( ) — значение текущей строки экрана,
col ( ) — значение текущей позиции строки экрана.

Таким образом, оператор

@row ( ) + 1, col ( ) + 1 CLEAR

очистит правую нижнюю часть экрана от текущей позиции курсора со смещением на одну строку вниз и один столбец вправо.

Оператор @ имеет две основные внутренние конструкции SAY ⟨. . .⟩ и GET ⟨. . .⟩. Первая конструкция служит для вывода, а вторая — для ввода. Их совместное использование позволяет обеспечить ввод данных с некоторой подсказывающей информацией, выводимой на экран перед вводом.

В конструкции SAY задается выражение, значение которого выводится на экран начиная с заданной позиции и, возможно, шаблон вывода в виде конструкции PICTURE ⟨шаблон⟩. В конструкции GET задается имя переменной или поля записи, значение которого вводится с клавиатуры, и, возможно, шаблон ввода в виде PICTURE ⟨шаблон⟩. Если задается конструкция ввода, то физический ввод запрашивается отдельным оператором READ. При этом READ запрашивает ввод для переменных (полей записи) всех предыдущих операторов @ (до предыдущего READ). Например,

CLEAR
@ 2,20 SAY 'ввод данных об улицах'

```
@ 3,5 SAY 'введите:'
@ 4, 5 SAY 'район:          ' GET region
@ 5,5 SAY 'улица:           ' GET street
@ 4,5 SAY 'протяженность:   ' GET length
@ 4,5 SAY 'покрытие:        ' GET surface
READ
```

вызовет вывод на экран соответствующего формата ввода (подсказывающих наименование полей и буферных позиций для ввода) и установку курсора в начало поля ввода в четвертой строке в ожидании ввода с клавиатуры соответствующих значений. При вводе с клавиатуры обеспечивается режим экранного редактирования, т. е. возможность движения и корректировки во всех заданных полях.

Конструкции PICTURE ⟨. . .⟩ используются для форматирования и контроля выводимых и вводимых данных. Понятие ⟨шаблон⟩ включает комбинацию некоторых функциональных кодов, управляющих выводом, и шаблонных литер, контролирующих данные. К функциональным кодам относятся следующие:

C — вывод знака кредита после положительных чисел
X — вывод знака дебета после отрицательных чисел
(— заключение отрицательных чисел в скобки
B — выравнивание чисел слева
Z — вывод нулевых значений в виде строки пробелов
D — вывод дат в американском формате (мм/чч/гг)
E — вывод дат в европейском формате (чч/мм/гг)
A — вывод только алфавитных символов
! — вывод любых символов с переводом букв в прописной регистр
R — указание на то, что символы в строке шаблонных литер, не являющиеся специальными кодами, не являются частью входного потока, а только визуализируются (см. ниже)
S⟨n⟩ — ограничение видимого поля вводимой переменной ⟨n⟩ символами с горизонтальным перемещением (скролингом) данных этого поля при вводе или коррекции

Функциональные коды могут объединяться, если это логически непротиворечиво, например, XC вызовет вывод обоих знаков, задаваемых каждым кодом.

Шаблонные литеры (template) образуют строку, используемую для управления и контроля при вводе — выводе. Управление и контроль заключаются в том, что каждый символ шаблонной строки соотносится с вводимыми или выводимыми символами для соответствующей позиции. Следующие символы шаблонной строки имеют специальное значение:

9 — в данной позиции допускается только цифра или знак числа (контроль числовых данных)
— допускается только цифра, пробел или знак числа
A — допускаются только буквы
L — допускаются только логические данные
Y — допускаются только логические данные в форме Y, y, N, n
N — допускаются только буквы и цифры
X — допускается любой символ
! — преобразование букв в прописной регистр
$ — вывод знака доллара вместо ведущих (левых) нулей
* — вывод знака звездочки вместо ведущих нулей
. — позиция десятичной точки для чисел
, — вывод знака запятой, если слева от запятой есть цифры.

Конструкция ⟨шаблон⟩ записывается в кавычках '. . .' , при этом перед функциональными кодами ставится знак @. Если в шаблоне одновременно используются функциональные коды и шаблонные строки, то они разделяются пробелом. Любой, отличный от специальных, символ в строке шаблона становится частью входного потока (т. е. автоматически вставляется в заданную позицию), если не указан функциональный код @R. Например,

```
CLEAR
@, 3, 3 SAY numberl PICTURE '@X 99999'
temp='123456789'
@, 4, 3 GET temp PICTURE '@R##t####'
  , 5, 3 SAY 'ввод числа' GET number 2 PICTURE '9999'
READ
```

Первый оператор @ выведет в 3-й строке значение числового поля (переменной) numberl в виде пятисимвольной строки с выравниванием числа по правой границе. Если число имеет отрицательное значение, то за ним будут выведены буквы DB (дебет). Следующий оператор @ поместит в 4-й строке буфер поля вводимого значения переменной temp длиной 7 позиций с видимой частью текущего значения и вставленной буквой t в третьей позиции, т. е. 12t3456. Последующий оператор выведет в 5-й строке сообщение 'ввод числа' и сформирует буфер длиной 4 позиции для ввода значения переменной number 2. Предполагается, что переменные numberl и number2 определены ранее. Оператор READ вызовет физический запрос ввода путем установки курсора в первую позицию буфера ввода переменной temp. При вводе в этом поле позиция буквы t блокируется для ввода, но в формировании нового значения не участвует, поскольку задан функциональный код R. Так, при вводе значений в этом поле, например '98t7654', окончательное значение переменной будет '987654789'. В качестве вводимых символов шаблон этого оператора допускает цифры, пробелы и знаки + или —. Для вводимого значения переменной number2 заданный шаблон допустит только 4 символа цифр и знаков + или —.

В рассматриваемом операторе @ в конструкции ввода может быть также предусмотрен дополнительный контроль диапазона значений вводимых числовых данных с помощью опции RANGE ⟨минимальное значение⟩, ⟨максимальное значение⟩.

Неправильный ввод данных сопровождается звуковым сигналом. dBASE-машина может быть установлена в состояние отключения звукового сигнала оператором SET BELL OFF. Оператор SET BELL ON восстанавливает состояние звуковой сигнализации.

При использовании оператора @ для цветных дисплеев соответствующим полям можно задать отдельные цвета с помощью упомянутого ранее оператора SET COLOR TO, который имеет следующую форму:

SET COLOR TO [⟨цвет вывода⟩[, ⟨цвет ввода⟩] [, ⟨рамка⟩ [ ]

Конструкции ⟨цвет вывода⟩ и ⟨цвет ввода⟩ задаются в виде пар буквенных кодов цветов, разделенных знаком /. Буквенные коды следующие:

N — черный	B — синий	G — зеленый
BG — циан	X — пустой (невидимый)	R — красный
RB — фиолетовый	GR — коричневый	W — белый

Знак * вызывает мерцание, а + — повышенную яркость. Первый код обозначает цвет выводимых символов, а второй — цвет фона

экрана. Например, N/GR+ означает вывод черных символов на желтом (ярко-коричневом) фоне. Задание цвета вывода означает соответствующую раскраску выводимых операторами вывода на экран данных, а цвет ввода определяет раскраску буферных полей ввода, задаваемых соответствующими операторами ввода. Конструкция ⟨рамка⟩ задает цвет обрамляющей части экрана (вне 24 × 80 знакомест). Так, если рассмотренной последовательности операторов ввода — вывода предшествовал оператор

SET COLOR TO W/B, N/G, R

то экран будет окрашен в синий цвет с красным обрамлением, сообщения о вводе с наименованиями полей выведутся белым цветом, буферные поля для ввода будут зелеными, а вводимые с клавиатуры значения будут черными.

При выводе данных оператором @ на экран дисплея автоматически контролируется соответствие начальных заданных координат и длины выводимых строк размерам экрана. Информация, не помещающаяся в строке, при наличии доступных нижеследующих строк переносится в следующую, а при некорректных условиях вывода на экран выводится соответствующее диагностическое сообщение.

При выводе на устройство печати значение координат может быть в пределах 0—255. При этом выводной поток может быть переключен с экрана на печать оператором SET DEVICE TO PRINT. Оператор SET DEVICE TO SCREEN переключает выводной поток на экран. Выполнение оператора @ для устройства печати в сравнении с дисплеем имеет некоторые естественные отличия. Так, посылка следующего @-оператора с номером строки, меньшим, чем в предыдущем операторе, вызовет перевод страницы; опция GET не переключается и сохраняется для интерпретации на экране.

Имеется вариант оператора @, позволяющий выделять на экране некоторые окна, для которых можно планировать операции ввода — вывода, включая возможности горизонтального скролинга информации в окнах. Оператор организации окон имеет следующий вид:

@⟨строка 1⟩, ⟨столбец1⟩ [CLEAR] TO ⟨строка2⟩, ⟨столбец2⟩ [DOUBLE]

По этому оператору на экране рисуется прямоугольное окно, обрамленное одинарной или двойной (при указании DOUBLE) линией. Координаты левого верхнего и правого нижнего углов окна задаются соответствующими парами ⟨строка⟩, ⟨столбец⟩. Опция CLEAR обеспечивает очистку окна.

К группе команд ввода — вывода относится оператор ввода символьных данных следующего вида:

ACCEPT [⟨сообщение⟩] TO ⟨переменная⟩

Данный оператор выводит заданное сообщение в текущей строке экрана и требует ввода с клавиатуры символьного значения, присваиваемого заданной переменной. Сообщение может быть опущено.

Мощным средством вывода форматированной информации из базы данных является оператор REPORT (генератор отчетов). Данный оператор использует дополнительно специальный тип файлов (с расширением .frm), содержащий информацию о формате отчета.

1.3.4. Служебные команды

К служебным командам относятся команды управления состоянием машины dBASE III Plus, включающие группу команд (операторов) установки SET, обслуживания файлов и др., а также используемые для других системных целей, в частности для сопряжения работы dBASE-машины с другими программно-аппаратными средствами ПЭВМ. Полное их описание приводится в следующей главе.

1.4. Базовые средства программирования

dBASE-машина ориентирована на диалоговый режим решения задач, однако включает все необходимые языковые средства программирования различных приложений аналогично универсальным системам программирования. Программы для dBASE-машины представляют собой текстовые файлы со стандартным расширением .prg, в которых записывается требуемая последовательность операторов языка dBASE III Plus. Формирование и корректировка этих программных файлов производятся командой

MODIFY COMMAND ⟨имя файла⟩

По этой команде вызывается текстовый редактор, с помощью которого можно вводить и редактировать текст программы, при этом dBASE-машина допускает подключение произвольных текстовых редакторов. Выполнение программного файла осуществляется командой

DO ⟨имя файла⟩

При этом эффективное использование языка связано с квалифицированной проекцией предметной области решаемой задачи на структуры данных и управляющие структуры языка dBASE III Plus аналогично понятиям, принятым в традиционном программировании. Это касается, в частности, программирования линейных, разветвляющихся и циклических участков программ, а также подпрограмм.

Линейные участки образуются как последовательности операторов, выполнение которых производится в порядке их записи.

Для программирования разветвлений и циклов используются конструкции, которые не могут быть использованы в диалоговых режимах. Разветвляющиеся программы на dBASE используют две основные конструкции для программирования условий. Первая соответствует традиционному условному оператору и имеет следующий вид:

```
IF ⟨логическое выражение⟩
   ⟨последовательность операторов 1⟩
[ELSE
   ⟨последовательность операторов 2⟩]
ENDIF
```

Выполнение приведенной конструкции заключается в вычислении логического выражения, значение которого управляет дальнейшим исполнением операторов. Если значение логического выражения истинно, то выполняется ⟨последовательность операторов 1⟩, в противном случае — ⟨последовательность операторов 2⟩ (которая может отсутствовать). Эти последовательности операторов в свою очередь могут включать рассматриваемую условную конструкцию.

Другой возможностью программирования разветвлений является переключательная конструкция следующего вида:

```
DO CASE
    CASE ⟨логическое выражение 1⟩
         ⟨последовательность операторов 1⟩
    CASE ⟨логическое выражение 2⟩
         ⟨последовательность операторов 2⟩
    . . .
    CASE ⟨логическое выражение n⟩
         ⟨последовательность операторов n⟩
    [OTHERWISE
         ⟨последовательность операторов n + 1⟩]
ENDCASE
```

Выполнение ее производится путем последовательного вычисления заданных логических выражений до тех пор, пока не будет найдено первое истинное значение. После этого выполняется соответствующая 1-я последовательность операторов и управление передается на оператор, следующий за ENDCASE. Если все выражения не истинны, то выполняются операторы, записанные после OTHERWISE (последние могут отсутствовать). Исполняемые последовательности операторов могут содержать внутренние условные операторы вида IF и DO CASE.

Программирование циклических участков производится с помощью следующей конструкции:

```
DO WHILE ⟨логическое выражение⟩
    ⟨последовательность операторов⟩
ENDDO
```

Заданная последовательность операторов выполняется до тех пор, пока вычисляемое на каждой итерации логическое выражение истинно. В частности, если при первоначальном вычислении логическое выражение ложно, то цикл не выполняется и управление передается на оператор, следующий за ENDDO.

Для программирования циклов дополнительно могут использоваться следующие операторы. Оператор

```
EXIT
```

вызывает прекращение цикла путем передачи управления на оператор, следующий за ENDDO. Оператор

```
LOOP
```

вызывает переход на следующую итерацию цикла, т. е. на очередное вычисление логического выражения — условия цикла.

В dBASE отсутствует явный оператор безусловного перехода по метке, т. е. в произвольную точку программы, что в определенной мере вынуждает программировать в стиле структурного программирования.

Язык dBASE обеспечивает также средства структуризации программ путем разбиения их на некоторые подпрограммы аналогично традиционным приемам программирования. Подпрограмма может быть отдельным программным файлом, аналогичным главной программе (т. е. той, которая вызывается из диалогового режима командой DO), при этом вызов такой подпрограммы производится из вызывающей программы тем же оператором DO ⟨имя файла⟩. Возврат

управления в вызывающую программу осуществляется при достижении конца файла подпрограммы либо при выполнении оператора

```
RETURN
```

Связь по данным между вызывающей и вызываемой программами осуществляется следующим образом. При переходе к подпрограмме состояние памяти и процессора dBASE-машины сохраняется, т. е. доступны открытые файлы данных и определенные ранее переменные. Другими словами, выполнение подпрограммы производится так, как будто она текстуально содержится в вызывающей программе. Однако переопределение типов существующих переменных или определение новых переменных (операторами присваивания) в подпрограмме делает их недоступными в вызывающей программе.

Второй способ организации подпрограмм близок к реализациям этого механизма в универсальных языках программирования. Подпрограммы этого типа называются процедурами, и их тексты помещаются в процедурный файл, также имеющий стандартное расширение . prg. В один процедурный файл помещается одна или несколько процедур, имеющих следующую структуру:

```
PROCEDURE ⟨имя процедуры⟩
    [PARAMETERS ⟨список параметров⟩]
      ⟨последовательность операторов⟩
RETURN
```

Как видно из структуры, процедура может иметь параметры, с помощью которых можно передавать скалярные данные. Вызову процедуры должен предшествовать оператор

```
SET PROCEDURE TO ⟨имя процедурного файла⟩
```

По данному оператору становятся доступными процедуры, записанные в указанный файл. Дальнейший вызов осуществляется оператором

```
DO ⟨имя процедуры⟩ [WITH ⟨список аргументов⟩]
```

Список аргументов соответствует списку параметров и может включать выражения соответствующих типов. При задании аргументов в виде выражений их значения вычисляются и передаются соответствующим параметрам. Если в качестве аргументов задаются имена переменных, то при вызове они заменяют параметры по именам, что позволяет передавать некоторые значения из процедур в вызывающую программу.

В dBASE-машине предусмотрено также использование процедур, написанных на других языках программирования.

1.5. Дополнительные средства программирования

Список команд и функций dBASE приводится в следующих главах. Рассмотрим некоторые дополнительные средства dBASE-машины, расширяющие возможности описанных базовых средств. Эти возможности связаны с организацией некоторых «надстроек», которые специализируют dBASE-машину для конкретных приложений, значительно упрощая взаимодействие конечного пользователя с dBASE-машиной.

1.5.1. Генератор отчетов

Для заданной базы данных вывод информации на экран дисплея или на печать может быть форматизован специальным способом, при котором можно получить данные в виде некоторого отчета. Формат вывода записывается в отдельный файл, имеющий стандартное расширение .frm, и в дальнейшем используется в операторе вывода следующего основного вида:

REPORT FORM ⟨имя .frm файла⟩ [⟨зона действия⟩]
[FOR ⟨условие⟩] [WHILE ⟨ограничение⟩]

Для подготовки .frm файла должен быть открыт необходимый файл данных и использована команда

CREATE REPORT ⟨имя .frm файла⟩

По этой команде dBASE-машина переходит в меню-ориентированный микродиалоговый режим задания формы отчета, в котором определяются основные размеры выходного документа, его постраничное разбиение, условия вывода на печать, а также используемые поколонно поля базы данных (или допустимые выражения над этими полями), их ширина на плоскости отчета, наименования «шапок». В задание формата вывода отчета могут входить также указания о группировках отдельных строк с выводом соответствующих итогов и подытогов.

Последующие модификации созданного файла можно производить по команде

MODIFY REPORT ⟨имя .frm файла⟩

по которой обеспечиваются те же действия, что и по команде CREATE.

Оператор REPORT может использовать конструкции условий типа FOR и/или WHILE аналогично рассмотренным ранее операторам, а также ряд дополнительных опций, рассмотренных в следующей главе.

1.5.2. Фильтры

Во многих рассмотренных ранее командах dBASE-машины употребляются конструкции условий типа FOR, которые можно назвать горизонтальным фильтром базы данных, поскольку они допускают заданные командой манипуляции только над теми записями, которые удовлетворяют условиям. При необходимости эти условия фильтрации можно записать в специальный файл со стандартным расширением .qry и в дальнейшем не указывать условия явно, а фактически сделать видимой только указанную часть базы данных.

Создание .qry файла для открытой базы данных в меню-ориентированном микродиалоге производится по команде

CREATE QUERY ⟨имя .qry файла⟩

При этом условие формируется в виде последовательности строк отношений типа

⟨имя поля⟩ ⟨операция⟩ ⟨константа⟩ [⟨логическая операция⟩]

Имя поля выбирается из предлагаемых имен полей открытого .dbf файла. Операция выбирается из набора допустимых для выбранного типа поля операций (сравнения, поиска подстроки и т. п.).

Константа вводится в соответствии с типом значения выбранного поля, а логическая операция типа .AND., .OR. и др. используется, если формируется сложное условие. Допускается формирование скобочных выражений типа

```
(num1   <   10   .OR.
 num1   >    0 )  .AND.
 name2  =  'Иванов'
```

В целях отладки фильтра предусмотрен процесс его проверки путем пробного вывода данных без выхода из команды CREATE. Последующую модификацию .qry файла можно произвести командой

MODIFY QUERY ⟨имя .qry файла⟩

которая повторяет режим команды CREATE QUERY.

При наличии файла фильтра его активизация производится командой

SET FILTER TO FILE ⟨имя .qry файла⟩

По этой команде видимыми для дальнейших манипуляций становятся только те записи файла данных, которые удовлетворяют записанному условию. Отмена режима фильтрации производится при переходе к другому фильтру либо по команде отмены

SET FILTER TO

Кроме рассмотренного горизонтального фильтра может быть установлен также вертикальный фильтр, заключающийся в указании доступных полей базы данных. Режим вертикального фильтра устанавливается двумя следующими командами:

SET FIELDS ON/OFF и
SET FIELDS TO ⟨список полей⟩

Первая команда разрешает (ON) или отменяет (OFF) использование режима вертикального фильтра, а вторая специфицирует поля, которые становятся доступными.

1.5.3. Формирование экранных форм

При конструировании диалога между пользователем и dBASE-машиной удобным средством программирования обмена является оператор @. . . SAY. . . GET. . . , который позволяет спланировать экранную форму диалога, т. е. расположение вводимых полей данных и соответствующее обрамление этих полей подсказывающей информацией, графическими элементами типа окон и пр. Для упрощения процесса программирования такого диалога в dBASE-машине предусмотрены средства создания специальных файлов, которые сохраняют форму экрана, а также форматные проверки данных при вводе — выводе и легко вызываются, заменяя группы команд @. . . SAY . . . GET . . . Создание этих файлов также производится в отдельном микродиалоге с помощью команды

CREATE SCREEN ⟨имя .scr файла⟩

Корректировка файла производится командой

MODIFY SCREEN ⟨имя .scr файла⟩

При выполнении этих команд допускается также создавать новые и/или корректировать существующие файлы данных (.dbf файлы), готовить документирующие тексты и т. п., что позволяет достаточно быстро создавать несложные, но полные приложения. При завершении формирования экранной формы фактически создаются два файла с одинаковым именем и расширениями .scr и .fmt. Второй файл содержит текст программы на языке dBASE, обеспечивающей заданный ввод и вывод, поэтому этот текст может быть переписан в программный файл (.prg) и дополнен другими командами.

Использование экранных форм (.scr и .fmt файлов) производится с помощью оператора

SET FORMAT TO [⟨имя .fmt файла⟩]

1.5.4. Связь файлов данных

Модель памяти dBASE-машины допускает одновременную работу с несколькими (не более десяти) файлами данных, открываемыми для доступа в соответствующих рабочих зонах. Однако изменение текущих записей в каждой из рабочих зон производится индивидуально и требует ряда вспомогательных действий по переключению рабочих зон (операторами SELECT). В то же время база данных, состоящая из нескольких файлов данных, может быть сконструирована таким образом, что информация, размещаемая в одном файле, логически связана с информацией другого файла. При этом желательно, чтобы переход к требуемой записи в одном файле вызывал изменение указателя текущей записи другого, связанного с данным, файла. Подобная связь файлов данных реализуется в dBASE-машине с помощью понятия отношения.

Для создания отношения в виде связи двух файлов используется оператор

SET RELATION TO [⟨ключ⟩] INTO ⟨рабочая зона⟩

Этот оператор применяется для некоторого открытого файла данных, который назовем ведущим. В другой рабочей зоне открыт файл данных, имеющий логическую информационную связь с ведущим. Существует две основные формы подобной связи. В первом случае ведущий файл содержит поле, по которому проиндексирован связанный с ним подчиненный файл (предполагается, что в выражение для индекса подчиненного файла входит имя указанного поля ведущего файла). Смысл связи заключается в том, что изменение указателя записи ведущего файла приводит к автоматической установке указателя связанного файла на позицию, соответствующую значению ключевого поля ведущего файла. Подобное ключевое (для связи) поле указывается в рассматриваемом операторе в позиции ⟨ключ⟩. Опция ⟨рабочая зона⟩ используется для указания (в виде букв A, B, C и т. д.) зоны, в которой открыт связываемый файл.

Во втором случае в качестве ключа указывается числовое выражение. При этом связываемый файл не должен иметь индекса. Значение этого выражения используется для связи по номеру записи. Таким образом, при изменении указателя записи ведущего файла связанный файл устанавливается на запись с номером, равным вычисленному выражению.

Для многократного использования рассмотренных «надстроек» над базовыми средствами dBASE-машины можно сохранить соответствующие структуры в специальном файле, имеющем расширение .vue. Создание подобного файла производится командой

CREATE VIEW ⟨имя .vue файла⟩

По команде

MODIFY VIEW ⟨имя .vue файла⟩

производится корректировка существующего .vue файла.

По указанным командам осуществляется переход в меню-ориентированный микродиалог формирования ряда условий «видимости» базы данных. К этим условиям относятся: выбор файлов данных с возможными индексами для организации цепочек связей аналогично оператору SET RELATION, выбор горизонтального (как файлом .qry) и вертикального (как SET FIELDS) фильтров, а также определение используемого форматного (.fmt) файла для ввода — вывода. После создания такого файла команда

SET VIEW TO ⟨имя .vue файла⟩

переводит dBASE-машину в состояние, при котором доступной для обработки последующими операторами базой данных будет описанная во .vue файле часть .

1.5.5. Обработка этикеток

В dBASE III Plus предусмотрены средства создания специальных форматных файлов (.lbl файлов) для хранения и использования информации типа списка адресов, организованной в виде этикеток (меток). Форматированная информация (метка) представляет собой совокупность строк (до 16), в каждой из которых (длиной до 120 символов) может помещаться dBASE-выражение, использующее поля открытых файлов данных и, возможно, связанное с какими-нибудь текстами (символьными константами). Вывод их на печать может производиться для специальных бланков (например, конвертов) или с целью последующей разрезки и наклейки на другие документы. При этом на поле печати можно удобно разместить несколько таких меток. Создание подобных файлов осуществляется командой

CREATE LABEL ⟨имя .lbl файла⟩

а модификация — командой

MODIFY LABEL имя .lbl файла⟩

По этим командам происходит переход в меню-ориентированный режим, в котором задается формат этикетки (количество строк, длина строки, количество этикеток на страницу и др.), а затем для каждой из строк формируются выражения, в которые могут входить данные открытых файлов данных.

Для вывода этикеток служит оператор, имеющий следующий основной вид:

LABEL FORM ⟨имя .lbl файла⟩ [⟨зона действия⟩]
[FOR ⟨условие⟩] [WHILE ⟨ограничение⟩]

1.5.6. Каталогизация файлов

В общем случае создаваемые dBASE-машиной файлы помещаются при отсутствии явных указаний в каталог диска, в котором размещены файлы программ, реализующих саму dBASE-машину.

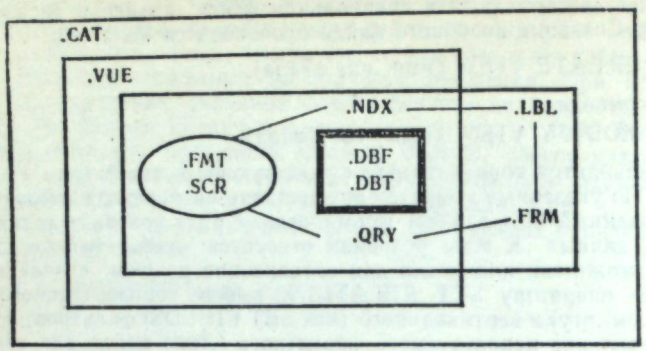


Рис. 2. Взаимосвязь файлов.

При работе с несколькими приложениями это может привести к неудобствам, связанным с хранением в одном каталоге слишком большого числа файлов, и даже к ошибкам, вызванным коллизией совпадающих имен файлов. С другой стороны, законченное приложение, использующее файлы dBASE-машины, удобно оформить как нечто целое с учетом логической взаимосвязи файлов.

Для этих целей предусмотрен специальный тип файла данных, однако имеющий стандартное расширение .cat. Так, например, взаимосвязь файлов реальной базы данных можно представить следующим образом (рис. 2). Файл каталога содержит информацию о взаимосвязи ведущего файла данных (.dbf, .dbt) с его вспомогательными или непосредственно подчиненными файлами (.fmt, .scr, ndx, .lbl, .frm, .qry), а также с другими файлами, связь с которыми определена через .vue файл. Структура каталога включает основной мастер- файл, создаваемый автоматически при создании .cat файлов, который содержит ссылки на пользовательские каталоги. Для создания каталога приложения используется команда

SET CATALOG TO ⟨имя .cat файла⟩

В общем случае эта команда закрывает предыдущий открытый каталог и создает новый (или открывает существующий под тем же именем). В дальнейшем команды CREATE/MODIFY для перечисленных типов связанных файлов, а также команды SET FILTER, SET FORMAT, SET VIEW и некоторые другие автоматически корректируют содержимое каталога, делая «видимыми» только файлы соответствующего приложения. Файл-каталог может быть открыт так же, как файл данных, командой USE ⟨имя .cat файла⟩ и быть доступен для обработки основными средствами dBASE, например командой EDIT.

1.6. Компилятор Clipper

Как указывалось в п. 1.1, СУБД dBASE III Plus имеет специальный компилятор Clipper, с помощью которого можно получать законченные приложения, написанные на языке Clipper, практически совместимом с языком dBASE, что позволяет после удобной диалоговой отладки программ в интерпретирующей среде dBASE скомпилировать программы и получить исполнительный програм-

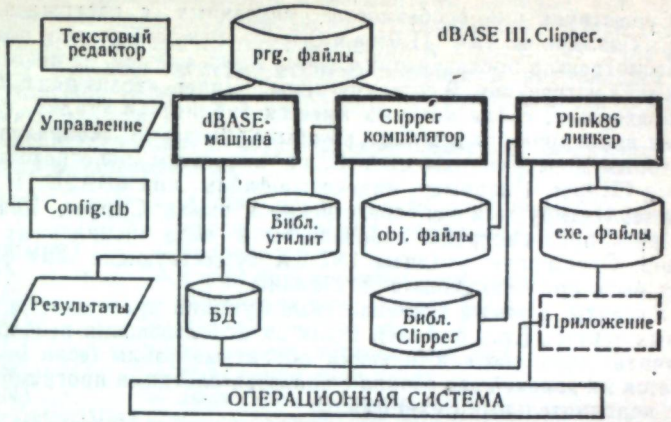


Рис. 3. Технология использования инструментального комплекса dBASe III Plus, Clipper.

мный модуль для операционной системы, независимый от dBASE-интерпретатора.

Общий цикл проектирования и создания приложений заключается в создании и отладке диалоговыми средствами dBASE базы данных (в смысле совокупности файлов, рассмотренных в предыдущем разделе) с последующей разработкой различных программ обработки данных. Средства dBASE для многих приложений позволяют ограничиться сопровождением базы и выводом информации по разнообразным запросам базовыми средствами, однако эти средства ориентированы на англоязычного конечного пользователя. Кроме того, база данных может содержать информацию, обработка которой требует специальных программ, создаваемых профессиональными программистами. Например, для организации ввода и корректировки информации с русскоязычным интерфейсом необходимы специальные программы. Поэтому будем считать, что законченное приложение включает кроме файлов базы данных специальные программы (.prg файлы). При использовании dBASE-машины эти файлы интерпретируются путем вызова их командой DO ⟨имя .prg файла⟩. Трансляция их с помощью компилятора Clipper позволяет получить исполнительные модули уровня операционной системы. На рис. 3 показана технология использования инструментального комплекса dBASE, Clipper.

Исходные программы (.prg файлы) проектируются и отлаживаются с применением традиционных технологий программирования (структурного, модульного и т. п.). После отладки в среде dBASE они транслируются компилятором Clipper, который создает объектные модули с именем исходного .prg файла и с расширением .obj. При этом если транслируемый модуль вызывает другие .prg файлы (DO ⟨имя .prg файла⟩ SET PROCEDURE TO ⟨имя .prg файла⟩), то последние автоматически подключаются к процессу трансляции и входят в состав объектного модуля. Полученные объектные модули (порознь протранслированные) связываются с помощью программы PLINK86 (линкер) в один исполнительный модуль с именем главной программы (первого объектного модуля в списке линкера) и расширением .exe. Этот модуль вызывается на исполнение операционной системой и не зависит от среды dBASE.

Существует ряд особенностей, связанных с расхождениями входных языков dBASE III Plus и Clipper. Эти особенности подробно рассмотрены в последующих главах. Здесь укажем на некоторые приемы их устранения. В состав системы Clipper входит ряд библиотек поддержки, среди которых имеется библиотека утилит, содержащая процедуры, написанные на языке Clipper и эквивалентные некоторым конструкциям dBASE, не входящим непосредственно в язык Clipper. Например, индексные файлы .ndx dBASE III Plus не интерпретируются непосредственно в языке Clipper, для них в Clipper предусмотрены эквиваленты в виде специальных .ntx файлов. С помощью утилиты INDEX существующие .ndx файлы могут быть преобразованы в .ntx файлы.

Утилиты, а также эквивалентные функции представлены в исходных текстах (.prg файлах), и для их использования необходимо получить либо соответствующие объектные файлы (если предполагается их дальнейшая сборка с пользовательскими программами), либо исполнительные (.exe) файлы.

В тех случаях, когда не удается достичь полной совместимости исходных программ для эквивалентного исполнения в среде dBASE и Clipper, можно использовать специальную глобальную переменную clipper, вводимую оператором

```
PUBLIC clipper
```

которая имеет логический тип и распознает среду исполнения. Можно запрограммировать разветвление по этой переменной для соответствующих конструкций языков, имеющих одинаковую семантику, но различный синтаксис, например

```
IF  clipper
    ⟨операторы языка Clipper⟩
ELSE
    ⟨операторы языка dBASE⟩
ENDIF
```

Глава 2

ОПИСАНИЕ КОМАНД

2.1. Соглашения по нотации

Как указывалось в п. 1.1, в качестве терминов, используемых при описании реляционных баз данных, поддерживаемых СУБД dBASE III Plus, употребляются понятия файла данных и эквивалентного понятия отношения, записи файла данных и кортежа отношения, поля данных файла данных и атрибута отношения. Первый набор терминов введен разработчиками СУБД dBASE III Plus, а второй относится к стандартной терминологии реляционной модели данных. В приведенном ниже описании команд используется реляционная терминология. Такой выбор позволяет более явно разделить реляционные операции и файловые манипуляции в пояснениях к командам.

В гл. 1 дано общее описание синтаксиса языка dBASE, его структур и типов данных, а также проведена классификация команд.

Ниже описаны команды в алфавитном порядке. По каждой команде последовательно приводятся: наименование команды с указанием возможности ее использования в dBASE III Plus и/или Clipper; краткое пояснение; строгий синтаксис в оригинальной нотации; описание использования; ссылка на группу семантически родственных команд.

В синтаксических конструкциях используются следующие обозначения метапеременных:

⟨clause⟩	— фраза, используемая для описания формата ввода — вывода в виде символьной строки
⟨condition⟩	— условие
⟨exp⟩	— выражение произвольного типа
⟨expC⟩	— выражение символьного типа
⟨expD⟩	— выражение типа дата
⟨expL⟩	— выражение логического типа
⟨expM⟩	— выражение типа текст
⟨expN⟩	— выражение числового типа
⟨expression list⟩	— список выражений произвольных типов
⟨field⟩	— атрибут
⟨field list⟩	— список атрибутов
⟨filename⟩	— имя файла
⟨filetype⟩	— тип файла
⟨memvar⟩	— переменная памяти произвольного типа (синоним ⟨var⟩)
⟨memvar list⟩	— список переменных памяти
⟨prompt⟩	— выдаваемое сообщение
⟨record⟩	— кортеж
⟨row, col⟩	— номера строки и столбца позиции вывода при описании вывода на экран или принтер
⟨scope⟩	— зона действия
⟨string⟩	— символьная строка
⟨type⟩	— тип
⟨var⟩	— переменная памяти произвольного типа (синоним ⟨memvar⟩)

2.2. Список команд

2.2.1. ?/?? (dBASE III Plus и Clipper)

?/?? вычисляет и выводит значения списка выражений. ?/? интерпретируется как "Чему равно . . . ?" или "Каково значение . . . ?"

Синтаксис. ?/?? ⟨expression list⟩

Использование. Одиночный знак ? предполагает вывод с новой строки, а ?? — с текущей. ?? используется для вывода списка выражений с текущего положения курсора или головки принтера.

2.2.2. @ ... SAY ... GET ... (dBASE III Plus и Clipper)

Используется для описания форм ввода и вывода информации. Производит ввод — вывод информации в заданном формате по указанным координатам.

Синтаксис. Для dBASE III Plus:

@⟨row, column⟩ [SAY ⟨exp⟩ [PICTURE ⟨clause⟩]]
[GET ⟨var⟩ [PICTURE ⟨clause⟩]
[RANGE ⟨exp⟩, ⟨exp⟩]/[CLEAR]

Для Clipper:

@⟨row, column⟩ [SAY ⟨exp⟩ [PICTURE ⟨clause⟩]]
[GET ⟨exp⟩ [PICTURE ⟨clause⟩]
[RANGE ⟨exp⟩, ⟨exp⟩] [VALID ⟨exp⟩]
[CLEAR]

Использование. Row и Column — числовые выражения. Для терминала 24 * 80 строка (row) может варьировать от 0 до 22, а столбец (column) — от 0 до 79. В dBASE III Plus строка 0 обычно резервируется как статусная строка, а 22-я — как строка индикации. Для освобождения этих строк под другое использование в dBASE III Plus необходимо дать команды SET STATUS OFF и SET SCOREBOARD OFF.

Для направления потока @. . . SAY . . . на принтер следует воспользоваться командой SET DEVICE TO PRINT, при этом @ . . . GET . . . не направляются на принтер. При выводе на принтер значения row и column не должны превышать 255. При выводе на принтер последовательности @ команд появление в последовательности команды с параметром row, меньшим, чем row предыдущей команды, ведет к переводу страницы.

@ row, col CLEAR очищает экран ниже и правее заданных координат.

@ row, col очищает строку с заданной позиции.

В dBASE III Plus команды CREATE/MODIFY SCREEN могут быть использованы для автоматического создания форматных файлов, содержащих наборы команд @ . . . SAY . . . GET . . . для воспроизведения конструируемых экранных форм.

Опции. Опция SAY выводит определенную информацию. Выражение может быть любым значащим выражением, но не может содержать memo-атрибуты.

Опция GET выводит и позволяет редактировать информацию из существующих переменных памяти или атрибутов. Команда @ . . . GET . . . сама по себе не позволяет редактировать значения переменных или атрибутов, она лишь определяет некоторое поле ввода. Команда READ, употребленная совместно с одной или несколькими заданными ранее командами @ . . . GET . . . , активизирует режим полноэкранного редактирования, и все ранее определенные поля ввода могут быть изменены (определены). GET опция может быть использована и с атрибутами типа memo. Последовательность @ . . . [SAY]. . .GET. . . команд создает список полей ввода, редактирование которых активизируется командой READ. Этот список может быть аннулирован командой CLEAR (см. 2.2.15).

Опция RANGE используется для задания верхних и нижних границ для числовой информации, а также информации типа дата. Выражения определяют минимальные и максимальные значения, которые могут быть введены в ответ на GET. Так как даты не имеют литерального представления, необходимо использовать здесь функцию CTOD ( ) при определении дат-констант. Например, RANGE CTOD ('01/01/85'), CTOD ('12/31/85') исключает из рассмотрения при вводе все даты с годом, отличным от 1985. Если требуется задать только одну границу, то вводится только одно выражение: RANGE 30, или RANGE, 10. Если в процессе ввода RANGE-условие не выполняется, запрашивается ввод до тех пор, пока заданные условия (⟨, ⟩) не будут выполнены. Если же @ команда содержит опцию RANGE, а в ответ на GET (поле ввода) следует нажатие Enter, то RANGE-проверка не производится, т. е. если атрибут или переменная установлены в значение, выходящее за пределы диапазона, то нажатие Enter оставляет значение неизменным.

Опция VALID для Clipper в добавление к опциям PICTURE и RANGE дает возможность проверить выполнение некоторого логического выражения при окончании ввода. Пользователь может завершить ввод в данное поле, только если выражение истинно либо нажав Esc в случае режима SET ESCAPE ON (для Clipper следует нажать Alt — C).

Для Clipper ошибочное сообщение не выдается, если координаты (⟨row, col⟩) указаны неверно.

Если необходимо использовать многостраничный формат (.fmt файл), в котором команды @ . . . SAY . . . GET продолжаются на 2—32 экранах, нужно включить команду READ в местах прерывания. Клавиши PgUp и PgDn будут листать страницы в режиме экранного редактирования, инициированном командой READ. Многостраничные форматные файлы могут работать, только если они активизированы командой SET FORMAT TO.

Опция PICTURE дает возможность определить формат (шаблон) вывода и ввода. Конструкция может состоять из функции и/или специальных шаблонов (описанных ниже) и должна быть в ограничителях. Если используется функция, то первым символом в конструкции должен быть @. Если функция используется совместно с шаблоном, то их должен разделять пробел.

Функции опции PICTURE

C	—выводит CR (кредит) после положительных чисел
X	— выводит DB (дебет) после отрицательных чисел
(	— заключает отрицательные числа в скобки
)	— то же, но с подавлением ведущих пробелов (только для Clipper)
B	— числовые данные выводятся с левой границы поля
Z	— вывод числового нуля в случае пустой строки
D	— американский формат представления дат мм/дд/гг
E	— европейский формат представления дат дд/мм/гг
A	— только буквы
!	— допускает любые символы, при этом буквы преобразуются в прописные
R	— литералы в шаблонах не являются частью вводимых данных
S ⟨n⟩	— ограничивает ширину поля просмотра до ⟨n⟩ символов и «прокручивает» горизонтально символьную строку внутри этих ⟨n⟩ столбцов, ⟨n⟩ должно быть целым числом, заданным литерально

Некоторые функции из приведенного списка применяются лишь к определенным типам данных. Функции C, X, (, B и Z могут быть применены только к числовым данным. Кроме того, C, X и (могут использоваться только в SAY конструкции. D и E применимы к дате, символьным и числовым данным. A, !, R и S уместны лишь над символьными данными.

Путем логического комбинирования перечисленных функций можно определить новые функции. Например, функция XC проставляет DB после отрицательных чисел и CR после положитель-

ных. Однако некоторые функции не могут использоваться совместно, например A!, DE и X(.

S ⟨n⟩ применяется для вывода и редактирования символьной GET переменной в меньшем, чем длина переменной, числе столбцов. Число столбцов задается литерально целым ⟨n⟩, при этом пробелы между «S» и этим числом недопустимы. Строка «прокручивается» в фиксированном окне, позволяя просматривать и редактировать существующую символьную информацию. Управляющие клавиши «влево», «вправо», Ctrl — F, Ctrl — A используются для скролинга строки в окне. При вводе данных скролинг осуществляется автоматически, поддерживая курсор вблизи середины области скролинга.

Шаблоны опции PICTURE

Шаблон формируется путем задания одиночных масок-шаблонов для каждого выводимого или вводимого символа. Хотя при определении шаблона могут использоваться любые символы, символы, отличные от масок-шаблонов, не имеют специального значения. Если используется функция R в сочетании с шаблоном, содержащим отличные от определенных масок символы, то эти символы будут включаться в вывод, однако не будут запоминаться как часть GET переменной. Если же функция R не используется, то они будут считаться также частью ввода. Это применимо только к символьным выражениям. В числовых подобные символы всегда включаются в вывод и никогда не считаются частью значения ввода. Следует избегать применения не являющихся масками символов дат и логических выражений.

Символы-маски

9 — допускает только цифру в этой позиции для символьных данных и цифру или знак для числовых данных
— допускает только цифру, пробел или знак
A — допускает только латинскую букву
L — допускает только логические данные
Y — допускает только логические Y, y, N, n; преобразует y и n в Y и N
N — допускает цифры и латинские буквы
X — допускает только символы
! — преобразует латинские буквы в прописные и не влияет на другие символы
$ — выводит знак $ вместо ведущих нулей
* — выводит * вместо ведущих нулей
. — специфицирует позицию десятичной точки
, — выводится, если есть цифры слева от запятой

Символы 9, #, A, N, L, Y, X и ! могут эффективно использоваться в конструкциях SAY. . . и GET. . . При использовании шаблона для ввода десятичного числа десятичная точка должна быть включена в шаблон.

PICTURE шаблон работает корректно в случае символьных переменных или атрибутов, если их длина составляет два или более байт.

См. также APPEND, CHANGE, COL ( ), CREATE/MODIFY SCREEN, EDIT, INSERT, MODIFY COMMAND, PCOL ( ), PROW (.), READ, ROW ( ), SET BELL, SET CONFIRM, SET DELIMITERS, SET DEVICE, SET FORMAT, SET INTENSITY, SET FIELDS.

2.2.3. @ ... BOX (только Clipper)

Используется для рисования прямоугольников на экране. Могут быть заданы размеры и относительная позиция прямоугольника на экране, и, кроме того, до 9 различных символов может использоваться при их выводе. Подобная команда для dBASE III Plus — @ . . . TO.

Синтаксис. @ ⟨t, l, b, r⟩ BOX ⟨string⟩

Использование. Здесь t и l — координаты верхнего левого, а b и r — нижнего правого угла прямоугольника. При этом t и b принимают значения от 0 до 24, а l и r — от 0 до 79.

Для ⟨string⟩ можно задать до 8 различных ASCII кодов для каждого из четырех углов и сторон (начиная с верхнего левого и по часовой стрелке). Если задан 9-й символ, то он используется в качестве заполнителя (фона) прямоугольника.

Задание пустой строки ⟨string⟩ — " " стирает прямоугольник.

2.2.4. @ ... PROMPT ... [MESSAGE ...] (только Clipper)

@ . . . PROMPT команда совместно с командами SET MESSAGE TO . . . и MENU TO . . . образует мощную систему конструирования меню при разработке приложений средствами Clipper.

Синтаксис. SET MESSAGE TO ⟨expN⟩ назначает строку для выдачи расширенных сообщений — пояснений текущей позиции

@ ⟨row, col⟩ PROMPT ⟨expC⟩ [MESSAGE ⟨expC⟩]
MENU TO ⟨var⟩ && аналогично READ

Использование. Команды @ . . . PROMPT используются для описания позиционных текстовых меню на экране. Каждая команда @ . . . PROMPT определяет одну позицию в меню. Параметр PROMPT команды описывает информацию, выводимую в качестве наименования данной позиции меню, а параметр MESSAGE содержит расширенное пояснение данной позиции (его может и не быть). Команда MENU TO ⟨var⟩ активизирует меню, описанное предварительно командами @ . . . PROMPT; она выделяет подсветкой первую позицию (prompt), позволяя затем передвигаться курсором от позиции к позиции меню и поместить в указанную переменную (var) порядковый номер выбранной позиции.

Допускается до 32 позиций в одном меню.

SET MESSAGE TO позволяет определить номер строки экрана, в которую выводятся сообщения, поясняющие ту позицию меню, на которой находится курсор, и определяемые в соответствующей данной позиции команды @ . . . PROMPT. Если задано SET MESSAGE TO 0 или просто SET MESSAGE TO, то сообщения помощи появляться не будут.

Нажатие клавиш Enter, PgUp, PgDn вызывает выбор текущей позиции меню и запоминание ее относительного номера в переменной ⟨var⟩. Нажатие Esc возвратит 0 в переменной ⟨var⟩ — выбор не был сделан. Выбор позиции может быть также произведен по первому символу (нажатием соответствующей алфавитно-цифровой клавиши), однако текущая версия Clipper распознает лишь коды 20h—7Fh (это может быть устранено путем соответствующей корректировки библиотеки Clipper .lib).

Меню могут быть вложенными без очистки списка полей ввода (здесь имеется в виду список позиций каждого меню) в отличие от GET/READ механизма. Однако если одна и та же переменная используется для вложенных меню, то она сохраняет предыдущее значение. Поэтому рекомендуется использовать отдельные переменные для каждого меню.

2.2.5. @ ... TO (dBASE III Plus и Clipper)

Эта команда рисует и стирает прямоугольник, очерченный одиночной или двойной линией. Добавлена в Clipper версии Autumn'86 для совместимости с dBASE III Plus.

Синтаксис. @ ⟨row1, col1⟩ [CLEAR] TO ⟨row2, col2⟩ [DOUBLE]

Использование. ⟨row1, col1⟩ и ⟨row2, col2⟩ — координаты соответственно верхнего левого и правого нижнего углов прямоугольника. Если они совпадают, то строится горизонтальная линия.

@ ⟨row, col⟩ очищает строку, заданную row и col.

Если не задано DOUBLE, то @. . .TO строит прямоугольник, ограниченный одиночной линией.

CLEAR очищает заданную прямоугольную зону экрана.

Команда @. . . TO может эффективно использоваться совместно с функцией @S ⟨n⟩ команды @ . . . SAY . . . GET, ограничивающей зону скролинга (см. 2.2.2.).

См. также @. . .SAY. . .GET.

2.2.6. ACCEPT (dBASE III Plus и Clipper)

Используется в программах для приглашения пользователю ввести некоторое значение с клавиатуры. Эта команда создает символьную переменную, в которую запоминает вводимую пользователем информацию. Окончание ввода выполняется по нажатию Enter.

Синтаксис. ACCEPT [⟨prompt⟩] TO ⟨memvar⟩

Использование. Сообщение (⟨prompt⟩) может быть символьной строкой (ограниченной ' ', " " или [ ]) или символьной переменной. Вводимая пользователем информация не требует ограничителей. При пустом вводе — немедленном нажатии на клавишу Enter — переменная получит «пустое» значение (т. е. без значения или ASCII 0). Максимальное число вводимых символов — 254.

См. также INPUT, WAIT.

2.2.7. APPEND (dBASE III Plus и Clipper)

Добавляет новый кортеж в конец активного отношения. В dBASE III PLus допускает эффективную работу в режиме экранного редактирования.

Синтаксис. Для dBASE III Plus:

APPEND [BLANK]

Для Clipper:

APPEND BLANK

Использование. Команда APPEND без опции BLANK для dBASE III Plus активизирует режим экранного редактирования для ввода данных, выводя на экран специальную форму (чистый бланк), содержащую имена всех атрибутов отношения.

Одна команда позволяет добавить сразу несколько кортежей в экранном режиме. Окончание процесса добавления происходит по нажатию Enter в ответ на значение первого из запрашиваемых атрибутов нового кортежа либо по нажатию Ctrl — End. Нажатие Ctrl — End в ответ на значение первого атрибута нового кортежа ведет к добавлению пустого кортежа к отношению.

Нажатие PgUp позволяет перейти к редактированию предыдущих кортежей. Порядок перехода всегда соответствует порядку хранения и не зависит от активного индекса (в случае, если он есть). Клавиша PgDn осуществляет последовательный возврат (кортеж за кортежем) из режима редактирования в режим добавления. Переход в режим добавления происходит при нажатии PgDn в процессе редактирования последнего существующего кортежа.

Для ввода значения атрибута типа memo следует нажать Ctrl — PgDn, когда курсор установлен на начало соответствующего поля. Управление передается экранному редактору, и характер дальнейшей работы зависит от выбранного в процессе инсталяции (запись в файле CONFIG.DB) текстового редактора для редактирования значений атрибутов типа memo. В случае использования внутреннего встроенного редактора dBASE III Plus экран очистится, позволяя вводить текст. Команды управления аналогичны командам режима MODIFY COMMAND. Вспомогательное меню управления курсором может быть высвечено нажатием на клавишу F1. Повторное нажатие стирает меню. Для выхода из редактирования необходимо нажать Ctrl — End для сохранения сделанных изменений и/или внесенного текста либо Esc для выхода без сохранения.

Один APPEND добавляет кортежи только в одно отношение. Использование форматных файлов дает возможность использовать @. . .GET для ввода значений в атрибуты кортежей различных логически связанных отношений. Использование APPEND с таким форматным файлом не позволяет добавлять кортежи к неактивизированным отношениям.

Опции. Опция BLANK позволяет добавить пустой кортеж к отношению без выхода в экранный режим. Предполагается, что содержимое будет затем введено с помощью других команд.

Для Clipper допустим только такой синтаксис (APPEND BLANK), поскольку компилятор не знает структуры реального отношения и, следовательно, не может сгенерировать «хороший» код для обеспечения экранного режима редактирования. Это относится и к другим командам — EDIT, BROWSE, CHANGE и т. п., которые не поддерживаются компилятором.

Добавленный по команде APPEND BLANK кортеж становится текущим.

Все индексные файлы, активные в момент интерпретации команды APPEND, модифицируются, отражая добавление новых кортежей к отношению.

См. также MODIFY COMMAND, SET CARRY, SET FORMAT TO.

2.2.8. APPEND FROM (dBASE III Plus и Clipper)

Копирует определенные кортежи некоторого отношения (или записи некоторого файла) в конец активного отношения.

Синтаксис. Для dBASE III Plus:

APPEND FROM ⟨filename⟩ [FOR ⟨condition⟩] [TYPE] [⟨file type⟩]

Для Clipper:

APPEND [scope] [⟨field list⟩] FROM ⟨filename⟩ [FOR ⟨condition⟩] [WHILE ⟨condition⟩] [SDF/Delimited]

Использование. Файл FROM может быть как отношением, так и некоторым «внешним» по отношению к dBASE файлом.

Если файл FROM — отношение, то: а) кортежи, отмеченные к удалению, все равно копируются и не помечаются к удалению в отношении-приемнике; б) копируются только атрибуты, найденные в обоих отношениях (Clipper, однако, позволяет дополнительно выбрать из них некоторое подмножество ⟨field list⟩), порядок их следования в отношениях не имеет значения; в) в этом случае не следует указывать тип файла FROM.

Если длина атрибута во FROM отношении превосходит соответствующую длину в отношении-получателе, то символьные данные усекаются, а числовые замещаются звездочками.

Для dBASE III Plus FOR выражение может содержать только атрибуты, общие для обоих отношений.

Опции. Для TYPE ⟨filetype⟩ можно задавать DELIMITED [WITH ⟨delimiter⟩/BLANK], что позволяет определить формат ограничителей при импорте информации из внешних ASCII файлов. Данные копируются символ за символом начиная слева. Каждая запись должна заканчиваться символами «возврат каретки» (код 0Dh) и «перевод строки» (0Ah). Если не задан специальный ограничитель (delimiter), то считается, что запятая разграничивает значения атрибутов, а двойные кавычки « » ограничивают символьные данные. Такое определение эквивалентно заданию опции DELIMITED WITH. DELIMITED WITH BLANK работает с файлами, содержащими значения атрибутов, разделенные одиночными пробелами.

DELIMITED WITH ⟨delimiter⟩ работает с файлами, содержащими значения атрибутов, разделенные запятой, а символьные строки могут (опционально) заключаться в специфицированный ограничитель (слева и справа).

Кроме этого могут быть указаны следующие типы файлов: SDF — (System Data Format for ASCII file). Данные копируются символ за символом начиная слева. Каждая запись имеет фиксированную длину и заканчивается кодами 0Dh, 0Ah. Этот формат может быть задан и для dBASE III Plus и для Clipper. Ниже приводится еще три формата, применение которых допускается в dBASE III Plus (согласно документации); DIF — VisiCalc file format. Строки таблицы VisiCalc преобразуются в кортежи отношения dBASE III Plus; SYLK — Multiplan spreadsheet format. Аналогично DIF, но dBASE III Plus не сможет выполнить APPEND FROM для таблицы Multiplan, сохраненной по столбцам (см. описание системы Multiplan); WKS — Lotus 1—2—3 spreadsheet format. Аналогично DIF.

При импорте информации из таблиц следует помнить, что если первая строка таблицы содержит символьную информацию о содержимом столбцов, то она должна быть предварительно удалена. Кроме того, таблицы обязательно должны быть сохранены предварительно по строкам.

Если команда APPEND FROM выполняется из файла, для которого dBASE III Plus подразумевает некоторое стандартное расширение имени, а файл не имеет никакого расширения, то нужно включить точку в имя файла в команде .

Не следует использовать опцию «DELIMITED WITH ,», если значения атрибутов содержат запятую внутри.

Специальные случаи. Для dBASE III Plus для импорта информации из PFS файлов следует использовать команду IMPORT (см. 2.2.60)). При импорте из отношений значение символьных атрибутов в формате даты может использоваться для импорта в атрибуты типа «дата» при совпадении имен атрибутов. Обратное также справедливо. Из текстовых файлов даты могут импортироваться только в форме ГГГГММДД без ограничителей.

См. также IMPORT.

2.2.9. ASSIST (только dBASE III Plus)

ASSISTANT представляет собой меню-управляемый монитор, дающий возможность пользователю организовать и вести свою информационную базу без программирования на языке dBASE. При выдаче команды ASSIST (которая может быть выдана автоматически при загрузке dBASE, будучи включенной в конфигурационный файл CONFIG.DB) управление передается монитору ASSISTANT, интерфейс которого представляет собой иерархию текстовых позиционных меню, позволяющих быстро и удобно специфицировать требуемые действия, например создание нового отношения, индекса, отчета, экранной формы, бланка, «запроса», «внешнего представления» (см. п. 1.5); активизацию любого из существующих объектов перечисленных выше типов; организацию добавления (модификации) — удаления информации в отношения БД в режиме экранного редактирования с возможным применением подготовленных ранее экранных форм; поиск информации по задаваемому в диалоге условию; выдачу требуемых отчетов; спецификацию параметров операционного окружения и др.

См. также HELP.

2.2.10. AVERAGE (dBASE III Plus и Clipper)

AVERAGE вычисляет среднее арифметическое для заданного выражения.

Синтаксис. Для dBASE III Plus:

AVERAGE [⟨expression list⟩] [⟨scope⟩] [WHILE ⟨condition⟩] [FOR ⟨condition⟩] [TO ⟨memvar list⟩]

Для Clipper:

AVERAGE ⟨field list⟩ TO ⟨memvar list⟩ [WHILE ⟨condition⟩] [FOR ⟨condition⟩]

Использование. Вычисляются средние арифметические для определенного списка атрибутов (dBASE III Plus позволяет умалчивать этот список — тогда обрабатываются все числовые атрибуты отношения). В процессе вычисления просматриваются все кортежи, удовлетворяющие условию выборки (FOR — WHILE опции).

Ситуация задания некоторого выражения над обрабатываемыми атрибутами, не определенная в синтаксисе команды, для Clipper не проверялась.

См. также COUNT, SUM.

2.2.11. BROWSE (только dBASE III Plus)

BROWSE —это команда вызова меню-управляемого монитора, обеспечивающего полноэкранный режим редактирования и добавление кортежей в отношение или во «внешнее представление» (см. п. 1.5) — файлы .dbf и .vue соответственно. Монитор позволяет одновременно обрабатывать до 17 кортежей, используя режим горизонтального скролинга в случае превышения ширины экрана суммарной длиной атрибутов. Это существенно отличает этот режим экранного редактирования от режимов, обеспечиваемых командами APPEND, EDIT.

Синтаксис. BROWSE [FIELDS ⟨fields list⟩] [LOCK ⟨expN⟩] [FREEZE ⟨field⟩] [NOFOLLOW] [NOMENU] [WIDTH ⟨expN⟩] [NOAPPEND]

Использование. Строка меню для режима BROWSE не присутствует постоянно на экране. Она вызывается нажатием функциональной клавиши F10. После выбора требуемой опции меню вновь исчезает. Меню ключей редактирования включается и выключается последовательными нажатиями клавиши F1.

Опции. FIELDS ⟨field list⟩. В списке можно указывать те атрибуты, которые подлежат просмотру/редактированию и в том порядке, в котором желательно их видеть.

LOCK ⟨expN⟩. Опция специфицирует число последовательно идущих слева на экране атрибутов, не подвергающихся горизонтальному скролингу по нажатиям Ctrl — «вправо» или Ctrl — «влево». Оставшаяся зона экрана подвержена скролингу и позволяет просматривать значения остальных атрибутов отношения, включенных в список FIELDS команды BROWSE (либо всех остающихся).

FREEZE ⟨field⟩. Если специфицирована опция FREEZE для некоторого атрибута, то только значения этого атрибута могут подвергаться редактированию в процессе BROWSE, хотя значения остальных атрибутов могут просматриваться.

NOFOLLOW. Влияет только на обработку индексированных отношений. Если не задана опция NOFOLLOW и изменено значение ключевого атрибута в кортеже, то указатель на кортеж изменяет свое значение (перемещается) в соответствии с новым содержимым ключа и кортеж по-прежнему остается текущим. Если же указано NOFOLLOW, то положение кортежа меняется (по индексу), но текущим становится кортеж, следовавший по индексному порядку за данным.

NOMENU. Использование этой опции запрещает активизацию строки-меню режима BROWSE в процессе редактирования.

WIDTH ⟨expN⟩. Ограничивает размер окна (количество символов) для любого атрибута режима BROWSE. Содержимое может просматриваться в соответствующем окне путем скролинга (клавиши направлений «вправо», «влево», Home, End) либо в процессе редактирования. Если размер атрибута меньше, то атрибуту выделяется меньшее окно согласно его размеру.

NOAPPEND. Запрещает добавление кортежей в процессе BROWSE.

Опции меню-строки режима BROWSE. Если не специфицировано NOMENU, то нажатие на клавишу F10 вызывает появление меню-строки, которое позволяет: разыскивать требуемый кортеж — опции Top (первый кортеж отношения), Bottom (последний), Record No. (поиск кортежа по номеру), Find (только для индексированных отношений — поиск по индексному ключу для активного индекса); определять опции FREEZE и LOCK в процессе редактирования.

См. также APPEND, EDIT.

2.2.12. CALL (dBASE III Plus и Clipper)

CALL позволяет программисту писать собственные процедуры на любом языке программирования, расширяющие возможности dBASE III Plus и Clipper.

Синтаксис. Для dBASE III Plus:

CALL ⟨module name⟩ [WITH ⟨expC⟩/⟨memvar⟩]

Для Clipper:

Call ⟨process⟩ [WITH ⟨parameter list⟩]

Использование. dBASE III Plus позволяет программисту написать некоторые процедуры на любом из имеющихся языков программирования (ЯП). Из каждой процедуры (модуля) следует сформировать .BIN файл (это можно сделать с помощью утилиты EXE2BIN), который перед вызовом должен быть загружен командой LOAD (см. 2.2.72). В команде CALL не следует указывать расширение имени файла при задании имени модуля (процедуры). Имя файла при загрузке автоматически становится именем соответствующего модуля. Регистр CS указывает при вызове на точку входа в модуль, а пара DS : BX содержит адрес первого байта передаваемой переменной памяти или символьного выражения. Передаваемая символьная строка заканчивается кодом NULL (ASCII 0). Тип передаваемой переменной памяти может быть произвольным. Примеры использования приведены в описании команды LOAD.

Clipper позволяет программисту написать набор программ (подпрограмм, процессов) на любых из имеющихся ЯП, компиляторы с которых имеют выход на стандартный объектный уровень Microsoft LINK. Созданные таким образом объектные модули линкуются с основными модулями разрабатываемого приложения, написанными на Clipper.

К подобной программе предъявляется всего лишь два требования: 1) модуль должен быть выполнен в стандартном объектном формате .OBJ («INTEL 8086 relocatable object file format»), доступном для редакторов связей MS LINK , PLINK86; 2) должны соблюдаться соглашения языка СИ о вызовах и передаче параметров.

В случае использования ЯП, отличного от Макроассемблера, в процессе «линкования» следует определить объектную библиотеку данного ЯП помимо объектной библиотеки CLIPPER.LIB (см. руководство по выбранному ЯП).

Список параметров в команде CALL может содержать до семи параметров. Разрабатываемые программы следует определять как «далекие» процессы, т. е. они должны заканчиваться длинным возвратом (FAR RET), что соответствует использованию «большой модели памяти» для большинства ЯП. Все параметры, передаваемые по ссылке, представляются четырехбайтовыми указателями (SEGMENT: OFFSET). Параметры укладываются в стек в порядке их передачи.

Правила передачи параметров нуждаются в уточнении, так как содержат порой противоречивые сведения. Однако можно отметить следующее.

1. Параметр передается по ссылке только в том случае, если указывается идентификатор, в противном случае (передается значе-

ние некоторого выражения — простейшее выражение имеет вид— (⟨идентификатор⟩) передается значение. Элементы массивов могут быть переданы только по значению. Возможно, что указание имени массива в качестве параметра ведет к передаче вызываемой программе ссылки на массив, однако и этот факт, и сама внутренняя структура массива (массив в Clipper — индексированная совокупность разнотипных в принципе элементов) нуждаются в уточнении.

2. Символьные строки оканчиваются кодом NULL (ASCII 0). Числовые переменные передаются в восьмибайтовом представлении с плавающей запятой (53 бита — основание, 11 бит — степень, смещенная на 1023), что соответствует формату DOUBLE языка СИ. В случае, если передается число, не превышающее 32K, можно использовать при передаче функцию Clipper WORD ( ) (см. 3.4.91), преобразующую числовое значение в двухбайтовое целое. что соответствует типу INT в языке СИ (CALL ⟨process⟩ WITH WORD (⟨name⟩)). Тип memo передается как длинная символьная строка.

3. При передаче параметров по ссылке изначальная длина параметра не должна увеличиваться, так как он может затереть информацию в зоне данных Clipper,— это в основном касается строковых значений типа character и memo.

4. Регистровые пары DX : BX и ES : BX указывают на первый параметр, что позволяет использовать поддержку, написанную ранее для dBASE III Plus путем простого добавления двух команд в начало каждого модуля

```
PUBLIC ⟨proc⟩
MOV DS, DX
```

актуализируя таким образом стандартный указатель dBASE III Plus -- DS : BX (регистр DS резервируется в Clipper из некоторых соображений совместимости).

5. Программа должна сохранять значения регистров BP, SS и DS.

Команда LOAD отсутствует в языке Clipper.

Объектная библиотека CLIPPER. LIB содержит в своем составе ряд функций, используемых при передаче параметров из Clipper в программы на языках СИ и Макроассемблер в случае применения этих языков для конструирования функций, «определенных пользователем» (UDF — User Defined Functions, см. команду FUNCTION). Эти функции позволяют определить количество переданных из Clipper параметров, проверить тип каждого из них, преобразовать данные из определенного формата Clipper в соответствующий формат языка, выполнить возврат параметра с обратным преобразованием.

См. также LOAD, RELEASE (для dBASE III Plus).

2.2.13. CANCEL (dBASE III Plus и Clipper)

CANCEL прекращает выполнение программы (закрывает все открытые командные файлы для dBASE III Plus, но не закрывает процедурные) и осуществляет выход в интерактивный режим для dBASE III Plus либо в DOS для Clipper.

См. также RETURN.

2.2.14. CHANGE (только dBASE III Plus)

CHANGE позволяет редактировать в полноэкранном режиме заданный набор атрибутов для определенных кортежей активного отношения для dBASE III Plus.

Синтаксис. CHANGE [⟨scope⟩] [FIELDS ⟨field list⟩] [WHILE ⟨condition⟩] [FOR ⟨condition⟩]

Использование. Для перемещения курсора по кортежу используются клавиши направлений. PgUp возвращает к предыдущему кортежу, а PgDn переводит к последующему. Ctrl — End заканчивает CHANGE и сохраняет все изменения. Esc заканчивает CHANGE без сохранения изменений в текущем кортеже (все остальные изменения сохраняются).

Для редактирования memo-атрибута следует поместить курсор на слово memo и нажать Ctrl — PgDn. Для окончания редактирования memo необходимо нажать Ctrl — End для сохранения сделанных изменений или Esc — для выхода без сохранения изменений (имеется в виду выход из стандартного редактора dBASE III Plus); если же для редактирования memo-атрибутов инстолирован другой редактор (запись в файле CONFIG.DB), то окончание редактирования производится средствами этого редактора. Управление редактированием при использовании встроенного редактора dBASE III Plus приведено в описании команды MODIFY COMMAND.

Режим CHANGE автоматически прерывается после обработки последнего кортежа, удовлетворяющего заданному в команде условию.

Команды CHANGE и EDIT в dBASE III Plus идентичны.

См. также EDIT, MODIFY COMMAND, SET FORMAT TO.

2.2.15. CLEAR (dBASE III Plus и Clipper)

CLEAR очищает экран и позиционирует курсор в левый верхний угол (координаты 0,0). Использование CLEAR в качестве опции команды @. . . позволяет очистить прямоугольную зону экрана.

Синтаксис. CLEAR

Использование. CLEAR также очищает список полей ввода (см. @. . .SAY. . .GET команду).

См. также @, CLEAR GETS.

2.2.16. CLEAR ALL (dBASE III Plus и Clipper)

Закрывает все открытые отношения, индексы, форматные файлы и связи, установленные командами SET. Уничтожает все переменные памяти и выбирает рабочую зону 1 (SELECT 1).

Синтаксис CLEAR ALL

Использование. Закрывает все .dbf файлы, связанные с ними .dbt и .ndx (.ntx для Clipper) файлы, форматные .fmt файлы и открытые каталоги (.cat файлы для dBASE III Plus).

См. также CLEAR MEMORY, CLOSE, RELEASE ALL

2.2.17. CLEAR FIELDS (dBASE III Plus)

Аннулирует список атрибутов, созданный командой SET FIELDS TO.

Синтаксис. CLEAR FIELDS

Использование. CLEAR FIELDS не производит никаких действий, если предварительно не был создан список атрибутов непосредственно командой SET FIELDS TO либо через «внешнее представление» (см. п. 1.5).

В отличие от команды SET FIELDS TO ⟨Enter⟩, которая удаляет только атрибуты активного отношения из списка атрибутов, CLEAR FIELDS аннулирует весь список атрибутов.

CLEAR FIELD автоматически выполняет SET FIELDS OFF.

См. также SET FIELDS ON, SET FIELDS TO, SET VIEW.

2.2.18. CLEAR GETS (dBASE III Plus и Clipper)

Очищает (аннулирует) все определенные ранее поля ввода (см. @. . .SAY. . .GET команду), возникшие со времени выдачи последней CLEAR ALL, CLEAR GETS или READ команды.

Синтаксис CLEAR GETS

Использование. dBASE III Plus позволяет задавать до 128 полей ввода, если в файле CONFIG.DB не специфицируется их количество (может быть задано от 35 до 1023). Предельный размер списка полей ввода для Clipper в документации не оговаривается.

CLEAR GETS не уничтожает переменных памяти.

2.2.19. CLEAR MEMORY (dBASE III Plus и Clipper)

Синтаксис. CLEAR MEMORY

Использование. Уничтожает текущие переменные памяти. Команда аналогична команде RELEASE ALL для интерактивного режима dBASE III Plus. В командном режиме различие этих команд заключается в следующем. CLEAR MEMORY уничтожает все переменные памяти: private и public (см. команду PUBLIC), в то время как RELEASE ALL уничтожает только private-переменные в текущей подпрограмме (или функции для Clipper).

2.2.20. CLEAR TYPEAHEAD (dBASE III Plus)

Синтаксис. CLEAR TYPEAHEAD

Использование. Команда CLEAR TYPEAHEAD используется для очистки буфера ввода (dBASE III Plus имеет специальный буфер ввода, куда последовательно укладываются все символы и управляющие коды, вводимые пользователем с клавиатуры; затем содержимое этого буфера используется при интерпретации команд, требующих ввода). В некоторых случаях требуется получить от пользователя информацию точно в определенной точке программы (например, при интерпретации команды WAIT) независимо от предыдущего ввода (содержимого буфера ввода — TYPEAHEAD). В этих случаях применяется команда CLEAR TYPEAHED.

См. также SET TYPEAHEAD TO.

2.2.21. CLOSE (dBASE III Plus и Clipper)

Команда CLOSE закрывает активные отношения, индексы, форматные файлы, процедурные файлы, файлы-протоколы.

Синтаксис. Для dBASE III Plus:


```
CLOSE ⟨file type⟩/ALL
Для Clipper:
CLOSE [⟨file type⟩]
где ⟨file type⟩ : := ALTERNATE/DATABASES/
                     FORMAT/INDEX/PROCEDURE
```

Использование. Если необходимо закрыть только активное отношение и связанные с ним файлы, следует использовать команду USE без параметров (см. 2.2.170) либо команду CLOSE для Clipper.

Для dBASE III Plus CLOSE DATABASES не влияет на рабочую область 10, если открыт каталог (см. п. 1.5).

Команда CLOSE ALL для dBASE III Plus закрывает все открытые файлы, не затрагивая переменных памяти. Clipper не разрешает использование параметра ALL.

2.2.22. CONTINUE (dBASE III Plus и Clipper)

Команда продолжает поиск, инициированный командой LOCATE. CONTINUE ищет следующий кортеж в активном отношении, который удовлетворяет условию, заданному в последней команде LOCATE.

Синтаксис. CONTINUE

Использование. Команда CONTINUE прекращает поиск, когда специфицированный кортеж найден либо когда достигнут конец диапазона поиска, определенного в LOCATE.

Команды LOCATE и CONTINUE относятся лишь к той рабочей области, в которой они были употреблены. Можно использовать различные LOCATE/CONTINUE для разных областей. При переключении на работу в другой рабочей области установки LOCATE/CONTINUE для покинутой области сохраняются, ожидая возврата.

Для dBASE III Plus успешный CONTINUE сопровождается выводом на экран номера найденного кортежа. Иначе dBASE III Plus выдает сообщение «End of locate scope» («Конец зоны поиска»). Если поиск был безуспешным, то указатель позиционируется на последний кортеж диапазона, заданного LOCATE, при этом если это последний кортеж отношения, то функция EOF ( ) получает значение «Истина» (.T.).

См. также LOCATE, FOUND ( ).

2.2.23. COPY (dBASE III Plus и Clipper)

COPY копирует содержимое активного отношения (или его части) в новое отношение (или внешний текстовый файл.). COPY — базовая команда для вывода информации во внешние текстовые файлы.

Синтаксис. Для dBASE III Plus:

```
COPY TO ⟨new filename⟩ [⟨scope⟩]
        [FIELDS ⟨field list⟩]
        [WHILE ⟨condition⟩] [FOR ⟨condition⟩]
        [TYPE] [⟨file type⟩]
```

Для Clipper:

COPY TO ⟨new filename⟩ [⟨scope⟩]
[FIELDS ⟨field list⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]
[SDF/DELIMITED
[WITH BLANK/⟨delimiter⟩]]

Использование. Все кортежи, включая кортежи, помеченные к удалению, копируются, если обратное не задано опциями ⟨scope⟩, WHILE. . . , FOR. . . данной команды или переключателем SET DELETED ON.

Все атрибуты копируются, если список требуемых не задан опцией FIELDS ⟨field list⟩.

Если тип файла-получателя не указан, то предполагается копирование в отношение (.dbf). При задании SDF/DELIMITED опции memo-атрибуты не копируются. Если требуется копировать в отношение, хранимое в .dbf файле, следует задать только имя этого отношения, но не указывать расширение. Если же необходимо копировать во внешний файл, имя которого не содержит расширения, то нужно включить «.» как последний, дополнительный символ имени файла.

Дата и время для метки в оглавлении, соответствующей создаваемому файлу-копии, проставятся те же, что были в метке исходного файла (отношения).

Опции для TYPE ⟨file type⟩ в dBASE III Plus:

DELIMITED [WITH ⟨delimiter⟩/BLANK] — формат с разделителями. Файл-копия — ASCII файл. Данные копируются символ за символом начиная слева. Каждая запись кончается кодами ⟨возврат каретки⟩, ⟨перевод строки⟩ (⟨CR⟩ ⟨LF⟩). Если не указан символ-разграничитель, то значения атрибутов разделяются запятой и символьные значения берутся в двойные кавычки " ". DELIMITED WITH BLANK разделяет значения атрибутов одним пробелом, а DELIMITED WITH ⟨delimiter⟩ позволяет разделять значения выбранным символом. Расширение файла-копии по умолчанию — .txt;

SDF — System Data Format. ASCII файл. Данные копируются символ за символом начиная слева. Каждая запись фиксированной длины заканчивается ⟨CR⟩⟨LF⟩. Расширение файла-копии по умолчанию — .txt;

DIF — VisiCalc формат (только для dBASE III Plus). Кортежи преобразуются в строки VisiCalc, а атрибуты — в столбцы;

SYLK — Multiplan spreadsheet formula (только для dBASE III Plus). Аналогично DIF, но для Multiplan;

WKS — Lotus 1—2—3 spreadsheet формат (только для dBASE III Plus). Аналогично DIF и SYLK, но для Lotus 1—2—3.

Если COPY используется для записи в трех описанных форматах электронных таблиц, имена атрибутов записываются как заголовки столбцов в файле-копии. Файл создается в порядке «по строкам».

Специальные случаи. Для преобразования dBASE III Plus файлов в файлы PFS : FILE следует использовать команду EXPORT.

Нельзя использовать буквы A—J как имена результирующих отношений, так как они резервируются для имен рабочих областей (ALIAS names).

См. также APPEND FROM, COPY FILE, COPY STRUCTURE, EXPORT, SET DELETED, SET SAFETY.

2.2.24. COPY FILE (dBASE III Plus и Clipper)

Эта команда создает копию файла любого типа.

Синтаксис. COPY FILE ⟨file1⟩ TO ⟨file2⟩

Использование. Оба имени должны включать расширения и код диска (если в процессе копирования один из файлов должен быть на диске, отличном от «default drive» — диска умолчания). Если копируется отношение, содержащее memo-атрибуты, то соответствующие .dbt файлы должны копироваться отдельно.

Команда COPY FILE неприменима к открытому файлу.

Нельзя использовать буквы A—J как имена результирующих отношений, так как они резервируются для имен рабочих областей (ALIAS names).

См. также CLOSE, COPY, USE.

2.2.25. COPY STRUCTURE (dBASE III Plus и Clipper)

Копирует только структуру исходного отношения в результирующее, создавая таким образом «пустое» новое отношение.

Синтаксис. COPY STRUCTURE TO ⟨filename⟩
[FIELDS ⟨field list⟩]

Использование. Имя файла-отношения должно включать код дисковода, если он не на диске умолчания — «default drive». По умолчанию TO файл получает расширение .dbf. Копируется существующая структура либо (если задана FIELDS опция) только подструктура, содержащая выбранные атрибуты.

См. также SET SAFETY, DISPLAY STRUCTURE.

2.2.26. COPY STRUCTURE EXTENDED (dBASE III Plus и Clipper)

По этой команде создается новое отношение с четырьмя атрибутами: имя атрибута, тип атрибута, длина атрибута, число десятичных позиций. Содержимое вновь созданного отношения представляет собой описание структуры отношения-источника.

Синтаксис. COPY TO ⟨new file⟩ STRUCTURE EXTENDED

Использование. Эта команда часто используется в прикладных программах для порождения новых и модификации существующих отношений, структура которых определяется в ходе выполнения программы, без обращения к стандартным интерактивным процедурам CREATE и MODIFY STRUCTURE dBASE III Plus. Для Clipper это единственный путь создания и модификации отношений средствами прикладной программы, так как интерактивные команды не поддерживаются в Clipper (в связи с невозможностью их эффективной компиляции). Для Clipper включена специальная команда CREATE ⟨filename⟩, которая создает специальный шаблон-заготовку — новое отношение с четырьмя атрибутами: имя атрибута, тип атрибута, длина атрибута, число десятичных позиций. Содержимое вновь созданного отношения вначале пусто, а затем заполняется в ходе исполнения программы, чтобы после окончательного заполнения создать результирующее отношение командой CREATE FROM.

Следует использовать команду CREATE FROM для создания нового отношения по содержимому STRUCTURE EXTENDED отношения.

Структура отношения STRUCTURE EXTENDED следующая:

Атрибут	Имя	Тип	Длина	Длина после десятичной точки
1	FIELD—NAME	C	10	0
2	FIELD—TYPE	C	1	0
3	FIELD—LEN	N	3	0
4	FIELD—DEC	N	3	0

См. также CREATE FROM.

2.2.27. COUNT (dBASE III Plus и Clipper)

Команда подсчитывает число кортежей в активном отношении, удовлетворяющих заданному условию.

Синтаксис. Для dBASE III Plus:

```
COUNT [⟨scope⟩] [WHILE ⟨condition⟩]
      [FOR ⟨condition⟩] [TO ⟨memvar⟩]
```

Для Clipper:

```
COUNT [⟨scope⟩] [WHILE ⟨condition⟩]
      [FOR ⟨condition⟩] TO ⟨memvar⟩
```

Использование. Для Clipper необходимо определить переменную памяти, в которую запишется результат выполнения команды.

Если никаких условий не задано, то подсчитываются все кортежи в отношении.

См. также AVERAGE, SUM, TOTAL.

2.2.28. CREATE (dBASE III Plus и Clipper)

Эта команда имеет различный смысл для dBASE III Plus и для Clipper. В Clipper команда CREATE ⟨filename⟩ создает пустое отношение типа STRUCTURE EXTENDED (см. COPY STRUCTURE EXTENDED), предназначенное для формирования требуемой структуры отношения, которое затем будет создано по команде CREATE FROM.

В dBASE III Plus это интерактивная команда, позволяющая непосредственно в диалоге создавать требуемое отношение и, если необходимо, начать вводить в него информацию.

Clipper содержит специальную утилиту для создания отношений CREATE.

Синтаксис. Для dBASE III Plus:

```
CREATE [⟨filename⟩]
```

Для Clipper:

```
CREATE ⟨filename⟩
```

Использование. В имени файла подразумевается расширение .dbf, если другое не задано.

Далее рассмотрим интерпретацию команды CREATE для dBASE III Plus. Команда организует интерактивный режим для задания структуры создаваемого отношения. По каждому атрибуту требуется задать имя, тип, длину, а в случае числового атрибута может также быть определено количество позиций после десятичной точки.

Тип атрибута вводится путем нажатия первой буквы имени соответствующего типа (на любом регистре):

C — символьный (character);
N — числовой (numeric);
L — логический (logical);
D — дата (date);
M — текст (memo).

Кроме того, можно нажимать «пробел», пока в графе «тип» для данного атрибута не появится требуемый тип,— тогда следует нажать Enter для его фиксации. Если выбран тип memo, то автоматически создастся соответствующий .dbt файл.

Десятичная точка (если используется для числовых атрибутов) занимает одно знакоместо в общей длине и учитывается как позиция при определении числа позиций после точки. Текущий определяемый атрибут выделен на экране подсветкой (или другим цветом). Подробные инструкции и сообщения об ошибках появляются в нижней части экрана.

Клавиши направлений позволяют спозиционировать курсор к описанию любого атрибута. Когда отношение содержит большое количество атрибутов и их описания не помещаются на одном экране, нажатие PgUp и PgDn ведет к скроллингу экрана. Отношение может содержать до 128 атрибутов (в Clipper — до 1024 !).

Подобным образом описываются один за другим атрибуты создаваемого отношения; при этом есть возможность вернуться — скорректировать описание, вставить в определенном месте новый атрибут, удалить некоторый атрибут. Вставка атрибута производится путем позиционирования курсора на требуемую позицию и нажатия затем Ctrl—N. Удаление атрибута производится путем позиционирования курсора на описание удаляемого атрибута и нажатия затем Ctrl—U.

Для выхода из режима CREATE и сохранения структуры следует нажать Ctrl — End либо Enter, когда запрашивается определение нового атрибута. Выход по Esc аннулирует созданную структуру.

Не следует использовать буквы A—J в качестве имен создаваемых отношений, так как они резервируются для имен рабочих областей (ALIAS names).

2.2.29. CREATE FROM (dBASE III Plus и Clipper)

CREATE FROM создает отношение, структура которого определяется содержимым FROM отношения, созданного командой COPY STRUCTURE EXTENDED (либо CREATE ⟨filename⟩ для Clipper).

Синтаксис. CREATE ⟨new file⟩
FROM ⟨structure extended file⟩

Использование. Отношение, созданное этой командой, становится активным.

2.2.30. CREATE/MODIFY LABEL (только dBASE III Plus)

Команда инициирует интерактивный меню-управляемый режим экранного редактирования для создания и редактирования этикеток (меток, бланков) — .lbl файлов (см. п. 1.5). Команда позволяет соз-

давать стандартные этикетки по отношениям (.dbf), запросам (фильтрам) (.qry) и внешним представлениям (.vue). Печать заполненных этикеток производится командой LABEL.

Данная команда определена в dBASE III Plus. Clipper имеет специальную утилиту — LABEL .EXE, предназначенную для создания и редактирования этикеток.

Синтаксис. CREATE/MODIFY LABEL ⟨filename⟩/?

Использование. Если расширение не задано, dBASE III Plus подразумевает .lbl. Если открыт каталог и SET CATALOG находится в состоянии ON (значение по умолчанию), то .lbl файл добавляется в каталог.

CREATE LABEL ⟨.lbl filename⟩ создает новую, а MODIFY LABEL ⟨.lbl filename⟩/? модифицирует существующую этикетку.

Для изменения существующей этикетки из открытого каталога, имя которой забыто, используется MODIFY LABEL ?. Эта форма команды вызывает выдачу списка .lbl файлов, содержащих этикетки, связанные с активным отношением.

После того как структура этикетки создана, этикетку можно распечатать на экране или на принтере, используя команду LABEL.

Нажатие ключа F1 включает/выключает навигационное меню (меню управления курсором).

Строка-меню CREATE/MODIFY LABEL содержит три позиции (подменю): опции, содержимое, выход.

Подменю «опции» позволяет специфицировать ряд размеров для этикетки. dBASE III Plus предлагает выбрать один из пяти стандартных видов этикеток:

Ширина	Длина	Число этикеток на листе (по ширине)
3—1/2"	15/16"	1
3—1/2"	15/16"	2
3—1/2"	15/16"	3
4"	1—7/16"	1
3—2/10"	11/12"	3 (чеширский стиль)

После выбора вида этикетки dBASE III Plus автоматически заполняет все оставшиеся размеры. Если необходимо, можно уточнить эти параметры.

Параметр	Допустимый диапазон значений
Ширина этикетки	1—120 символов
Длина этикетки	1—16 строк
Левая граница (отступ от края листа)	0—250 символов
Число строк между этикетками	0—16 строк
Число пробелов между этикетками	0—120 символов
Число этикеток на листе (по ширине)	1—15 бланков

Максимальная суммарная ширина всех этикеток на листе — 250 символов.

Подменю «содержимое» определяет тип информации для каждой строки этикетки. Можно ввести столько строк, сколько определено при создании этикетки в подменю «опции».

Если строка должна содержать имена атрибутов, то для активизации списка имен атрибутов, из которого можно выбрать требуемое имя, необходимо нажать клавишу F10. Возврат в основное положение осуществляется нажатием стрелок «вправо» или «влево». Если

необходимо разместить более одного атрибута в строке, то dBASE III Plus разделит атрибуты запятыми. При печати значения атрибутов разделяются пробелами. Можно вводить выражения. Например, для того чтобы создать строку вида

Киев, Печерский р-н 252015

необходимо ввести

Label contents 1> City,',', Region, Zip

Подменю «выход» содержит опции 'Save' (сохранение) и 'Abandon' (сброс), позволяющие закончить работу над бланком либо сохранив его (или сделанные изменения), либо нет.

См. также LABEL.

2.2.31. CREATE/MODIFY QUERY (только dBASE III Plus)

Эта команда (либо CREATE, либо MODIFY) инициирует полноэкранный меню-управляемый режим, предназначенный для построения или модификации запроса (фильтра), связанного с некоторым отношением или внешним представлением (см. п. 1.5). Активизация фильтра (см. SET FILTER TO FILE . . .) позволяет выдавать командам, осуществляющим обработку активного отношения, только те кортежи, которые удовлетворяют условию фильтрации.

Синтаксис. CREATE/MODIFY QUERY ⟨.qry filename⟩/?

Если расширение не указано, dBASE III Plus подразумевает .qry. Если открыт каталог и SET CATALOG в положении ON (значение по умолчанию), то .qry файл добавляется к открытому каталогу.

Использование. Команда CREATE QUERY ⟨.qry filename⟩ используется для построения нового фильтра, а MODIFY QUERY ⟨.qry filename⟩/? для модификации существующего.

Если необходимо модифицировать существующий фильтр из открытого каталога, можно воспользоваться параметром ? команды, что приведет к выдаче списка всех фильтров, связанных с выбранным каталогом или внешним представлением.

Для активизации существующего фильтра без входа в Query-меню следует использовать SET FILTER TO FILE ⟨.qry filename⟩/? При выходе из Query меню по опции Save (сохранение) dBASE III Plus автоматически активизирует созданный / модифицированный фильтр. Для его деактивации можно использовать команду SET FILTER TO.

Опции. Query меню является меню-строкой и содержит четыре позиции. Первая позиция — Set Filter. При создании нового запроса необходимо начать с этой позиции. Ее выбор активизирует таблицу, в которой показываются условия фильтрации по мере их создания/модификации. Помимо таблицы активизируется столбчатое меню, каждая позиция которого соответствует определенному этапу в построении очередного условия фильтрации. Запрос может содержать до семи (включительно) условий фильтрации. Каждое условие представляет собой конструкцию вида ⟨имя атрибута⟩ ⟨оператор⟩ ⟨константа/выражение⟩. Условия связываются между собой логическими связками. Описанной конструкции соответствует таблица построения запроса, содержащая семь строк (по максимальному числу условий в запросе) и четыре столбца: «атрибут», «оператор» «константа/выражение», «логическая связка».

Приведем позиции столбчатого меню (Set Filter).

Field Name (имя атрибута) вызывает позиционное подменю, содержащее все имена атрибутов активного отношения и позволяющее выбрать нужный атрибут.

Operator (оператор) вызывает позиционное подменю, содержащее список всех возможных операторов (набор операторов зависит от типа атрибута) и позволяющее выбрать необходимый.

Constant/Expression (константа/выражение) — единственная позиция, где можно ввести собственные данные, а не выбирать их в позиционном меню (см. ниже).

Connect (логическая связка) связывает два условия вместе в логическое выражение — позиционное подменю, позволяющее выбрать требуемую связку.

Line Number (номер строки) дает возможность перейти к редактированию (созданию) требуемого условия в запросе. Каждому условию соответствует строка в таблице. По умолчанию номер строки — следующая строка.

Для вставки нового условия в текущей строке необходимо нажать Ctrl—N (все строки подвинутся начиная с текущей).

Для удаления условия в текущей строке следует нажать Ctrl-U (текущая строка удалится, все строки ниже подвинутся на одну вверх).

Следующей позицией Query меню является Nest (вхождение). Эта позиция позволяет заключать условия в создаваемом / модифицируемом запросе в скобки, тем самым изменяя порядок, в котором они вычисляются. Nest меню позволяет добавлять и убирать скобки. Опция Start добавляет или убирает левую (открывающую) скобку, а опция End — правую.

Позиция Display (просмотр) позволяет просмотреть результаты созданного/модифицированного запроса, чтобы убедиться в его правильности. Выводится один кортеж за один раз в формате типа EDIT (см. команду EDIT). Нажатие PgUp и PgDn позволяет просматривать другие кортежи.

Позиция Exit (выход) позволяет сохранить созданный запрос (опция Save либо нажатие Ctrl—End в любой момент) или уничтожить результаты работы (опция Abandon либо нажатие Esc в любой момент).

Рассмотрим процесс конструирования констант-выражений в зависимости от типа атрибута, по которому налагается условие.

1. Символьные атрибуты. Здесь должно вводиться только символьное выражение. Если это константа, то требуется выполнить следующие действия: нажать Enter; ввести начальный ограничитель ' (либо", либо [); ввести требуемую символьную константу; ввести закрывающий ограничитель; вновь нажать Enter — константа зафиксируется в таблице условий — или продолжать ввод выражения.

Если это имя некоторого атрибута, то следует: нажать Enter; нажать F10 для вызова списка имен атрибутов; выбрать требуемый атрибут; нажать Enter для его фиксации или продолжать вводить выражение.

Если это выражение, то следует ввести требуемые инструкции, включая допустимые в данном контексте функции, строчные константы, имена атрибутов, соединяя их допустимыми операциями.

2. Числовые атрибуты. Выражения вводятся аналогично предыдущему случаю.

3. Даты. Здесь в большинстве случаев требуется представить дату символьной строкой. Для этого можно использовать функцию CTOD (преобразование символьный тип — дата), для ввода которой

следует: нажать Enter; набрать CTOD(' ; ввести символьную строку в формате даты, в которую необходимо преобразовать введенную строку; набрать '). Остальное аналогично предыдущим случаям.

См. также SET CATALOG, SET FILTER.

2.2.32. CREATE/MODIFY REPORT (только dBASE III Plus)

Команда инициирует полноэкранный меню-управляемый режим, предназначенный для генерации или редактирования отчета (отчетной формы — .frm файла). Она позволяет сконструировать отчетную форму в формате «по столбцам» по некоторому отношению либо внешнему представлению (см. п. 1.5). В Clipper с той же целью существует специальная утилита REPORT, однако более удобно требуемые отчетные формы создаются в среде dBASE III Plus с последующим их использованием приложениями на Clipper.

Синтаксис. CREATE/MODIFY REPORT (.frm filename)/?

Использование. Если иное не специфицировано, dBASE III Plus подразумевает расширение .frm. Если открыт каталог и SET CATALOG находится в значении ON (значение по умолчанию), то создаваемый отчет (.frm файл) будет добавлен в каталог.

CREATE форма команды позволяет создать новый, а MODIFY — модифицировать существующий отчет. Параметр ? допустим для MODIFY формы команды в случае открытого каталога и применяется в том случае, если забыто имя отчета, который надо редактировать.

После того как отчет создан/отредактирован (имеется в виду его структура), можно распечатать собственно отчет командой REPORT на экран или принтер.

Клавиша F1 в процессе работы над структурой отчета позволяет включать и выключать меню навигации (управления курсором).

Опции. Строчное меню команды CREATE/MODIFY REPORT содержит пять позиций.

Первая — Options — позволяет задать геометрическую конфигурацию отчета и содержит девять подпунктов.

1. Title (заголовок) дает возможность создать заголовок, который начинает каждую страницу отчета. Заголовок может состоять не более чем из четырех строк, каждая из которых не превышает 60 символов. Нажатие Enter на этом подпункте приводит ко входу в режим редактирования заголовка. Для завершения следует нажать Ctrl—End.

2. Page Width (ширина листа). Значение по умолчанию — 80 символов. Его можно изменить. Допустимы значения от 1 до 150.

3. Left Margin (левая граница) — отступ от левого края листа при печати.

4. Right Margin (правая граница) — отступ от правого края листа при печати.

dBASE III Plus автоматически проверяет условие ширина листа — (левая граница + правая граница) > 0.

5. Number of lines per page (число строк на странице). Можно установить значение от 1 до 500.

6. Double space report (двойной интервал в отчете). Значение по умолчанию — НЕТ.

7. Page eject before printing (перевод листа перед печатью). Значение по умолчанию — ДА. При изменении его на НЕТ отчет начнет

печататься без прогона бумаги непосредственно с текущего места. Это можно сделать и непосредственно в команде REPORT, включив в нее опцию NOEJECT, которая отменяет установку, сделанную при создании — модификации отчета.

8. Page eject after printing (перевод листа после печати). Значение по умолчанию — НЕТ. Можно вызвать перевод листа после печати отчета, изменив значение. Это эквивалентно выдаче команды EJECT после печати отчета.

9. Plain page (планировка отчета) определяет, выводить ли системную дату, номер страницы и заголовок (заданный Heading опцией команды REPORT) на каждой странице отчета. Значение по умолчанию— OFF (выключение планировки), эта информация выводится на каждой странице отчета. Если включить планировку (Plain page ON), то заголовок появится только в начале отчета, а дата и номера страниц вообще появляться не будут. Эту опцию можно активизировать и непосредственно в команде REPORT (включив в ее синтаксис опцию PLAIN) — даты, номера страниц и заголовки в отчете появляться не будут.

Вторая позиция главного меню команды CREATE/MODIFY REPORT — Group (группирующее) меню. Это опциональная возможность, не требующаяся для многих отчетов. Позиции этого меню позволяют упорядочить отчет, сгруппировав выводимые записи по определенным признакам. Рассмотрим позиции Group меню.

1. Group on expression (группирование по выражению) дает возможность организовать отчет таким образом, чтобы записи с одинаковым значением атрибута или ключевого выражения, определенного в Columns-меню, собирались в отчете вместе. Это особенно наглядно при подытоживании значений числовых атрибутов — каждая группа содержит итог (сумму).

При создании групп dBASE III Plus начинает новую группу, как только обнаруживает новое значение. Поэтому предварительно следует обязательно проиндексировать либо отсортировать подотчетное отношение по ключу группирования.

2. Group heading (заголовок группы). Каждый раз, выдавая новую группу, dBASE III Plus озаглавливает ее этим текстом.

3. Summary report only (только итоговый отчет). Содержимое групп не выдается, а в отчете приводятся лишь итоговые значения по каждой группе. Значение по умолчанию — НЕТ (содержимое выдается).

4. Page eject after group (перевод листа после каждой группы). Значение по умолчанию — НЕТ.

5. Subgroup on expression (выделение подгрупп по выражению) позволяет ввести второй (вложенный) уровень группирования, разбивая группы на подгруппы. Отношение должно быть проиндексировано или отсортировано по вторичному ключу (ключу подгрупп) аналогично тому, как было сказано для первичного ключа (см. поз. 1).

6. Subgroup heading (заголовок подгруппы). Перед каждой подгруппой в отчете может выдаваться заголовок.

Третья позиция меню — Columns меню (столбцы). Эта позиция позволяет описать собственно содержимое отчета. Отчет может содержать до 24 столбцов.

Рассмотрим позиции Columns меню.

1. Contents (содержимое). Содержимым столбца могут быть значения некоторого атрибута либо выражения, которое может включать и переменные памяти. Процесс внесения содержимого аналогичен процессу конструирования констант-выражений при

задании условий фильтрации (см. команду CREATE/MODIFY QUERY).

2. Heading (заголовок). Идентифицирует содержимое столбца в отчете. Можно ввести до четырех строк длиной не более 59 символов каждая. Следует нажать Enter, ввести заголовок (пользуясь при необходимости ключами редактирования), вновь нажать Enter, затем перейти к другим меню.

3. Width (ширина). Специфицирует ширину данного столбца. По умолчанию автоматически принимает значение, равное размеру атрибута-выражения, специфицированного для данного столбца либо заголовка (что длиннее). Может быть изменена. В случае ее уменьшения заголовок и/или значения усекаются в отчете до заданного размера.

4. Decimal places (десятичные позиции). Используется для числовых величин. По умолчанию равно числу позиций после точки, зафиксированному в структуре отношения. При уменьшении этого числа значения округляются.

5. Total this column (итог по этому столбцу). Можно получить сумму (итог) по столбцу с числовым содержимым. Значение по умолчанию — ДА. Это ведет к выдаче подсумм по каждой группе и подгруппе (если они заданы) и общей суммы по данному столбцу в конце всего отчета.

Для удаления существующего столбца следует нажать Ctrl—U, для вставки нового столбца — Ctrl — N.

Четвертой позицией главного меню является Locate (позиционирование). При выборе этой позиции появляется список атрибутов и выражений, определенных в процессе работы над содержимым отчета в Column меню. Выбор любого из них приводит к переходу в Column меню в режим редактирования этого столбца. Таким образом, данная позиция является вспомогательной и употребляется в процессе работы с Column меню.

Пятая позиция — Exit (выход). Дает возможность сохранить результаты работы (Save-опция, что аналогично нажатию Ctrl — End) либо аннулировать их (опция Abandon либо нажатие Esc в любой момент).

Вид отчета

В процессе создания — редактирования отчета (путем ввода информации в Column меню) кодовая диаграмма, дающая представление о виде создаваемого отчета, выводится в нижней половине экрана. В ней приняты следующие коды (условные обозначения).

Код	Значение
>	— правая граница отчета
<	— левая граница отчета
9	— числовой столбец без подытоживания
#	— числовой столбец с подытоживанием
mm/dd/yy	— столбец дат. Если выбранный формат даты отличается от mm/dd/yy (например, dd/mm/yy — европейский формат), то выводится выбранный формат. Если сделана установка SET CENTURY ON (см.), то год выводится как yyyy
.L.	— столбец логических значений
?	— столбец, содержащий значения типа текст (memo).

См. также REPORT.

2.2.33. CREATE/MODIFY SCREEN (только dBASE III Plus)

Эта команда (CREATE или MODIFY форма) инициирует полноэкранный меню-управляемый режим, предназначенный для создания требуемых экранных форм, широко используемых в различных приложениях (см. .scr и .fmt файлы в п. 1.5) для организации ввода — модификации — просмотра информации. В процессе создания — модификации экранных форм могут быть созданы новые отношения либо модифицирована структура существующих.

Clipper не имеет в стандартном составе утилит средств создания .fmt файлов, однако их можно создавать в среде dBASE III Plus, используя затем в приложениях, написанных на Clipper.

Синтаксис. CREATE/MODIFY SCREEN (.scr filename)/?

Использование. Если другое не специфицировано, подразумевается расширение .scr в спецификации файла. Генерируемый форматный файл по умолчанию получает расширение .fmt.

Если открыт каталог и SET CATALOG находится в значении ON (значение по умолчанию), то создаваемые экранная форма (.scr файл) и соответствующий форматный файл (.frm файл) будут добавлены в каталог.

Следует использовать CREATE форму команды для построения новой экранной формы, а MODIFY — для редактирования уже существующей. Если забыто имя формы, которую нужно редактировать, но она содержится в открытом каталоге, то можно использовать команду MODIFY SCREEN ?, которая позволит выбрать нужную форму из списка существующих.

При выходе из редактора экранных форм (Screen меню) по опции Save (сохранение) dBASE III Plus автоматически генерирует форматный файл (.fmt) с тем же именем, что и экранная форма (.scr файл). Затем dBASE III Plus активизирует новый / отредактированный форматный файл (используя команду SET FORMAT TO (.fmt filename)). Для его деактивации (если это нужно) следует дать команду SET FORMAT TO. В дальнейшем при необходимости можно переименовать форматный файл.

Предупреждение. Если .scr файл удален, то больше нельзя использовать команду MODIFY SCREEN для редактирования соответствующего .fmt файла. Невозможно восстановить .scr файл по.fmt файлу, т. е. можно в дальнейшем редактировать .fmt файл только средствами текстового редактора (MODIFY COMMAND или др.).

Справедливо и обратное: если внесены изменения прямо в .fmt файл (с помощью текстового редактора), то в дальнейшем его нельзя редактировать с помощью MODIFY SCREEN, так как содержимое .scr файла уже не соответствует содержимому .fmt файла. Продолжать редактировать его можно с помощью текстового редактора или следует вернуться к старому варианту (представленному в .scr файле).

Ниже перечислены основные возможности редактора экранных форм: автоматическое создание команд @. . .SAY и @. . .GET по тем позициям, где размещены текст и окна для ввода — вывода значений атрибутов; создание отношений и модификация структуры существующих отношений непосредственно в процессе построения экранных форм; изменение размеров (длины) окон (значений атрибутов) в экранной форме без изменения структуры отношения; определение PICTURE-опции для вводимых/выводимых значений (для соответствующих окон) по списку допустимых функций и/или шаблонов (см. команду @. . .SAY. . .GET); задание диапазонов вводимых значений (RANGE-опция — см. команду @. . .SAY. . .GET. . .); рисование вертикальных и горизонтальных линий и прямоугольников (расчерчивание формы); автоматическая генерация форматного файла; генерация текстового файла ASCII, документирующего данную экранную форму.

Определим основные понятия редактора экранных форм.

1. Электронная доска. Область, в которой создается / редактируется экранная форма, называется электронной доской. Для переключения между электронной доской и Screen-меню используется клавиша F10. Далее в описании позиций меню будет объяснено, как вносить / редактировать текст и атрибутные окна на электронной доске. Доска занимает 1—20-ю строки экрана.

2. Редактирование текста. Используются стандартные клавиши режима экранного редактирования для изменения текста на электронной доске и для позиционирования курсора. Однако есть ряд специальных действий, связанных с определением атрибутных окон в конструируемой экранной форме. Рассмотрим специальные действия.

Для перемещения атрибутного окна или текста следует: перейти в режим Insert (вставка) нажатием клавиши Ins; нажимая Spacebar (пробел), переместить информацию.

Для перемещения только атрибутного окна следует: спозиционировать курсор на начало окна и нажать Enter; передвинуть курсор на новую позицию и вновь нажать Enter.

Для изменения размеров атрибутного окна на электронной доске без изменения длины атрибута в структуре отношения следует: спозиционировать курсор на требуемое окно; для увеличения размеров окна в экранной форме необходимо нажать Ins; для уменьшения размеров окна в экранной форме необходимо нажать Del.

Для изменения размеров атрибута в структуре отношения нужно нажать F10 и перейти в Width опцию Modify меню (которое, в свою очередь, является позицией Screen меню).

Для удаления окна с электронной доски следует спозиционировать на него курсор и нажать Ctrl—U. Если это окно представляло некоторый атрибут отношения, то на экране появится диалоговое окно, спрашивающее, не следует ли удалить соответствующий атрибут из структуры отношения? Таким способом можно модифицировать структуру отношения прямо в процессе работы с электронной доской.

После каждых 24 строк экранной формы dBASE III Plus создает «мягкий» (имеется в виду программный) перевод страницы (в .fmt файл вставляется команда READ), индицируемый на экране обнулением текущего номера строки в статусной строке.

Опции. Рассмотрим опции (подменю) Screen-меню.

Прежде всего отметим, что некоторые позиции (опции) рассматриваемых подменю могут быть недоступны в определенных ситуациях. Например, опция Range (диапазон) подменю Modify (изменения) не имеет смысла для символьных атрибутов. При обработке символьного атрибута она недоступна. Визуально это выражается в том, что все доступные опции активного меню выделяются повышенной яркостью либо цветом, а активная опция — инверсией либо альтернативной парой цветов. Недоступные опции выдаются обычным цветом и носят лишь информационный характер (в данной ситуации).

Позиция Set up (установки). Это первое меню. Оно позволяет создать новое отношение или выбрать одно из существующих, на основе которого будет создаваться / редактироваться экранная форма. Если в момент вызова команды CREATE/MODIFY SCREEN некоторое отношение уже было активизировано, то нет необходимости выбирать его повторно.

Опция Choosing a database file (выбор отношения) активизирует список отношений каталога (если он открыт) или список отношений, содержащихся в текущем директории активного диска.

Опция Load Fields (загрузка атрибутов) сгружает все или выбранные атрибуты отношения на электронную доску. Для окончания выбора атрибутов и их загрузки необходимо нажать «Пробел». Атрибуты при загрузке располагаются на доске вертикально (имеется в виду — каждый следующий на новой строке под предыдущим) начиная с текущей позиции курсора и образуют соответствующие поля. Таким образом, в любой момент, находясь в режиме редактирования информации на электронной доске, можно спозиционировать курсор в нужное место, затем нажать F10, перейдя в режим меню, загрузить требуемый атрибут (т. е. образовать для него окно) либо несколько атрибутов, а затем разнести окна по местам.

Позиция Modify (изменения). Это меню позволяет вводить имена атрибутов непосредственно на электронной доске и модифицировать структуру создаваемого / существующего отношения. Клавиша F10 переключает режимы меню и электронной доски.

Опция Action (действия) позволяет задать генератору экранных форм режим обработки окна: вывод информации — Display/SAY режим; ввод/редактирование — Edit/GET режим.

Соответственно выбору режима в форматном файле будет сгенерирована для данного места либо @. . .SAY. . . , либо @. . .GET. . . команда. По умолчанию установлен режим Edit/GET.

Опция Source (источник) содержит имя активного отношения. Оно может быть изменено только через Set Up меню, описанное ранее.

Опция Content (содержание) содержит имя атрибута, для которого требуется установить режим (Edit/GET или Display/SAY) с помощью опции Action (действия). Для выбора атрибута следует нажать F10 — при этом активизируется список атрибутов. Как только будет выбран нужный атрибут, опции Type (тип), Width (размер) и Decimals (десятичные позиции) получат свои значения автоматически из структуры отношения. Если требуется изменить любой из этих параметров атрибута, следует ввести новые значения соответствующей опции, и в структуре отношения будут сделаны необходимые изменения. В предлагаемом списке атрибутов есть позиция New Field (новый атрибут). При выборе этой позиции значения типа, длины и, возможно, количества позиций после десятичной точки необходимо заполнить самостоятельно. Этот атрибут затем войдет в структуру отношения.

При работе с внешним представлением (композицией нескольких отношений — .vue файл) недоступны все опции, модифицирующие структуру отношения.

Опция Type (тип) определяет тип атрибута: символьный, числовой, логический, дата или текст (memo). При обработке атрибутов типа memo для них выделяется фиксированное окно (четыре позиции, в которых записано слово «memo»). Размер окна изменить нельзя, а местоположение — можно. При использовании форматного файла пользователь, подойдя курсором к окну «memo», имеет возможность нажать Ctrl — Home, и тогда будет вызван текстовый

процессор по обработке memo-полей, который выполнит ввод — редактирование информации. В Clipper с этой целью существует специальная функция MEMOEDIT (см. гл. 3).

Опция Width (размер) определяет размер атрибута, а Decimals (десятичные позиции) — число позиций после точки для числовых атрибутов.

Опции Picture function (форматная функция) и Picture template (форматный шаблон) дают возможность задать формат вводимых/выводимых значений, используя для этого допустимые функции и/или шаблоны. Например, можно ограничить ввод для значений некоторого атрибута только цифрами или буквами или можно потребовать, чтобы отрицательные значения некоторого атрибута при выводе заключались в скобки и т. п. Подробная информация по конструированию PICTURE опций приведена в описании команды @. . . SAY. . .GET.

Опция Range (диапазон) активна только для числовых атрибутов и атрибутов типа дата. Она позволяет задать минимальное и максимальное значения, в пределах которых вводимая информация считается допустимой (верной), в противном случае dBASE III Plus выдает сообщение об ошибке ввода, указывает допустимый диапазон и просит повторить ввод.

Clipper позволяет дополнительно включать в процедуру контроля ввода опцию VALID (см. команду @. . .SAY. . .GET. . .).

Позиция Options (опции). Это меню позволяет создать текстовый файл, документирующий создаваемый/редактируемый форматный файл, а также строить вертикальные и горизонтальные линии и прямоугольники на электронной доске (расчерчивать таблицы и т. п.).

Опция Create text file image (создание текстового файла-описателя) помогает документировать содержимое экранной формы. Это отложенная операция. При выходе из генератора экранных форм помимо .scr и .fmt файлов создается также текстовый файл .txt (с тем же именем, что и остальные). Он содержит информацию о каждом использованном в форматном файле атрибуте: сведения о нем как об атрибуте отношения, а также размер и положение (строка, столбец) соответствующего окна, действие, которое будет проводиться в этом окне (вывод значений или ввод — редактирование), и сопровождающие это действие процедуры контроля, если таковые заданы (формат — шаблоны — функции, диапазон значений). Кроме того, этот текстовый файл содержит собственно экранную форму.

Опции Single bar (одинарная окаймляющая) и Double bar (двойная окаймляющая) дают возможность провести линию (одинарную или двойную) либо прямоугольник, окаймляющий нужную область экранной формы (его периметр также будет проведен одинарной или двойной линией — в зависимости от выбора опции). После выбора одной из этих опций dBASE III Plus автоматически осуществляет переход в режим электронной доски из режима меню. Для построения необходимо: установить курсор в позицию начала линии или угла прямоугольника; нажать Enter; спозиционировать курсор в конец линии (вертикальной или горизонтальной) либо диагональный угол прямоугольника и вновь нажать Enter.

Можно изменить размеры любого прямоугольника на электронной доске. Для этого необходимо: спозиционировать курсор в тот угол, который нужно передвинуть; нажать Enter; передвинуть курсор к новой позиции угла; нажать Enter для «перечерчивания прямоугольника».

Позиция Exit (выход). Опция Save осуществляет выход с сохранением всей проделанной работы (создаются/модифицируются .scr, .fmt файлы и, если было задано, .txt файл). Это эквивалентно нажатию Ctrl — End. При этом .fmt файл автоматически активизируется. Опция Abandon уничтожает создаваемый .scr файл либо изменения, сделанные в нем, если он существовал. Файл .fmt не генерируется. Это эквивалентно нажатию Esc в любой момент в процессе работы.

Замечания по программированию. Можно использовать команду CREATE/MODIFY SCREEN для генерации серии команд @ . . .GET. . .SAY. . . с целью использовать их в дальнейшем в программе. В этом случае после сохранения .fmt файла следует скопировать его содержимое в .prg файл и добавить необходимые команды.

См. также @. . .SAY. . .GET. . . , CREATE, MODIFY STRUCTURE, SET CATALOG, SET FORMAT.

2.2.34. CREATE/MODIFY VIEW (только dBASE III Plus)

Эта команда (CREATE или MODIFY) инициирует полноэкранный меню-управляемый режим, предназначенный для связывания нескольких отношений базы данных (связи устанавливаются по однородным информативно связанным атрибутам либо по номерам кортежей) в единое внешнее представление, эффективно используемое в разрабатываемых приложениях (см. .vue файлы в п. 1.5). Такое внешнее представление затем может трактоваться для некоторых целей (например, просмотр или редактирование) как виртуальное отношение. Объект внешнее представление определен лишь в dBASE III Plus, для Clipper подобный объект может быть создан командным путем с помощью команд SET RELATION. . . , SET FILTER. . . и др.

Синтаксис. CREATE/MODIFY VIEW ⟨.vue filename⟩/?

Если расширение не указано, dBASE III Plus подразумевает .vue.

Если открыт каталог и SET CATALOG в положении ON (значение по умолчанию), то .vue файл добавится к открытому каталогу.

Использование. Для построения нового внешнего представления используется CREATE форма команды, а MODIFY используется для редактирования существующего внешнего представления. Если требуется изменить существующее внешнее представление из открытого каталога, но забыто его имя, можно воспользоваться параметром ? команды, что приведет к выдаче списка всех внешних представлений, зафиксированных в открытом каталоге.

При выходе из View меню (главное меню генератора внешних представлений) по опции Save (сохранение) dBASE III Plus автоматически активизирует вновь созданное либо отредактированное внешнее представление, используя команду SET VIEW TO ⟨.vue filename⟩. Для его деактивации (если это нужно) следует использовать команду CLOSE DATABASES. Это приведет к закрытию всех открытых .dbf, .ndx и .fmt файлов без закрытия открытого каталога (если он есть и открыт).

Любое внешнее представление содержит следующие элементы: имена отношений с любыми индексными файлами и связанную с каждым отношением рабочую область; связи между отношениями;

выбранные имена атрибутов из каждого отношения (атрибуты, участвующие в виртуальном отношении, образованном внешним представлением).

Дополнительно внешнее представление может содержать: один форматный файл (.fmt), одно фильтрующее условие.

Опции. Рассмотрим главное меню генератора внешних представлений. Оно содержит пять позиций (подменю).

Позиция Set up (установки). Выдает список отношений в открытом каталоге либо в текущем директории активного диска (если нет открытого каталога).

Если каталог открыт, то можно выбрать до девяти отношений, образующих внешнее представление (так как сам каталог занимает десятую рабочую область). Если нет открытого каталога, то можно включить во внешнее представление до десяти отношений. Как правило, их число не превосходит трех, так как иначе очень сложно прослеживаются взаимосвязи отношений (т. е. логическая структура БД очень запутана).

Для выбора отношения следует спозиционировать курсор на его имя и нажать Enter. Если отношение имеет связанные с ним индексные файлы, то выдается их список, что позволяет выбрать те из них, которые необходимы для определения индексного механизма представления данного отношения в рамках создаваемого/модифицируемого внешнего представления.

Клавиша Enter позволяет выбрать файл (помечая его слева специальным символом-указателем) либо отменить выбор (повторное нажатие уничтожает отметку).

dBASE III Plus сохраняет информацию о том состоянии, в котором находилась система до активизации генератора внешних представлений (т. е. о том, в каких рабочих областях какие отношения были открыты, с какими индексами и т. п.), но открытые отношения не предлагаются изначально помеченными для включения во внешнее представление в меню Set Up. Если требуется создать/модифицировать внешнее представление на основании текущего состояния системы, следует использовать команду CREATE VIEW FROM ENVIRONMENT. Однако при выходе из CREATE/MODIFY VIEW по Esc dBASE III Plus восстановит состояние системы, каким оно было до входа в команду (напомним, что при «благополучном» выходе из генератора внешних представлений вновь созданное/модифицированное внешнее представление активизируется).

Позиция Relate (связи). При открытии Relate меню возникает список имен всех выбранных в меню Set Up отношений и задаются связи между ними. Из выбранных отношений создается цепочка, реализующая требуемое внешнее представление, по следующему алгоритму.

Выбирается отношение, которое будет ведущим во внешнем представлении (это активизирует список оставшихся отношений).

Выбирается отношение, которое будет следующим элементом в цепочке (появляется подсвеченная зона, приглашающая ввести имя атрибута или выражение, по которому будет установлена связь ведущего отношения со вновь выбранным).

Если требуется связать отношения просто по значениям некоторого атрибута (имеется в виду по совпадениям значений), нужно нажать F10, что вызовет выдачу списка совпадающих атрибутов. Если связываемое с ведущим отношение проиндексировано, то в строке сообщений появится выражение, описывающее ключ индексации. Связывающее выражение должно иметь тот же тип, что и ключ индексации. Если же это отношение неиндексировано (или

было выбрано без индекса), то выражение должно иметь числовое значение, чтобы связать два отношения просто по номеру кортежа. Обычно такое выражение строится на основе функции RECNO( ) (но это не обязательно).

Следует нажать «стрелку влево» или «стрелку вправо», чтобы выйти на верхний уровень Relate меню, либо повторить все, описанное ранее, чтобы продлить цепочку. При этом ведущим необходимо выбрать то отношение, которое раньше было ведомым, иначе цепь разорвется и станет невозможно создать внешнее представление. После выбора отношения dBASE III Plus выдаст список тех отношений, которые еще не участвовали в цепочке.

Отношения могут связываться лишь в простую цепочку и никак иначе. Отношение, уже включенное в состав цепи, не может появиться в ней вторично. Однако следует отметить, что в Clipper команда SET RELATION позволяет организовывать и нерархические структуры вида отец — сын1, отец — сын2 и т. п.

Для цепочки из двух отношений описанный выше алгоритм аналогичен (примерно) следующему набору dBASE команд:

```
. SELECT 1
. USE Отношение 1
. SELECT 3
. USE Отношение 3 INDEX Ключ _ индексации _ отношения 3
. SELECT 1
. SET RELATION TO Отношение 1 имя _ атрибута/выражение;
      INTO Отношение 2
```

Позиция Access (доступ). Это меню дает возможность выбрать из образованной цепочки те атрибуты, которые будут доступны при активизации внешнего представления. При выборе Access меню возникает список всех отношений, образующих цепочку. Выбирается отношение, получается список атрибутов (изначально считается, что все они входят во внешнее представление, что отмечается специальным символом-указателем), выбираются те атрибуты, которые не должны быть доступны в конструируемом внешнем представлении, и нажимается Enter напротив их имен. Повторное нажатие Enter возвращает атрибут в список доступных. Внутренне эта процедура вызывает выполнение команды SET FIELDS TO.

Позиция Options (опции). Это меню дает возможность присоединить к внешнему представлению форматный файл и/или фильтрующее условие. Обычно для форматных файлов им пользуются не в процессе создания внешнего представления, а уже в процессе его редактирования, так как необходимо вначале создать внешнее представление, затем форматный файл для этого внешнего представления (команды CREATE SCREEN либо MODIFY COMMAND), а затем, вызвав внешнее представление вновь для редактирования, подключить к нему созданный для него форматный файл. Фильтрующее условие можно создать/модифицировать прямо внутри этого меню.

Опция Format (формат) дает возможность выбрать из списка имеющихся форматный файл и подключить его к внешнему представлению. Выбранный командный файл всегда впоследствии будет активизироваться при активизации данного внешнего представления командой SET VIEW TO ⟨.vue filename⟩. Форматный файл может включать только те атрибуты внешнего представления (виртуального отношения, образованного данным внешним представлением), которые доступны (выбраны в Access меню).

Опция Filter (фильтр) позволяет определить фильтрующее условие, определяющее, какие из кортежей виртуального отношения следует включить во внешнее представление (см. SET FILTER TO для уточнения подробностей). Область ввода допускает горизонтальный скролинг, что дает возможность ввести логическое выражение длиной до 254 символов в качестве условия фильтрации. Важной особенностью Filter опции является то, что при конструировании условия выборки можно использовать и те атрибуты виртуального отношения, которые недоступны во внешнем представлении.

Позиция Exit (выход). Опция Save (сохранение) позволяет сохранить результаты работы (эквивалентно нажатию Ctrl — End), а опция Abandon (уничтожение) — уничтожить их (эквивалентно нажатию Esc в любой момент). Если осуществлен выход с уничтожением, то dBASE III Plus восстанавливает предыдущее состояние, если же с сохранением, то активизируется внешнее представление.

Замечание. При активизации внешних представлений одновременно открывается значительное количество файлов. В каждом конкретном случае следует убедиться, что операционное окружение позволяет это сделать (файл CONFIG. SYS содержит парамагр Files, который рекомендуется установить в значение не менее 20, что даст возможность dBASE III Plus одновременно обрабатывать до 15 файлов — предельное значение для dBASE III Plus). Можно воспользоваться для консультации имеющейся документацией по настройке операционной системы.

При активизации внешнего представления одновременно открываются все составляющие его .dbf, .ndx файлы, возможно, и .fmt файл. Сам .vue файл остается открытым ровно столько, сколько необходимо, чтобы активизировать все содержащиеся в нем объекты, а затем закрывается, т. е. в рамках одной команды SET VIEW TO.

См. также SELECT, SET CATALOG, SET FIELDS, SET FILTER, SET FORMAT, SET INDEX, SET RELATION, SET VIEW, USE.

2.2.35. CREATE VIEW FROM ENVIRONMENT (только dBASE III Plus)

Эта команда строит внешнее представление (.vue файл) исходя из текущего состояния систем (имеются в виду открытые в определенных рабочих областях отношения, используемые индексы, установленные внешние схемы (команда SET FIELDS), установленные связи (SET RELATION), открытый форматный файл, если есть). Затем это состояние системы может быть восстановлено в любой момент.

Синтаксис. CREATE VIEW ⟨.vue filename⟩ FROM ENVIRONMENT

Если расширение не указано, dBASE III Plus подразумевает .vue. Все атрибуты открытых отношений включаются во внешнее представление (будут доступны — см. CREATE/MODIFY VIEW), если предварительно не были даны соответствующие команды SET FIELDS TO.

Использование. CREATE VIEW строит внешнее представление со следующим содержимым: все открытые отношения и индексы с запоминанием номера рабочей области по каждому; все связи,

установленные между отношениями; номер активной рабочей области; установленные внешние схемы для открытых отношений (см. команду SET FIELDS TO); открытый форматный файл (если он был).

Для деактивации внешнего представления можно открыть другое командой SET VIEW TO либо дать команду CLOSE DATABASES.

Для изменения содержимого внешнего представления, созданного описанным выше способом, следует использовать команду MODIFY VIEW (см. CREATE/MODIFY VIEW).

См. также SELECT, SET FIELDS, SET FORMAT, SET INDEX, SET RELATION, SET VIEW, USE.

2.2.36. DECLARE (только Clipper)

Эта команда позволяет создать в памяти структуру типа массив в Clipper. В dBASE III Plus понятие массивов не введено и нет средств работы с ними, в случае необходимости их можно ограниченно моделировать с помощью простых переменных памяти. Использование массивов в программе на Clipper делает ее неинтерпретируемой в среде dBASE III Plus.

Синтаксис. DECLARE ⟨memvar⟩[⟨expN⟩] [, ⟨array list⟩]

Использование. Команда создает массив из ⟨expN⟩ элементов. Элементы массива являются, по сути, проиндексированными переменными памяти. Элементы одного массива могут иметь различные типы, т. е. понятие массива здесь представляет смесь понятий одномерного массива и структуры в языках программирования. По «программной видимости» массивы во всех случаях являются внутренними переменными (private) некоторой процедуры.

Квадратные скобки [ ], использованные с ⟨expN⟩ при задании синтаксиса команды, не индицируют здесь опциональный синтаксис. Они являются неотъемлемой частью команды.

Одна DECLARE команда может определить несколько массивов.

К элементам объявленного массива можно обращаться далее в программе посредством указания имени массива и индекса в квадратных скобках.

Функция LEN ( ) возвращает число элементов в массиве, если в качестве параметра задать только имя массива без индекса, иначе — длину конкретного элемента, определяемого заданным индексом.

Функция TYPE ( ) для массива вернет значение «A».

Присвоение значения переменной памяти с тем же именем, что и массив (по сути, ее объявление, так как язык dBASE не содержит средств явного объявления переменных памяти (кроме массивов в Clipper, для которых, однако, объявляется только размер, а типы элементов задаются тоже с помощью присвоения некоторых значений)), уничтожает массив и освобождает всю занятую им память.

Массивы не могут быть сохранены в .mem файлах (см.п. 1.5) и не могут быть переданы по ссылке как параметры в командах CALL и DO.

Выражения, включающие массивы, могут использоваться в макроаппарате.

2.2.37. DELETE (dBASE III Plus и Clipper)

Команда DELETE помечает определенные кортежи в отношении признаком удаления.

Синтаксис. DELETE [⟨scope⟩] [WHILE ⟨condition⟩] [FOR ⟨condition⟩]

Если опция ⟨scope⟩ и FOR, WHILE условия опущены, то только текущий кортеж активного отношения помечается признаком удаления.

Использование. При выдаче содержимого отношения командами DISPLAY и LIST кортежи, помеченные признаком удаления, идентифицируются выдачей «звездочки» (*) в первой позиции. Команда RECALL позволяет восстановить кортежи, помеченные признаком удаления. Команда PACK позволяет удалить их окончательно из отношения. Таким образом, удаление в dBASE проводится в два этапа: вначале команда DELETE помечает кортеж признаком удаления, что исключает его из поля зрения некоторых команд, однако он остается в отношении и может быть восстановлен, а затем команда PACK удаляет помеченные к удалению кортежи и сжимает отношение.

При использовании команд покортежной обработки, выходящих в режим экранного редактирования, таких, как, например, BROWSE или EDIT, кортежи, помеченные к удалению, индицируются появлением *Del в статусной строке. В таком режиме нажатие Ctrl—U работает как переключатель — помечает кортеж к удалению, если он не был помечен, и наоборот, повторное нажатие вызывает обратное действие.

Команда DELETE не изменяет положение указателя текущего кортежа. Поэтому если указатель был ранее установлен в конец отношения (функция EOF ( ) = .T.), как, например, после LIST или DISPLAY ALL, использование DELETE не будет иметь эффекта.

См. также DELETED ( ), PACK, RECALL, SET DELETED ON.

2.2.38. DIR (dBASE III Plus и Clipper)

Команда DIR выводит имя отношения, число кортежей в нем, дату последнего обновления и размер файла (в байтах) для каждого файла, содержащего отношение на указанном диске в указанном директории. Эту команду также используют для получения файлового оглавления аналогично команде DIR в DOS. В этом случае помимо выдачи списка файлов по заданному шаблону (аналогично DOS — ? обозначает любой символ в этой позиции, а * — переменное количество символов до конца имени или расширения) в указанном директории указанного диска выводится также количество найденных файлов, их общий размер в байтах и количество свободной памяти на указанном диске (в байтах).

Синтаксис. DIR [⟨drive:⟩] [⟨path⟩] [⟨sceleton⟩]

Если параметры опущены, то выдается список отношений в текущем директории активного диска.

Использование В директории может быть не отражена информация, связанная с добавлением/исключением кортежей в отношение, если это отношение не закрыто.

Альтернативой команды DIR для dBASE III Plus является команда DISPLAY FILE. Ее полный синтаксис

DISPLAY FILE [LIKE ⟨sceleton⟩] [TO PRINT]

Специальные случаи. Если в CONFIG. SYS имеется параметр Files-20 и открыто более 15 файлов (имеется в виду dBASE III Plus), то в графе «количество кортежей» (# records) при выводе списка отношений DIR проставляет «not readable» (нет возможности прочесть).

2.2.39. DISPLAY (dBASE III Plus и Clipper)

Команда DISPLAY используется для вывода содержимого отношения базы данных.

Синтаксис. Для dBASE III Plus:

```
DISPLAY [⟨scope⟩] [⟨expression list⟩]
        [WHILE ⟨condition⟩]
        [FOR ⟨condition⟩] [OFF] [TO PRINT]
```

Для Clipper:

```
DISPLAY [⟨scope⟩] ⟨expression list⟩
        [WHILE ⟨condition⟩]
        [FOR ⟨condition⟩] [OFF]
        [TO PRINT/TO FILE ⟨filename⟩]
```

Для Clipper обязательным является указание списка выводимых атрибутов-выражений. В отличие от dBASE III Plus Clipper не выводит имена атрибутов как заголовок вывода. dBASE III Plus предваряет вывод информации по команде DISPLAY таким заголовком.

Если опция ⟨scope⟩ и FOR, WHILE условия опущены, то будет выведена информация (которая специфицирована в ⟨expression list⟩) только по текущему кортежу. Если ⟨expression list⟩ опущен (что возможно только для dBASE III Plus), то выдаются значения всех атрибутов отношения. Если параметр OFF не задан, то выводятся номера кортежей (перед каждым кортежем).

Использование. Если выводится более 20 кортежей, DISPLAY делает паузу через каждые 20 строк, выдавая сообщение «Press any key to continue. . .» («Нажмите любую клавишу для продолжения. . .»).

Содержимое memo-атрибутов выводится только в случае явного задания их имен в ⟨expression list⟩. Если ⟨expression list⟩ пуст (dBASE III Plus), то в поле значений атрибута типа memo для каждого кортежа выводится имя этого атрибута. Если же memo-атрибут явно указан в списке выводимых выражений, то его значение для каждого кортежа выводится, причем длина поля вывода составляет 50 символов (по умолчанию, ее можно изменить — см. команду SET MEMOWIDTH).

Если найден конец отношения (EOF( )=.T.), а удовлетворяющие условию вывода кортежи обнаружить не удалось, то в случае Clipper не выводится ничего, а для dBASE III Plus — только заголовок.

Когда вывод кортежа (или указанной в ⟨expression list⟩ информации) занимает более 80 символов (длина строки дисплея), содержимое переносится на следующую строку и т. д.

Специальные случаи. Каждая графа вывода в dBASE III Plus имеет заголовок. Если выводится результат некоторого выражения, то в качестве заголовка высвечивается само выражение, причем так, как оно было указано в тексте команды (имеются в виду строчные и прописные буквы).

См. также SET MEMOWIDTH, SET HEADING.

2.2.40. DISPLAY FILE (только dBASE III Plus)

См. описание команды DIR.

2.2.41. DISPLAY HISTORY (только dBASE III Plus)

Эта команда выводит список команд, которые были выполнены и запомнены в режиме HISTORY («предыстория»).

Синтаксис. DISPLAY HISTORY [LAST ⟨expN⟩] [TO PRINT]

Число выполненных команд, запомненных в «предыстории» по умолчанию, 20. Это число можно изменить с помощью команды SET HISTORY TO.

Использование. Эта команда позволяет просматривать список последних N команд начиная с более давних и кончая наиболее свежими (предыдущей).

При заполнении экрана выдача списка приостанавливается до нажатия любой клавиши.

Опции. DISPLAY HISTORY выводит все команды, которые хранятся в «предыстории», если не задан параметр LAST. В случае его наличия выводится только указанное в числовом выражении количество наиболее «свежих» команд.

См. также LIST HISTORY, SET DOHISTORY, SET HISTORY, SET MEMOWIDTH.

2.2.42. DISPLAY MEMORY (только dBASE III Plus)

Эта команда осуществляет выдачу информации о переменных памяти: имя, тип, размер и статус каждой активной переменной памяти. Она также выдает число активных переменных, число переменных, которые еще могут быть созданы (дополнение до максимума — 256), объем занимаемой памяти и объем памяти, который еще может быть занят (дополнение занятой переменными памяти до максимального значения).

Синтаксис. DISPLAY MEMORY [TO PRINT]

Использование. Максимальное число переменных, активных в данный момент (для dBASE III Plus),— 256. Максимум 6000 байт памяти используется для переменных памяти, однако это ограничение может быть изменено.

Если в процессе вывода экран заполняется информацией, DISPLAY MEMORY останавливает вывод, выдает сообщение: «Press any key to continue. . .» («Нажмите любую клавишу для продолжения. . .») и ожидает ввода.

Вывод для числовых переменных содержит и внешнее и внутреннее представления переменной (которые могут существенно различаться).

См. также PRIVATE, PUBLIC.

2.2.43. DISPLAY STATUS (только dBASE III Plus)

Выводит информацию о текущем состоянии системы.

Синтаксис. DISPLAY STATUS [TO PRINT]

Использование. Для каждого открытого отношения DISPLAY STATUS выводит: имя отношения; номер рабочей области; второе имя; связи с другими отношениями; открытые индексные файлы;

ключ индексации для каждого открытого индексного файла; открытые .dbt файлы (содержащие значения memo-атрибутов отношений).

Эта команда также выводит: текущий «путь», используемый при поиске файлов во внешней памяти; код активного диска; назначение для печати; текущие установки большинства переключателей ON/OFF, выставляемые по SET командам; назначение для DEVICE (SCREEN или PRINT), установку левой границы печати; назначения функциональных клавиш.

2.2.44. DISPLAY STRUCTURE (только dBASE III Plus)

Выводит следующую информацию об активном отношении: имя файла, число кортежей, дату последнего изменения, описание каждого атрибута и число байтов в записи (кортеже).

Синтаксис. DISPLAY STRUCTURE [TO PRINT]

Использование. Общее число байтов в кортеже (записи) — сумма длин всех атрибутов плюс один байт. Дополнительный байт используется для признака удаления.

Если число атрибутов в отношении превышает 16, то DISPLAY STRUCTURE останавливает вывод через каждые 16 описаний атрибутов и требует нажать любую клавишу для продолжения.

DISPLAY STRUCTURE помечает имя атрибута специальным символом (код 16 ASCII), если атрибут входит в список SET FIELDS в случае, когда SET FIELD ON.

2.2.45. DO (dBASE III Plus и Clipper)

DO вызывает передачу управления подпрограмме (другому командному файлу или процедуре) с указанием параметров.

Синтаксис. DO ⟨.prg filename⟩/ ⟨procedure name⟩
[WITH ⟨parameter list⟩]

Для dBASE III Plus должен быть указан код диска, если вызываемый файл находится не на активном диске; если расширение не задано, dBASE III Plus подразумевает .prg.

Использование. Для Clipper команда DO аналогична команде CALL. Подпрограмма может быть оформлена как отдельный командный файл либо как процедура на языке dBASE (может быть как в данном .prg файле, так и в другом), либо как процедура на другом языке программирования. Соглашения о передаче параметров для команд DO и CALL идентичны для Clipper. Для dBASE III Plus это две разные команды: CALL используется для вызова программ поддержки, загружаемых командой LOAD, а DO вызывает «вложенные» (по структуре программы) командные файлы или процедуры, которые, по существу, также содержатся в некотором специальном «процедурном» командном файле. Для dBASE III Plus передача управления (вызов, загрузка) происходит непосредственно в процессе интерпретации, а для Clipper все модули связываются в единый загрузочный модуль (или оверлейную структуру) на этапе линкования объектных модулей.

Для dBASE III Plus:

каждый DO файл считается как один открытый файл (общее число открытых файлов в dBASE III Plus не может превышать 15). При использовании процедурного файла открывается лишь один

файл — тот, имя которого указано в команде SET PROCEDURE TO ⟨filename⟩;

DO не может вызывать открытые ранее командные файлы (т. е. рекурсия запрещена);

когда программа, вызванная по DO, завершена, управление передается вызывающей программе или происходит переход в диалоговый режим, если программа была вызвана оттуда;

опция WITH позволяет передать параметры вызываемой подпрограмме. Список параметров может содержать любое значащее выражение dBASE III Plus.

Для Clipper:

при передаче массивов и переменных памяти в качестве параметров имеют место следующие правила:

функциям параметры всегда передаются по значению;

параметры передаются процедурам и вызываемым программам по ссылке только в том случае, если передается идентификатор. Если передается выражение, то параметр передается по значению. Элементы массива могут быть переданы только по значению;

параметр, передаваемый по ссылке, может быть изменен вызываемой программой, а передаваемый по значению является копией, и изменение этой копии не изменяет оригинал.

См. также CALL, EXTERNAL, FUNCTION, MODIFY COMMAND, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC, SET PROCEDURE.

2.2.46. DO CASE (dBASE III Plus и Clipper)

Команда является средством структурного программирования, позволяя выбрать одну группу действий из набора групп в зависимости от значения параметра выбора.

Синтаксис. DO CASE
CASE ⟨condition⟩
⟨commands⟩
[CASE ⟨condition⟩]
⟨commands⟩
[OTHERWISE]
⟨commands⟩
ENDCASE

Использование. DO CASE является составной командой. ENDCASE завершает структуру DO CASE. Внутрь команды DO CASE могут быть вложены другие составные команды: DO CASE. . .ENDCASE, IF. . .ENDIF, DO WHILE. . .ENDDO. При этом должны соблюдаться стандартные правила вложения (команды должны быть целиком вложены в CASE блок).

Если условие очередного CASE блока истинно, то он выполняется, и управление передается команде, следующей за ENDCASE. Если все условия ложны, то выполняется OTHERWISE (если он есть) и осуществляется выход из DO CASE.

См. также DO, DO WHILE, IF, MODIFY COMMAND.

2.2.47. DO WHILE (dBASE III Plus и Clipper)

DO WHILE является командой структурного программирования, которая позволяет организовать циклическое выполнение группы команд. В Clipper дополнительно есть команда FOR. . .NEXT. . .

Синтаксис. DO WHILE ⟨condition⟩
⟨commands⟩
ENDDO

Использование. DO WHILE является составной командой. ENDDO завершает структуру DO WHILE. Внутрь команды DO WHILE могут быть вложены другие составные команды: DO CASE. . .ENDCASE, IF. . .ENDIF, DO WHILE. . .ENDDO. При этом должны соблюдаться стандартные правила вложения (команды должны быть целиком вложены в DO WHILE блок).

После ENDDO в той же строке могут идти любые комментарии.

Цикл WHILE выполняется, пока истинно WHILE выражение. Как только оно становится ложным, управление передается команде, следующей после ENDDO.

Можно использовать макросредства при конструировании WHILE выражения, только если входящие туда переменные не изменяются внутри цикла.

См. также FOR. . .NEXT. . . , LOOP, EXIT, RETURN

2.2.48. EDIT (только dBASE III Plus)

Команда EDIT позволяет модифицировать содержимое кортежей активного отношения в режиме экранного редактирования.

Синтаксис. EDIT [⟨scope⟩] [FIELDS ⟨list⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]

Если команда вызывается без параметров, то (по умолчанию) предполагается редактирование текущего кортежа активного отношения.

Использование. EDIT использует стандартные команды управления курсором режима экранного редактирования. Клавиши направлений передвигают курсор по кортежу, PgUp возвращает к редактированию предыдущего кортежа, а PgDn переводит к редактированию следующего. Ctrl — End осуществляет выход из EDIT с сохранением всех сделанных изменений, а Esc позволяет сохранить все изменения, кроме сделанных в текущем кортеже в последнем его просмотре.

Для редактирования значений memo-атрибута следует нажать Ctrl — PgDn, спозиционировав предварительно курсор в начало поля, представляющего значение этого атрибута в кортеже. Для редактирования подключится тот текстовый процессор, который назначен для обработки значений memo-атрибутов, либо стандартный процессор MODIFY COMMAND dBASE III Plus, если не назначен никакой.

Команды EDIT и CHANGE идентичны.

См. также MODIFY COMMAND, SET FORMAT TO, CHANGE SET FIELDS.

2.2.49. EJECT (dBASE III Plus и Clipper)

Команда приводит к прогону страницы на принтере.

Синтаксис. EJECT

Использование. EJECT посылает код 12 ASCII (Form Feed) на принтер. Использование этой команды, когда принтер не подключен, не включен или не готов, «подвешивает» компьютер, ожидающий готовности принтера, до ее (готовности) появления.

EJECT сбрасывает в ноль значения функций PROW ( ) и PCOL ( ).

См. также SET PRINT, SET DEVICE.

2.2.50. ERASE (dBASE III Plus и Clipper)

ERASE удаляет файл из дискового директория.

Синтаксис. Для dBASE III Plus:
ERASE ⟨filename⟩/?
Для Clipper:
ERASE ⟨filename⟩

Использование. Эта команда — аналог команды DELETE FILE. Имя файла должно быть указано с расширением, следует также указать код диска и путь, если файл находится не в текущем директории активного диска. В случае открытого каталога возможно использование ERASE ? для удаления некоторых из содержащихся в нем файлов (будет выдано меню — список файлов в каталоге).

Команда не позволяет удалить открытый файл (например, открытое отношение или индекс), а также не позволяет использовать символы-шаблоны, допустимые ДОС для множественного удаления.

См. также CLOSE, USE.

2.2.51. EXIT (dBASE III Plus и Clipper)

EXIT передает управление из тела цикла DO WHILE команде, непосредственно следующей за ним.

Синтаксис. EXIT

См. также DO WHILE, LOOP, MODIFY COMMAND.

2.2.52. EXPORT (только dBASE III Plus)

EXPORT строит из активного отношения dBASE III Plus внешний файл формата PFS. Команда предполагает знакомство с пакетом PFS.

Синтаксис. EXPORT TO ⟨filename⟩ TYPE PFS

Создаваемый PFS файл имеет то же имя, что и отношение, по которому он создается.

Использование. EXPORT требует наличия активного отношения.

Формат полей в создаваемом PFS файле зависит от ряда факторов.

Если отношение было создано командой IMPORT, то оно имеет связанный с ним формат (.fmt файл) в каталоге, который (формат) автоматически открывается, когда отношение активизируется (командой USE). Если этот формат не отменен командой SET FORMAT TO или SET FORMAT ⟨другой файл⟩, то EXPORT использует заданную в нем форму.

Если отношение создавалось не командой IMPORT, но открыт некоторый форматный файл, то EXPORT использует его данные.

Если форматный файл не открыт и нет никаких форматных файлов, связанных с этим отношением в открытом каталоге, то EXPORT использует формат, образованный на основе анализа структуры отношения.

Если TO файл уже существует и SET SAFETY ON, то dBASE III Plus выдаст запрос на подтверждение переписи файла. Если же SET SAFETY OFF, то файл перепишется автоматически.

Хотя может показаться, что команда EXPORT является аналогом команды COPY, но на самом деле это не так. Эта команда обрабатывает активное отношение и его подчиненные файлы, объединяя их (строя по ним) в единый PFS файл. Исходные файлы dBASE III Plus остаются неизменными после EXPORT.

PFS файлы обычно не имеют расширения. Если не используется расширение в создаваемом PFS файле, нужно поставить точку после последнего символа в имени файла.

См. также COPY TO, IMPORT, SET FORMAT, SET SAFETY.

2.2.53. EXTERNAL (только Clipper)

Эта команда используется для объявления имен для линкера.

Синтаксис. EXTERNAL ⟨procedure list⟩

Использование. Процедуры, объявленные в команде EXTERNAL, могут быть помещены в оверлейный модуль, однако вызываться по макроподстановке. Эта команда добавлена в Clipper в реализации Autumn' 86.

2.2.54. FIND (dBASE III Plus и Clipper)

Эта команда осуществляет поиск по индексу первого кортежа в активном отношении, удовлетворяющего ключу поиска — символьной строке или числу. FIND производит чрезвычайно быстрый поиск.

Синтаксис. FIND ⟨character string⟩/⟨numbar⟩

Использование. Команда SEEK аналогична команде FIND, но позволяет в качестве ключа поиска задать выражение (см. SEEK). В Clipper предпочтительнее использовать команду SEEK.

Команды FIND, SEEK, SKIP реализуют покортежную работу с отношением на базе индексного механизма доступа. При активизации индексного файла (см. USE. . .INDEX. . . , SET INDEX TO) доступ к кортежам отношения осуществляется через индекс, т. е. следующим является не физически следующий за данным кортеж, а кортеж, следующий за данным согласно ключу индексирования. Индексный механизм поддерживается средствами прямого доступа к файлу, обеспечиваемыми ДОС. Он позволяет чрезвычайно быстро найти кортеж с интересующим значением ключа индексирования (с помощью команды SEEK или FIND) и затем командами SKIP просматривать все кортежи с тем же (или рядом лежащим) значением ключа индексирования.

Команда LOCATE выполняет действия, аналогичные действиям, выполняемым командами FIND, SEEK, но не использует индексный механизм доступа, а осуществляет поиск простым последовательным перебором, что может занять значительное время в конкретных случаях.

Вернемся к рассмотрению команды FIND. Если кортеж с указанным ключом индексирования не найден, то указатель текущего кортежа указывает на последний кортеж в отношении, функция EOF ( ) принимает значение «Истина», а функция FOUND ( ) — «Ложь». Для dBASE III Plus в случае SET TALK ON выдается сообщение «No Find».

Специальные случаи. Заданная символьная строка берется в ограничители только в том случае, если ключ содержит ведущие пробелы.

FIND может искать только по первому символу (или первым символам) ключа.

Когда поиск предполагается производить по значению символьной переменной памяти, следует использовать макрофункцию (&). Если значение переменной может содержать ведущие пробелы, важные в задании ключа поиска, то нужно употребить следующую конструкцию: «& ⟨memvar⟩».

См. также INDEX, SEEK, SET INDEX, SET ORDER, EOF ( ), FOUND ( ).

2.2.55. FOR...NEXT... (только Clipper)

Эта команда позволяет организовать пошаговый цикл по счетчику.

Синтаксис. FOR ⟨memvar⟩ = ⟨expN⟩ TO ⟨expN⟩ [STEP ⟨expN⟩] ⟨commands⟩

. . .

NEXT

Использование. В dBASE III Plus эта команда моделируется с помощью цикла DO WHILE.

См. также DO CASE . . . ENDCASE, DO WHILE. . .ENDDO, IF. . .ENDIF, IF ( ), IIF ( ).

2.2.56. FUNCTION (только Clipper)

Это чрезвычайно важная команда Clipper, дающая возможность расширить язык путем доопределения пользователем собственных функций средствами самого языка dBASE (UDF — User Defined Functions — функции, определенные пользователем). В dBASE III Plus нет подобной возможности. В Clipper пользователь может описать ряд функций на языке dBASE (используя команду FUNCTION для их объявления), а также описать ряд необходимых ему функций на других языках программирования, например СИ и Макроассемблер. Эти функции можно скомпилировать отдельно и хранить в виде отдельных объектных модулей либо скомпоновать из них целые объектные библиотеки. Функции, написанные на dBASE, могут при желании компилироваться совместно с приложением.

Синтаксис. FUNCTION ⟨name⟩

. . . . .

RETURN (⟨value⟩)

Использование. Определенные пользователем функции так же, как и процедуры, могут иметь параметры. Эти функции могут использоваться в любом выражении при условии согласования типов: значения, возвращаемого функцией, и типа данных, уместного в данном контексте. Они могут использоваться в операторах присваивания, командах @. . .SAY. . . и т. д.

См. также PARAMETERS, а также описания функций расширения Clipper (гл. 3).

2.2.57. GO/GOTO (dBASE III Plus и Clipper)

Позиционирует указатель текущего кортежа на заданный кортеж в активном отношении.

Синтаксис. [GO/GOTO] ⟨expN⟩, или:

GO/GOTO BOTTOM/TOP

Использование. Если активизирован индексный механизм доступа, то TOP и BOTTOM указывают на первую и последнюю позиции по индексному файлу (а не на физически первый и последний кортежи отношения), а GO ⟨n⟩ позиционирует указатель на ⟨n⟩-й кортеж, а не на ⟨n⟩-ю позицию по индексу.

Если SET DELETED ON, то можно позиционировать курсор и на кортежи, помеченные признаком удаления.

2.2.58. HELP (только dBASE III Plus)

HELP — это меню-управляемый монитор, выдающий требуемую информацию о командах и использовании dBASE III Plus.

Синтаксис. HELP [⟨keyword⟩].

Использование. Команда HELP использует файл Help .dbs. Для получения помощи (информации о dBASE III Plus) необходимо иметь этот файл в директории, содержащем dBASE III Plus.

Keyword (ключевое слово) должно быть именем команды, функции dBASE III Plus, Help-экрана. Вызов HELP без параметров инициирует меню, позволяющее выбрать тему помощи. Можно просто нажать клавишу F1 для активизации монитора HELP.

Clipper не имеет средств получения помощи по самой системе программирования, однако позволяет очень эффективно включать помощь (Help-сопровождение) в разрабатываемое приложение.

Clipper позволяет включить специальную процедуру (или файл) HELP.PRG в разрабатываемое приложение. Если пользователь нажмет клавишу F1 в процессе ввода информации при интерпретации одной из команд: ACCEPT, INPUT, READ, MENU TO, WAIT, то Clipper вызовет процедуру HELP, передав ей следующие параметры:

1) call_prg — символьная переменная, содержащая имя процедуры (файла), откуда произошел вызов HELP;

2) line_num— числовая переменная, содержащая номер строки исходного текста этой процедуры (файла);

3) input_val — символьная переменная, содержащая имя переменной памяти, ожидающей пользовательского ввода.

Имена переменных, приведенные выше, не обязательны (можно использовать любые), однако все три параметра передаются HELP процедуре (их обязательно надо объявить — см. PARAMETERS, SET KEY). Рекурсивный вызов HELP может быть легко предотвращен помещением следующего фрагмента в начало процедуры HELP:

```
IF call _prg-"HELP"
   RETURN
ENDIF
```

Таким способом можно оснащать разрабатываемое приложение эффективным монитором предметно-ориентированной помощи, конструируя его по «контекстно-специализированному» (on — line — help), «обобщенному» (generalized help)или комбинированному методу организации интерфейса. Если включить вспомогательную информацию непосредственно в базу данных разрабатываемого

приложения в виде специальных атрибутов отношений или отдельных отношений,то такой монитор может иметь универсальный характер.

См. также SET HELP.

2.2.59. IF (dBASE III Plus и Clipper)

IF является командой структурного программирования и осуществляет условное выполнение команд. Команда IF — составная команда и должна оканчиваться ENDIF, образуя группу IF. . . ENDIF.

Синтаксис. IF ⟨condition⟩
⟨commands⟩
[ELSE
⟨commands⟩]
ENDIF

Использование. Любые составные команды, в том числе и сама группа IF. . .ENDIF, могут быть вложены (корректно) в группу IF. . .ENDIF.

В строке ENDIF следом за ENDIF может быть помещен любой комментарий (без предваряющего &&).

⟨condition⟩ (условие) представляет собой логическое выражение. Если оно принимает значение .T. (истина), то выполняются все последующие команды до достижения ELSE либо ENDIF (если ELSE не задано). Затем dBASE III Plus начинает выполнять команды после ENDIF. Если условие ложно (принимает значение .F.), то dBASE III Plus ищет ELSE фразу команды либо ENDIF (если ELSE опущено) и выполняет команды, идущие следом.

Если в программе описана последовательность вложенных IF команд, то первый ELSE относится к непосредственно предваряющему его IF.

См. также DO CASE, IIF ( ).

2.2.60. IMPORT dBASE III Plus

Команда создает набор объектов dBASE III Plus из PFS файла. Команда предполагает знакомство с PFS-пакетом.

Синтаксис. IMPORT FROM ⟨filename⟩ TYPE PFS

Создаваемые dBASE III Plus объекты будут иметь то же имя, что и PFS файл.

Если открыт каталог и SET CATALOG=ON (значение по умолчанию), то создаваемые объекты добавляются в каталог.

По умолчанию PFS файл разыскивается в активном директории активного диска.

Имя PFS файла должно включать и расширение, если оно есть, хотя PFS файлы не обязательно имеют расширения.

Использование. IMPORT по заданному PFS файлу строит три объекта (файла) dBASE III Plus — отношение (.dbf), форматный файл (.fmt) и внешнее представление (.vue). Все они имеют то же имя. Затем IMPORT заполняет созданное отношение информационным содержимым PFS файла и модифицирует каталог, если последний открыт.

Сразу после пересылки информации созданное отношение активизируется.

Таким образом, выполнение одной команды IMPORT помимо создания перечисленных выше объектов dBASE III Plus и импорта информации влечет выполнение еще следующей группы команд:

USE ⟨filename⟩
SET FORMAT TO ⟨.fmt filename⟩
CREATE VIEW ⟨.vue filename⟩ FROM ENVIRONMENT

dBASE III Plus, выдавая изнутри команду CREATE VIEW FROM ENVIRONMENT, создает внешнее представление, содержащее отношение и форматный файл. Все созданные объекты регистрируются в каталоге, если он открыт.

См. также APPEND FROM, CREATE VIEW, EXPORT, SET FORMAT, USE.

2.2.61. INDEX (dBASE III Plus и Clipper)

Команда создает индексный файл, в котором ключевое выражение, определенное на активном отношении, упорядочено по алфавиту, хронологически или численно. Индексный файл содержит ключевое значение и соответствующий ему номер кортежа для каждого кортежа в отношении.

Создаваемые командой индексные файлы имеют расширение .ndx для dBASE III Plus и .ntx — для Clipper. Индексные файлы dBASE III Plus и Clipper несовместимы, однако различие в расширениях дает возможность учесть этот факт в разрабатываемых приложениях, и если файл с соответствующим расширением не найден, то просто создать его командой INDEX.

Синтаксис. Для dBASE III Plus:

INDEX [ON ⟨key expression⟩ TO ⟨.ndx filename⟩ [UNIQUE]]

Для Clipper:

INDEX ON ⟨key expression⟩ TO ⟨.ntx filename⟩

Если диск не специфицирован, то индексный файл создается на активном диске в активном директории. Если расширение не задано, то для dBASE III Plus оно будет .ndx, а для Clipper — .ntx.

Использование. Ключ индексирования может быть выражением, включающим один или несколько атрибутов активного отношения. Логические и memo-атрибуты не могут включаться в ключ индексирования.

Индексирование всегда проводится по возрастанию значений ключа.

Кортежи отношения не переупорядочиваются физически (т. е. отношение не модифицируется) в ходе построения индекса.

Когда ключ индексирования включает много пунктов (атрибутов), следует расположить наиболее значащие вначале. Для объединения числовых атрибутов и атрибутов типа дата с символьными атрибутами в ключе индексирования нужно использовать функции STR ( ) и DTOC ( ) соответственно.

Максимальная длина выражения, составляющего ключ индексирования, равна 100 символам.

Если в dBASE III Plus вводится команда в сокращенной форме INDEX ⟨Enter⟩, то dBASE III Plus запросит ключ индексирования и имя индексного файла.

Если в dBASE III Plus SET SAFETY ON (значение по умолчанию), то dBASE III Plus запросит подтверждение в случае перезаписи индексного файла (при наличии индексного файла с тем же именем, что и создаваемый).

Команда INDEX дописывает новый индекс в каталог (как индекс активного отношения), если он открыт.

В Clipper размер ключа индексирования вычисляется путем расчета индексного выражения на пустом кортеже. Поэтому TRIM( ) от значения атрибута дает строку нулевой длины. Однако индекс может быть построен по усеченному атрибуту, если добавить в конец индексного выражения строку пробелов с длиной, равной длине значений усеченного атрибута. Например,

```
INDEX ON TRIM (Lname) +", "+TRIM (Fname);
      +SPACE (LEN (Lname + Fname)) TO Trim.ntx
```

Опции. Действие опции UNIQUE аналогично использованию команды SET UNIQUE ON перед выдачей команды INDEX или REINDEX. Оно состоит в том, что когда отношение содержит несколько кортежей с одинаковым значением ключа индексирования, то в индекс помещается номер только первого из них. Таким образом, индекс в некоторых случаях может быть эффективно сжат (если с его помощью предполагается выдавать информацию или осуществлять поиск кортежей только с уникальным значением ключа). При добавлении нового кортежа к отношению с индексом такого рода, значение ключа индексирования для которого уже есть в индексе (не уникально), модификация индекса не происходит. При удалении кортежа с неуникальным ключом модификация индекса также не происходит. В случае применения команды REINDEX для модификации индексного файла, созданного в режиме UNIQUE, модификация происходит с сохранением статуса UNIQUE независимо от того, установлен переключатель SET UNIQUE в положение ON или OFF.

См. также CLOSE, FIND, REINDEX, SEEK, SET INDEX, SET UNIQUE, USE.

2.2.62. INPUT (dBASE III Plus и Clipper)

Команда INPUT используется для организации ввода пользователя с клавиатуры. Ввод заканчивается нажатием Enter.

Синтаксис. INPUT [⟨prompt⟩] TO ⟨memvar⟩

Использование. ⟨prompt⟩ — символьное выражение, которое выдается пользователю в качестве приглашения на ввод информации.

Не требуется предварительно создавать переменную памяти, она создается в процессе ввода, и тип ее зависит от типа вводимой пользователем информации. Пользователь может ввести любое значащее выражение dBASE III Plus/Clipper. При вводе символьной строки ее необходимо заключать в ограничители (простые или двойные кавычки либо квадратные скобки). Если пользователь ввел ошибочное выражение, то возвращается сообщение о синтаксической ошибке. Если введено только Enter, то данные не запоминаются в переменную памяти.

Если ответ пользователя ожидается в виде символьной строки, то следует использовать команду ACCEPT, WAIT или @...GET с соответствующей PICTURE опцией. ACCEPT подразумевает, что любой пользовательский ввод — символьная строка, не нуждающаяся в ограничителях при вводе.

Если необходимо получить от пользователя дату, то ее ввод должен выглядеть примерно так:

CTOD ("12/04/87")

См. также ACCEPT, STORE, WAIT.

2.2.63. INSERT (только dBASE III Plus)

INSERT добавляет новый кортеж в отношение в месте, определяемом значением указателя текущей позиции. К сожалению, в Clipper эта команда отсутствует, хотя ее форма INSERT BLANK, вероятно, легко может быть скомпилирована.

Синтаксис. INSERT [BLANK] [BEFORE]

Использование. Если опция BLANK опущена, то dBASE III Plus добавляет в активное отношение кортеж после текущего (или перед ним, если указана опция BEFORE), а затем переходит в режим экранного редактирования, позволяя ввести информацию — значения атрибутов для добавленного кортежа.

Если отношение используется в режиме индексного доступа, то кортеж добавляется в конец отношения.

Значения в атрибут типа memo вводятся путем позиционирования курсора на слово memo и нажатия комбинации клавиш Ctrl — PgDn. При этом подключится инстолированный для обработки memo-атрибутов текстовый процессор либо внутренний текстовый редактор MODIFY COMMAND, если не инстолирован другой.

Клавиши направлений перемещают курсор по кортежу. Клавиша Esc прерывает процесс без добавления кортежа к отношению. Комбинация клавиш Ctrl — End заканчивает редактирование и добавляет кортеж к отношению.

Если указана опция BLANK, то новый кортеж добавляется, но переход в экранный режим не происходит. Данные могут быть внесены позднее с использованием команд редактирования (EDIT, CHANGE, BROWSE и др.) или без них (REPLACE, @. . .GET...).

Для автоматического переноса содержимого текущего кортежа во вставляемый следует установить SET CARRY ON.

См. также @, APPEND, CHANGE, EDIT, MODIFY COMMAND, SET CARRY, SET FORMAT.

2.2.64. JOIN (dBASE III Plus и Clipper)

JOIN создает новое отношение путем слияния определенных кортежей (определенных из атрибутов) из двух открытых отношений. В терминах алгебры отношений эта команда реализует теоретико-множественное произведение с последующей фильтрацией.

Синтаксис. JOIN WITH ⟨alias⟩ TO ⟨filename⟩
FOR ⟨condition⟩ [FIELDS ⟨field list⟩]

Использование. Когда активное отношение комбинируется с открытым отношением из невыбранной (неактивной) рабочей области, второе отношение идентифицируется своим альтернативным именем (alias — имя соответствующей рабочей области), которое может совпадать с основным.

Список атрибутов может включать атрибуты из обоих отношений. Если список атрибутов опущен, берутся все атрибуты первого (активного) отношения и затем дополняются атрибутами из второго (атрибуты с совпадающими именами из второго отношения не берутся) либо до исчерпания списка, либо (для dBASE III Plus) до превышения лимита в 128 атрибутов.

JOIN обновляет каталог, если последний открыт.

Механизм работы команды следующий. Указатель текущего кортежа устанавливается на первый кортеж в активном отношении.

Поочередно каждый кортеж второго отношения совместно с первым кортежем активного отношения проверяется на выполнение (истинность) FOR-условия команды. Если условие истинно, то новый кортеж, образованный из значений атрибутов первого кортежа активного отношения и очередного кортежа второго отношения, добавляется в результирующее (порождаемое) отношение. Затем указатель в активном отношении перемещается на второй кортеж, весь процесс повторяется и так далее до исчерпания кортежей в первом (активном) отношении.

Следует быть внимательным при задании условия FOR, так как эта команда при некорректном употреблении может породить громадное по объему отношение. Так, например, если в активном отношении 1000 кортежей и во втором — также 1000, то при пустом FOR-условии результирующее отношение будет содержать 1000 * * 1000 = 1000000 кортежей.

Нельзя использовать буквы от A до M в качестве имен отношений (файлов), так как они зарезервированы под альтернативные имена (имена рабочих областей).

Если отношения имеют совпадающие имена атрибутов и требуется специфицировать в списке атрибутов атрибут из второго отношения, нужно использовать конструкцию «альтернативное имя» — ⟩ «имя атрибута». Для выбора атрибута активного отношения следует использовать просто его имя.

См. также INSERT, SET FIELDS, SET RELATION.

2.2.65. KEYBOARD (только Clipper)

Эта команда заполняет входной буфер (буфер ввода) заданной последовательностью кодов, имитируя, таким образом, ввод пользователя с клавиатуры. Каждое выполнение этой команды очищает имевшийся буфер ввода. Поэтому последовательность команд KEYBOARD не может быть использована для организации очереди последовательностей кодов. Команда KEYBOARD "" может быть использована для очистки буфера ввода.

Синтаксис. KEYBOARD ⟨expC⟩

2.2.66. LABEL (dBASE III Plus и Clipper)

Команда LABEL реализует «рассылку писем» — заполнение подготовленных заранее этикеток .lbl файлов (см. п. 1.5 и описание команды CREATE/MODIFY LABEL) информацией из активного отношения.

Синтаксис. LABEL FORM ⟨label filename⟩/? [⟨scope⟩]
[SAMPLE] [WHILE ⟨condition⟩]
[FOR ⟨condition⟩] [TO PRINT]
[TO FILE ⟨filename⟩]

Использование. Опция ? применима только для dBASE III Plus. В случае открытого каталога имя .lbl файла можно выбрать из списка всех .lbl файлов, зарегистрированных в данном каталоге.

Имя файла должно включать дисковый спецификатор, если файл не на активном диске. Если FORM расширение опущено, то подразумевается— .lbl. Если TO FILE расширение опущено, то подразумевается .txt. Если опущены ⟨scope⟩, FOR и WHILE, то этикетки заполняются по всем кортежам отношения.

Опция SAMPLE используется, если требуется получить образец заполненной этикетки; этот процесс может повторяться, если отвечать «Y» на вопрос «Do you want more samples?» («Хотите еще образцы?»).

Опция TO FILE позволяет переслать заполненные этикетки в стандартный текстовый ASCII файл.

Если необходимо направить вывод на печать, нужно убедиться, что принтер включен, иначе компьютер может «зависнуть».

Важное замечание. Для Clipper содержимое поля в этикетке должно быть значащим выражением. dBASE III Plus позволяет указывать список символьных атрибутов через запятую, в то время как в Clipper это недопустимо и приводит к неполной совместимости .lbl файлов dBASE III Plus и Clipper.

Описание содержимого поля в этикетке:

в dBASE III Plus:	в Clipper:
Fname, Lname	TRIM (Fname) + " " + Lname

См. также CREATE/MODIFY LABEL.

2.2.67. LIST (dBASE III Plus и Clipper)

Команда LIST используется для вывода содержимого отношения базы данных. Эта команда удобна при выводе на печать.

Синтаксис. Для dBASE III Plus:

LIST [⟨scope⟩] [⟨expression list⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]
[OFF] [TO PRINT]

Для Clipper:

LIST [⟨scope⟩] ⟨expression list⟩
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]
[OFF] [TO PRINT/TO FILE ⟨filename⟩]

Для Clipper обязательно указание списка выводимых атрибутов/выражений. Clipper в отличие от dBASE III Plus не выводит имена атрибутов как заголовок вывода. dBASE III Plus предваряет вывод информации по команде LIST таким заголовком.

Если опция ⟨scope⟩ и FOR, WHILE условия опущены, то выводится информация (которая специфицирована в ⟨expression list⟩) по всем кортежам отношения. Для приостанова выдачи следует нажать комбинацию клавиш Ctrl — S, для продолжения — любую клавишу.

Если ⟨expression list⟩ опущен (что возможно только в случае dBASE III Plus), то выдаются значения всех атрибутов отношения. Если параметр OFF не задан, то выводятся номера кортежей (перед каждым кортежем).

Использование. Команда LIST аналогична команде DISPLAY, за исключением того, что по умолчанию выдаются все кортежи отношения, и пауза через каждые 20 кортежей не делается. Это удобно при выводе на печать.

См. также DISPLAY, SET MEMOWIDTH, SET HEADING.

2.2.68. LIST HISTORY (только dBASE III Plus)

Аналог DISPLAY HISTORY, но осуществляет выдачу без пауз по заполнению экрана, что удобно при выдаче на печать.

Синтаксис. LIST HISTORY [LAST ⟨expN⟩] [TO PRINT]

2.2.69. LIST MEMORY (только dBASE III Plus)

Аналог DISPLAY MEMORY, но осуществляет выдачу без пауз по заполнению экрана, что удобно при выдаче на печать.

Синтаксис. LIST MEMORY [TO PRINT]

2.2.70. LIST STATUS (только dBASE III Plus)

Аналог DISPLAY STATUS, но осуществляет выдачу без пауз по заполнению экрана, что удобно при выдаче на печать.

Синтаксис. LIST STATUS [TO PRINT]

2.2.71. LIST STRUCTURE (только dBASE III Plus)

Аналог DISPLAY STRUCTURE, но осуществляет выдачу без пауз по заполнению экрана, что удобно при выдаче на печать.

Синтаксис. LIST STRUCTURE [TO PRINT]

См. также DISPLAY STRUCTURE, SET FIELDS.

2.2.72. LOAD (только dBASE III Plus)

Команда LOAD дает возможность загрузить некоторую поддержку (процедуру или набор процедур), написанную на Ассемблере (или на другом языке: СИ, PASCAL и др.) и оформленную в виде .bin файла (это можно сделать с помощью утилиты EXE2BIN.EXE). Команда LOAD работает в паре с командой CALL.

Синтаксис. LOAD ⟨binary filename⟩ [⟨extension⟩]

Использование. LOAD загружает бинарный файл в память, откуда он может быть исполнен при помощи команды CALL.

Замечания по программированию. dBASE III Plus считает каждый загруженный по LOAD файл подпрограммой, а не внешней программой.

Могут быть одновременно загружены максимум пять файлов. Каждый может быть до 32000 байт длиной и должен быть в бинарном формате. Каждый загружаемый файл должен иметь уникальное имя. Расширением по умолчанию является .bin. Если загружается файл с тем же именем, что и ранее загруженный, но с другим расширением, то загружаемый файл замещает в памяти предыдущий. Для просмотра имен загруженных модулей (файлов) используется команда DISPLAY/LIST STATUS. dBASE III Plus не проверяет целостность загружаемого файла. Следует убедиться, что он в бинарной форме и работает.

При разработке модулей поддержки необходимо соблюдать следующие правила:

первая исполняемая инструкция в модуле должна идти со смещением 0 (т. е. сразу);

программа не должна размещать или использовать переменные памяти, находящиеся над ней и «сверх» нее (т. е. начинающиеся в ней и заканчивающиеся выше), так как LOAD использует размер файла, чтобы определить количество памяти, требуемое для размещения модуля;

программа не должна удлинять или укорачивать переменные памяти dBASE III Plus, передаваемые ей как параметры по CALL... ...WITH... в DS : BX;

перед возвратом управления в dBASE III Plus программа должна восстановить значения CS- и SS-регистров;

управление в dBASE III Plus следует возвращать с помощью FAR RET, а не по EXIT (такие программы следует исполнять командой RUN).

Ассемблерную программу для процессоров 8086/8088 следует подготавливать на Microsoft MASM, используя следующую последовательность команд:

```
MASM ⟨filename⟩ ⟨filename⟩ NULL NULL
LINK ⟨filename⟩ NULL NULL
EXE2BIN ⟨filename⟩
```

Параметр MAXMEM в файле Config. db должен быть установлен в значение, большее 256K, так как загружаемые по LOAD модули не перезагружаются после исполнения внешней программы по RUN. Нужно установить MAXMEM в значение, превосходящее 256K на величину, равную размеру загружаемых модулей.

В качестве решения проблемы процедурной поддержки в dBASE III Plus может быть предложено следующее:

сама поддержка, содержащая пакет различных процедур, написанных на любом (возможно, на различных) из языков программирования, объединяется в загрузочном (.exe) файле, который в качестве главной программы содержит процедуру инстоляции обработчика некоторого (например, 65-го) прерывания и собственно обработчика этого прерывания;

обработчик прерывания является диспетчером, его функция — передача управления одной из процедур пакета в соответствии со значением переданных ему параметров;

главная программа по окончании инстоляции обработчика прерывания заканчивает свою работу, выдав команду DOS — «закончить и остаться резидентно в памяти»;

приготовляемый для dBASE III Plus. .bin файл содержит одну команду — «прерывание» (INT 41h, например);

из dBASE III Plus выполняется вначале команда RUN ⟨файл exe⟩, затем LOAD ⟨файл — bin⟩, а затем в требуемых местах — CALL ⟨файл bin⟩ WITH ⟨параметры⟩ и т. д.

Такой метод используют некоторые фирмы — разработчики средств расширения возможностей dBASE III Plus.

См. также CALL, RELEASE.

2.2.73. LOCATE (dBASE III Plus и Clipper)

LOCATE производит последовательный поиск в активном отношении кортежа, который удовлетворяет условию поиска.

Синтаксис. LOCATE [⟨scope⟩] [FOR ⟨condition⟩]
[WHILE ⟨condition⟩]

Если иное не задано в опциях, LOCATE начинает поиск кортежа, удовлетворяющего условию, с первого кортежа отношения.

Использование. Опция NEXT ⟨n⟩ ограничивает зону поиска заданным числом кортежей (⟨n⟩) и начинает поиск с текущего кортежа.

Для нахождения следующих кортежей, удовлетворяющих этому условию поиска, следует использовать команду CONTINUE.

Если кортеж, удовлетворяющий условию, найден, то указатель текущего кортежа указывает на него, а функция FOUND ( ) принимает значение .T. (истина). Если же такой кортеж не найден, то указатель текущего кортежа указывает на конец отношения (EOF ( ) = .T.) либо на конец зоны поиска, если таковая была специфицирована. dBASE III Plus высвечивает сообщение «End of locate scope», а функция FOUND ( ) возвращает .F.

Специальные случаи. Если не задана опция NEXT ⟨n⟩, или REST, или WHILE, LOCATE начинает поиск с начала отношения. Если последовательно дается несколько LOCATE команд, то каждая последующая уничтожает установки предыдущей (имеется в виду продолжение поиска с помощью команды CONTINUE).

LOCATE и CONTINUE характерны для рабочей области, которая является активной в момент их выдачи. Можно определить различные LOCATE/CONTINUE для каждой рабочей области. Если в ходе работы нужно покинуть рабочую область, то определенный для нее LOCATE/CONTINUE все равно остается и, вернувшись, можно продолжить поиск очередного кортежа, удовлетворяющего условию поиска, командой CONTINUE.

См. также CONTINUE, FOUND ( ).

2.2.74. LOOP (dBASE III Plus и Clipper)

LOOP возвращает управление в начало DO WHILE. . .ENDDO-структуры, предотвращая выполнение оставшихся в теле цикла команд.

Синтаксис. LOOP

Использование. Если необходимо, следует использовать LOOP команду внутри вложенных в DO WHILE. . .ENDDO IF. . .ENDIF и DO CASE. . .ENDCASE конструкциях.

См. также DO, DO CASE. . .ENDCASE, DO WHILE. . .ENDDO, EXIT, IF. . .ENDIF

2.2.75. MENU TO (только Clipper)

Команда MENU TO ⟨memvar⟩ активизирует меню, описанное предварительно командами @. . .PROMPT, выделяет подсветкой первую позицию (prompt), позволяя затем передвигаться курсором от позиции к позиции меню и поместить в указанную переменную (memvar) порядковый номер выбранной позиции. Эта команда похожа на команду READ.

Синтаксис. MENU TO ⟨memvar⟩

Использование. Нажатие клавиш Return, PgUp, PgDn вызывает выбор текущей позиции меню и запоминание ее относительного номера в переменной ⟨memvar⟩. Нажатие клавиши Esc возвратит нуль в переменной ⟨memvar⟩ — выбор не был сделан. Выбор позиции может быть также произведен по первому символу (нажатием соответствующей алфавитно-цифровой клавиши), однако текущая версия Clipper распознает лишь коды 20h—7Fh (это может быть устранено соответствующей корректировкой библиотеки Clipper .lib). Режим «Помощь» может быть активизирован путем нажатия клавиши F1 (в случае наличия HELP. PRG), и в качестве параметра input_var процедура help получит имя переменной ⟨memvar⟩, созданной командой MENU TO. Меню могут быть вложенными (т. е. внутри help) без очистки GET (CLEARing GETS) в отличие от GET/READ механизма. Однако если одна и та же переменная используется для вложенных меню, то она сохраняет предыдущее

значение. Поэтому рекомендуется использовать отдельные переменные для каждого меню.

См. также @...PROMPT, HELP, SET MESSAGE TO.

2.2.76. MODIFY COMMAND (только dBASE III Plus)

MODIFY COMMAND — это текстовый редактор dBASE III Plus. Главной его задачей является создание и редактирование командных и форматных файлов, но он может быть использован и для обычных ASCII-текстовых файлов.

Синтаксис. MODIFY COMMAND ⟨filename⟩
или альтернативный синтаксис:
MODIFY FILE ⟨filename with extention⟩

Если в имени файла код диска не указан, то подразумевается активный диск. Если расширение умалчивается, то подразумевается .prg.

Использование. По MODIFY COMMAND III Plus пытается найти указанный файл. Если он есть, то система приступает к его редактированию, иначе — создает его. Каждый раз по окончании редактирования старая версия файла сохраняется с расширением .bak.

В качестве текстового редактора можно подключить любой текстовый процессор. Если используется встроенный редактор dBASE III Plus, то следует иметь в виду, что он «прорезает» старший (8-й) бит в кодах (устанавливает его в «0»). Максимальный размер файла, который он может обработать — 5000 байт. При вводе строки длиннее 66 символов он продолжает ее на следующей строке, автоматически осуществляя перенос.

Для получения твердой копии текста следует использовать команду TYPE ⟨filename⟩ TO PRINT.

Значения memo атрибутов не могут редактироваться с помощью этой команды (можно использовать EDIT или CHANGE, которые вызывают в общем случае другой текстовый процессор, если это предусмотрено в конфигурации).

Во встроенном текстовом редакторе dBASE III Plus используются следующие управляющие клавиши, которые выполняют следующие функции:

клавиши направлений	— передвигают курсор на одну позицию в соответствующем направлении
Backspace, Rub	— удаляет символ слева от курсора
F1, Ctrl — \	— включение/выключение меню управления курсором
Ctrl — A, Home	— сдвигает курсор в начало текущего или предыдущего слова
Ctrl — B, Ctrl — «стрелка вправо»	— курсор в конец строки
Ctrl — C	— скролинг вниз — следующие 18 строк текста (следующая страница)
Ctrl — D	— эквивалентно «стрелка вправо»
Ctrl — E	— эквивалентно «стрелка вверх»
Ctrl — F, End	— передвигает курсор в начало следующего слова
Ctrl — G, Del	— удаляет символ, на который указывает курсор
Ctrl — KB	— переформатирует параграф
Ctrl — KF	— находит первое вхождение заданной строки в тексте

Ctrl — KL	— находит следующее вхождение строки, найденной по команде Ctrl — KF
Ctrl — KR	— считывает другой файл в редактируемый файл начиная с текущего положения курсора
Ctrl — KW	— записывает редактируемый файл в другой файл
Ctrl — M, Enter	— курсор помещается в начало следующей строки; если INSERT ON (режим вставки), то добавляет пустую строку
Ctrl — N	— добавляет пустую строку после курсора
Ctrl — Q, Esc	— выход из редактирования без сохранения сделанных изменений
Ctrl — R, PgUp	— скролинг вверх — предыдущие 18 строк текста (предыдущая страница)
Ctrl — S	— эквивалентно «стрелка влево»
Ctrl — T	— удаляет все символы начиная от позиции курсора и до начала следующего слова
Ctrl — V, Ins	— переключатель режима вставки: вкл/выкл
Ctrl — KW, Ctrl-End	— выход с сохранением
Ctrl — X	— эквивалентно «стрелка вниз»
Ctrl — Y	— удаление текущей строки
Ctrl — Z, Ctrl — «стрелка влево»	— курсор помещается в начало строки

2.2.77. MODIFY LABEL (только dBASE III Plus)

См. CREATE/MODIFY LABEL.

2.2.78. MODIFY QUERY (только dBASE III Plus)

См. CREATE/MODIFY QUERY.

2.2.79. MODIFY REPORT (только dBASE III Plus)

См. CREATE/MODIFY REPORT.

2.2.80. MODIFY SCREEN (только dBASE III Plus)

См. CREATE/MODIFY SCREEN.

2.2.81. MODIFY STRUCTURE (только dBASE III Plus)

Эта команда позволяет изменить структуру существующего отношения.

Синтаксис. MODIFY STRUCTURE ⟨filename⟩

Если расширение не указано, то dBASE III Plus подразумевает .dbf.

MODIFY STRUCTURE всегда вначале сохраняет старый вариант отношения в файле .bak. После завершения модификации

структуры dBASE III Plus возвращает содержимое из .bak файла в измененное отношение. При изменении атрибутов типа memo в отношении dBASE III Plus создает промежуточную копию .dbt файла с расширением .$$$.

Предупреждение. Хотя можно менять и имена атрибутов, и их длину, однако если делать это в одном сеансе MODIFY STRUCTURE, то dBASE III Plus не сможет добавить значения корректно (возвратить в отношение с измененной структурой) и все атрибуты с измененными именами получат пустые значения. Следует сделать необходимые изменения в именах атрибутов в одном сеансе MODIFY STRUCTURE, а затем изменить длины, вызвав команду вторично. Удаление атрибута из отношения эквивалентно замене его длины нулевой. Поэтому нужно делать удаления атрибутов в том сеансе, где изменяются длины (а не имена). Значения атрибутов в модифицированном отношении также заполняются пробелами при изменении типа атрибута на нечисловой или добавлении нового атрибута.

Структура отношения определяется следующей информацией по каждому атрибуту: имя; тип; длина; число позиций после десятичной точки (только для числового типа в случае, если значения этого атрибута не целые числа).

При указании/корректировке числа позиций после десятичной точки следует помнить, что десятичная точка и знак минус входят в общую длину атрибута.

Информация о текущем атрибуте высвечивается инверсным или другим цветом.

Подробные инструкции и сообщения об ошибках высвечиваются внизу экрана.

Нажатие F1 включает/выключает меню управления курсором. Нажатие Ctrl — Home включает/выключает меню опций. Клавиши направлений перемещают курсор по опциям, а Enter осуществляет выбор. Когда количество атрибутов превосходит размеры экрана, Ctrl — «стрелка вправо» и Ctrl — «стрелка влево» используются для скролинга. Ввод типа атрибута осуществляется по первой букве (C, N, L, D или M) или последовательными нажатиями пробела до появления требуемого, после чего нажимается Enter для его фиксации. Курсор не может передвигаться по записи, описывающей недоопределенный атрибут, однако он может перемещаться от атрибута к атрибуту.

Можно включать новые атрибуты. Нажатие Ctrl — N приводит к добавлению атрибута перед текущим.

Для выхода нужно нажать Ctrl — End или Enter, когда определяется новый атрибут. Esc — выход без сохранения сделанных изменений.

2.2.82. MODIFY VIEW (только dBASE III Plus)

См. CREATE/MODIFY VIEW.

2.2.83. NOTE/* (dBASE III Plus и Clipper)

Команда NOTE или звездочка (*) в начале строки превращают эту строку в строку комментария, исключая ее из обработки интерпретатором или компилятором. Используется для внесения комментариев в текст программы.

Синтаксис. NOTE/* ⟨text⟩

Использование. Если строка комментариев оканчивается точкой с запятой (;), то следующая строка также считается комментарием. Для включения комментария в строку какой-либо команды (после нее) следует использовать &&. Команды завершения структурных групп ENDDO, ENDIF, ENDCASE позволяют включать комментарии непосредственно после команды даже без &&.

2.2.84. ON (только dBASE III Plus)

ON прерывает исполнение программы, если специфицированное условие наступает, и инициирует обработку возникшей ситуации.

Синтаксис. ON ERROR/ESCAPE/KEY ⟨command⟩

Использование. Каждый ON остается активным до его удаления (деактивации). Для удаления ON выдается соответствующая команда ON ERROR/ESCAPE/KEY без ⟨command⟩, т. е. с пустыми действиями по обработке данной ситуации.

ON ESCAPE не будет работать, если реакция на Esc — SET ESCAPE — OFF.

Старшинство ON единиц в dBASE III Plus следующее:

ON ERROR — dBASE III Plus обнаружил ошибку в команде в ходе интерпретации;

ON ESCAPE — пользователь нажал Esc;

ON KEY — пользователь нажал любую клавишу.

Если, например, ON KEY и ON ESCAPE активны в одно и то же время, то нажатие Esc приводит к обработке ON ESCAPE ситуации (выполнению связанной команды), а не ON KEY. А если ON ESCAPE не определено, а заданы ON KEY и SET ESCAPE OFF, то активизируется обработка ON KEY (по нажатию Esc).

Если ON KEY задано (активно) и нажата любая клавиша (кроме Esc), то ON KEY обработка начнется после завершения выполнения текущей команды. Например, нажатие клавиши во время процесса индексирования отношения не прервет процесс индексирования.

Однако важно отметить, что, поскольку ввод буферизируется, ON KEY ситуация может обрабатываться несколько раз повторно.

Корректное использование ON KEY требует, чтобы код нажатой клавиши, запомненный в буфере ввода, извлекался из буфера с помощью команд, таких, как INKEY ( ) или READ. Если используется ON KEY WAIT TO ⟨memvar⟩, то код нажатой клавиши присваивается переменной памяти ⟨memvar⟩, а ожидание не выполняется.

ON ERROR не распознает ошибки уровня операционной системы, такие, как отказ диска или принтера. Эта ситуация возникает лишь при ошибках уровня dBASE III Plus— синтаксических или по ненахождению файлов.

См. также SET ESCAPE, INKEY ( ), READKEY ( ).

2.2.85. PACK (dBASE III Plus и Clipper)

PACK удаляет кортежи, помеченные признаком удаления, из активного отношения.

Синтаксис. PACK

Использование. Все открытые индексы автоматически подвергаются реорганизации (REINDEX).

После упаковки отношение сжимается и внешняя память, занимавшаяся удаленными кортежами, возвращается операционной системе (но не сразу, а по закрытии отношения). Сведения о числе кортежей отношения также могут обновиться лишь после его закрытия.

См. также DELETE, RECALL, REINDEX, ZAP.

2.2.86. PARAMETERS (dBASE III Plus и Clipper)

Команда PARAMETERS устанавливает соответствие между локальными переменными процедуры или функции (в Clipper) и значениями, переданными ей вызывающей программой. Эта команда является «принимающей» для переменных, переданных с помощью: DO ⟨procname/filename⟩ WITH ⟨parameter list⟩ — dBASE III Plus / Clipper;

CALL ⟨procname/filename⟩ WITH ⟨parameter list⟩ — dBASE III Plus / Clipper;

⟨functionname⟩ ((param 1), . . . ,⟨param n⟩) — Clipper.

Синтаксис. PARAMETERS ⟨parameter list⟩

Использование. PARAMETERS должна быть первой исполнимой командой в процедуре — функции. Для каждой процедуры — функции может быть задана только одна команда PARAMETERS. Передаваться может любое допустимое выражение dBASE III Plus / Clipper. Заявленное количество параметров должно совпадать с количеством переданных значений. Объявленные в списке PARAMETERS переменные считаются локальными и уничтожаются при возврате управления в вызывающую программу. Переменные памяти передаются по ссылке, что позволяет менять значение переданной переменной из вызываемой программы. Выражения передаются по значению. Это два основных правила передачи параметров.

При передаче массивов (Clipper) и переменных памяти в качестве параметров функций должны выполняться следующие правила.

1. Параметры всегда передаются по значению (Clipper).
2. По ссылке передаются только параметры-идентификаторы. Если передается выражение, то параметр передается по значению. Элементы массива могут быть переданы только по значению.

Параметр, передаваемый по ссылке, может быть изменен вызываемой программой, а передаваемый по значению — нет, поскольку он представляет собой копию оригинала. Изменение этой копии не изменяет оригинал.

Важное замечание. Clipper позволяет получить переданные прикладной программе параметры при ее вызове из операционной системы. Эти параметры представляются в виде одного параметра — символьной строки, содержащей в себе все символы командной строки DOS начиная с первого отличного от пробела символа после имени программы (исполнимого файла).

См. также DO, CALL, FUNCTION, LOAD, PRIVATE, PROCEDURE, PUBLIC, SET PROCEDURE.

2.2.87. PRIVATE (dBASE III Plus и Clipper)

PRIVATE предохраняет переменные памяти в программе — процедуре — функции от совмещения в памяти с переменными вызывающей программы или где-либо ранее объявленными глобальными переменными (PUBLIC) с такими же именами. По окончании

содержащей их программы — процедуры — функции локальные (PRIVATE) переменные уничтожаются.

Синтаксис. Для dBASE III Plus:

PRIVATE ⟨memvar list⟩

PRIVATE ALL [LIKE/EXCEPT ⟨skeleton⟩]

Для Clipper:

PRIVATE ⟨memvar list⟩

См. также DO, CALL, FUNCTION, LOAD, PARAMETERS, PROCEDURE, PUBLIC, SET PROCEDURE.

2.2.88. PROCEDURE (dBASE III Plus и Clipper)

Эта команда индицирует начало процедуры. Для dBASE III Plus — это начало каждой процедуры в командном файле. Для Clipper понятия процедуры и командного файла эквивалентны и соответственно эквивалентны команды DO и CALL. В Clipper в программном файле могут быть описаны некоторые процедуры. Они могут быть помещены либо совместно с главной программой, либо каждая в своем файле — важно лишь собрать все необходимые модули на объектном уровне.

В Clipper команда SET PROCEDURE TO открывает файл, содержащий процедуры, во время компиляции. Имя процедуры или функции не должно совпадать с именем файла, в котором они содержатся!

Синтаксис. PROCEDURE ⟨procedure name⟩

Использование. Когда dBASE III Plus встречает команду PROCEDURE в командном файле, который не является частью заявленного ранее командой SET PROCEDURE TO файла, система немедленно выполняет RETURN.

Имена процедур могут иметь длину до восьми символов. Они могут включать буквы, цифры и знак подчеркивания «—» и должны начинаться с буквы.

См. также DO, CALL, FUNCTION, LOAD, PARAMETERS, PRIVATE, PUBLIC, SET PROCEDURE.

2.2.89. PUBLIC (dBASE III Plus и Clipper)

PUBLIC объявляет переменные памяти глобальными, т. е. делает их видимыми из подпрограмм. В отличие от других переменных памяти они не уничтожаются по завершении программы.

Синтаксис. Для dBASE III Plus:

PUBLIC ⟨memory variable list⟩

Для Clipper:

PUBLIC ⟨memory variable list⟩ / Clipper

Использование. Переменные должны быть объявлены PUBLIC перед присвоением им значений. Значения глобальных переменных могут быть изменены любой подпрограммой, однако они могут быть сделаны временно невидимыми путем объявления в подпрограмме/процедуре/функции (для Clipper) локальных переменных (PRIVATE) с теми же именами.

В Clipper допустима альтернативная форма синтаксиса этой команды, позволяющая объявить глобальную переменную Clipper. Эта важная форма команды служит для обеспечения совместимости Clipper и dBASE III Plus. Она позволяет включать расширения Clipper (фрагменты программы, использующие команды или допол-

нительные возможности Clipper, не реализованные в dBASE III Plus) в существующие на языке dBASE III Plus программы, создавая программы, которые могут выполняться как в интерпретирующей среде dBASE III Plus, так и быть скомпилированы и отлинкованы в среде Clipper.

dBASE III Plus все логические переменные инициирует значением .F. (ложь). Clipper делает то же самое для всех переменных, за исключением специальной переменной Clipper, которой он присваивает значение .T. (истина) при инициализации. Будучи объявлена глобально, эта переменная служит ключом условной компиляции — интерпретации.

См. также DO, CALL, FUNCTION, LOAD, PARAMETERS, PROCEDURE, PRIVATE, SET PROCEDURE.

2.2.90. QUIT (dBASE III Plus и Clipper)

Эта команда выхода из среды dBASE III Plus или из Clipper программы. QUIT закрывает все файлы, прерывает сессию dBASE III Plus и возвращает управление операционной системе.

Синтаксис. QUIT

Использование. Это «гигиеничный» метод выхода из dBASE III Plus. Выключение компьютера без QUIT может повредить открытые файлы и привести к потере информации.

2.2.91. READ (dBASE III Plus и Clipper)

READ активизирует все «подвешенные» (ожидающие) GET, установленные командами @. . .SAY. . .GET. . . , выданными после последней команды CLEAR или CLEAR ALL, или CLEAR GETS, или READ. Команда READ осуществляет ввод — редактирование в режиме экранного редактирования и наиболее часто используется в программе с целью организации ввода — редактирования информации в экранном режиме.

Синтаксис. Для dBASE III Plus:

READ [SAVE]

Для Clipper:

READ

Использование. READ использует стандартные клавиши управления курсором в режиме экранного редактирования.

READ — это единственный путь для редактирования переменных памяти (в Clipper есть еще функция MEMOEDIT ()).

READ очищает все GET после конца редактирования.

Опции. READ SAVE для dBASE III Plus не очищает GET по окончании редактирования, поэтому при вызове READ в следующий раз активизируется тот же набор GET. При использовании READ SAVE не следует забывать о вызове CLEAR GETS до достижения 128 «подвешенных» GET (или количества, специфицированного в конфигурации dBASE III Plus).

Замечания по программированию. При использовании многостраничного формата (.fmt), в котором @. . .SAY. . .GET. . . продолжаются на 2—32 экранах, нужно включать READ в тех местах, где необходимо поставить перевод страницы. Тогда клавиши PgUp и PgDn будут переключать страницы. Мультистраничные .fmt файлы работают только в случае, когда форматный файл открыт по SET FORMAT TO.

См. также CLEAR, CLEAR GETS, CLEAR MEMORY, CREATE/MODIFY SCREEN, MEMOEDIT ( ), MENU TO, SET DEVICE TO, SET FORMAT TO.

2.2.92. RECALL (dBASE III Plus и Clipper)

RECALL восстанавливает (отменяет признак удаления) те кортежи активного отношения, которые помечены к удалению и удовлетворяют условию восстановления, заданному в команде RECALL.

Синтаксис. RECALL [⟨scope⟩] [WHILE ⟨condition⟩]
[FOR ⟨condition⟩]

Если условие восстановления не специфицировано (т. е. просто RECALL), то восстанавливается лишь текущий кортеж. RECALL не восстанавливает кортежи, необратимо удаленные ранее командами PACK или ZAP. RECALL ALL не имеет эффекта в случае, если SET DELETED ON. Необходимо специфицировать, какие кортежи восстанавливать.

См. также DELETE, DELETED( ), PACK, SET DELETED ON, ZAP.

2.2.93. REINDEX (dBASE III Plus и Clipper)

REINDEX перестраивает (обновляет) все активные индексные файлы (.ndx — для dBASE III Plus и .ntx — для Clipper).

Синтаксис. REINDEX

Использование. При обновлении индекса, созданного с SET UNIQUE ON или с опцией UNIQUE (что равносильно), новый индекс тоже получает статус UNIQUE (см. INDEX) независимо от того, включен или выключен режим UNIQUE.

Команда REINDEX может быть эффективно использована в случае, когда требуется интенсивная модификация активного отношения, затрагивающая ключи индексации. Такая модификация при использовании индексного механизма доступа проходит очень медленно, так как требует обновления индексов после каждого акта модификации. Эта проблема может быть решена следующим образом: отношение открывается без активизации индексов; интенсивно модифицируется; активизируются индексы и обновляются командой REINDEX.

См. также INDEX, PACK, SET INDEX TO, SET UNIQUE, USE.

2.2.94. RELEASE (dBASE III Plus и Clipper)

Удаляет переменные памяти и освобождает память для последующего использования.

Синтаксис. Для dBASE III Plus:

RELEASE [⟨memvar list⟩]
[ALL [LIKE EXCEPT ⟨skeleton⟩]]
[MODULE ⟨module name⟩]

Для Clipper:

RELEASE [⟨memvar list⟩]
[ALL [LIKE/EXCEPT ⟨skeleton⟩]]

Использование. В шаблоне (⟨skeleton⟩)? обозначает одиночный произвольный символ, а * — один или более произвольных симво-

лов. Для dBASE III Plus в интерактивном режиме RELEASE ALL уничтожает все переменные памяти. В программе эта команда удаляет все переменные, созданные текущей подпрограммой. RELEASE MODULE удаляет загруженный командой LOAD модуль из памяти. Это позволяет использовать более пяти модулей поддержки в приложениях (см. LOAD) или просто освобождать память для другого использования.

См. также CALL, CLEAR MEMORY, LOAD, RESTORE, RETURN, SAVE, STORE.

2.2.95. RENAME (dBASE III Plus и Clipper)

RENAME изменяет имя дискового файла.

Синтаксис. RENAME ⟨old file name⟩ TO ⟨new file name⟩

Старое и новое имена должны включать расширения, а если файлы не на текущем диске, то имя дисковода.

Использование. Новый файл не может иметь имя уже существующего на диске файла. Если переименовывается файл, содержащий отношение, которое включает memo-атрибуты, то соответствующий .dbt файл должен быть переименован отдельно.

Открытый файл не может быть переименован.

Не следует использовать буквы от A до M как имена файлов, содержащих отношения, так как эти буквы зарезервированы под имена рабочих областей. Например, AA — значащее имя, а A — нет.

См. также CLOSE, USE.

2.2.96. REPLACE (dBASE III Plus и Clipper)

REPLACE изменяет значения специфицированных атрибутов в удовлетворяющих условию поиска кортежах активного отношения.

Синтаксис. REPLACE [⟨scope⟩] ⟨field⟩ WITH ⟨exp⟩
[, ⟨field⟩ WITH ⟨exp⟩ . . .]
[FOR ⟨condition⟩] [WHILE ⟨condition⟩]

Если ни ⟨scope⟩, ни условие поиска не заданы, то значения заменяются в указанных атрибутах только текущего кортежа.

Использование. Значение заменяющего выражения должно совпадать с типом атрибута, значение которого заменяется.

Для числовых атрибутов заменяющее число (численное выражение) не должно превышать размерность атрибута. В противном случае (переполнение) запишутся звездочки (*) и некоторая информация может быть утеряна.

Замены в индексированном отношении (в отношении с активизированным механизмом индексного доступа) также обновляют индексный файл. Когда замена произведена, кортеж переходит на новую позицию по индексному файлу. Поэтому не следует использовать опции ⟨scope⟩ WHILE, FOR, когда выполняются замены в индексированном отношении.

Указатель текущего кортежа. Команда REPLACE не меняет позицию указателя текущего кортежа. Если он находится в конце отношения (EOF ( )= .T.), то замены не делаются. Следует убедиться, что указатель текущего кортежа верно спозиционирован такими командами, как GOTO или TOP, перед выполнением REPLACE.

2.2.97. REPORT (dBASE III Plus и Clipper)

REPORT выдает на экран или печать (или в ASCII файл) отчет по содержимому активного отношения в соответствии с подготовленной ранее формой отчета командой CREATE/MODIFY REPORT.

Синтаксис. Для dBASE III Plus:
REPORT FORM ⟨filename⟩/? [⟨scope⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]
[PLAIN] [HEADING ⟨expC⟩] [NOEJECT]
[TO PRINT] [TO FILE ⟨filename⟩]
[SUMMARY]

Для Clipper:
REPORT FORM ⟨filename⟩ [⟨scope⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]
[PLAIN] [HEADING ⟨expC⟩] [NOEJECT] [TO PRINT] [TO FILE ⟨filename⟩]

По умолчанию — расширение для формы отчета .frm (FORM файл). Если условие не специфицировано, то все кортежи отношения включаются в отчет. Если в отчетной форме специфицированы группирование и итог по каждой группе (см. CREATE/MODIFY REPORT), то отношение предварительно должно быть проиндексировано или отсортировано по ключу группирования.

Опции. В dBASE III Plus следует использовать опцию ? (запрос к каталогу в случае открытого каталога) для выдачи списка всех отчетных форм для активного отношения, хранящихся в данном каталоге.

PLAIN позволяет печатать отчет с подавлением выдачи номеров страниц и системной даты; кроме того, страничный заголовок, определенный в отчетной форме, печатается лишь один раз на первой странице.

HEADING, сопровождаемая символьным выражением, определяет дополнительный заголовок, который печатается на каждой странице в строке номера страницы. Если заголовок — символьная строка, то нужно ограничить ее. Опции PLAIN и HEADING являются взаимно исключающими, причем PLAIN обладает большим приоритетом.

NOEJECT совместно с TO PRINT подавляет начальный прогон бумаги, т. е. отчет начинает распечатываться с первой страницы, которая попадает в принтер.

TO FILE отсылает отчет в текстовый файл.

SUMMARY подавляет вывод отдельных строк, выводя лишь подытоживающие записи по группам и итоговую информацию по отчету в целом.

Замечание. Clipper выдает ошибку времени выполнения при попытке деления на ноль. Это часто происходит при делении на пустое значение числового атрибута (т. е. этот кортеж еще не получил значения данного атрибута). Это может быть устранено путем определения функции

```
FUNCTION ZERO
PARAMETERS znum1, znum2
IF znum2 < > 0
   RETURN (znum1/znum2)
ENDIF
RETURN (0)
```

и вызовом ее всюду, где это необходимо (при создании — модификации формы отчета). В Clipper с этой целью используется специальная утилита REPORT.

См. также CREATE REPORT, MODIFY REPORT.

2.2.98. RESTORE FROM (dBASE III Plus и Clipper)

Активизирует переменные памяти, хранящиеся в специфицированном .mem файле.

Синтаксис. RESTORE FROM (filename) [ADDITIVE]

Если расширение файла не специфицировано, то подразумевается .mem.

Использование. При активизации в программе переменных памяти из .mem файла все активные до этого момента переменные уничтожаются. Все активизированные переменные получают статус локальных (PRIVATE]) независимо от того, каким был их статус, когда они запоминались в этом .mem файле. Однако для dBASE III Plus переменные памяти, активизированные в интерактивном режиме, становятся глобальными (получают статус PUBLIC).

В dBASE III Plus не может быть одновременно более 256 переменных памяти, а в Clipper — более 2048. dBASE III Plus отводит под переменные 6000 байт (это число может быть изменено), а в Clipper нет ограничений на размер для переменных памяти. Clipper позволяет хранить в символьных переменных данные типа memo.

Опции. Опция ADDITIVE позволяет добавлять активизируемые переменные к уже существовавшим, не уничтожая последние. При этом те переменные, которые были сохранены в .mem файле со статусом PUBLIC, вновь станут глобальными после активизации.

См. также PUBLIC, RELEASE, SAVE, STORE.

2.2.99. RESTORE SCREEN (только Clipper)

Употребляется для восстановления сохраненного ранее экрана.

Синтаксис. RESTORE SCREEN [FROM (memvar)]

Использование. Эта команда используется совместно с командой SAVE SCREEN, чтобы избежать повреждения вида экрана, которое может произойти при выдаче некоторой вспомогательной информации, графической информации или в других случаях.

Экран может быть сохранен в переменную памяти, длина которой не ограничивается в Clipper 256 символами.

См. также SAVE SCREEN.

2.2.100. RESUME (только dBASE III Plus)

Команда RESUME является интерактивной командой (всегда выдается из режима диалога) и работает совместно с командой SUSPEND. Вызывает продолжение исполнения программы, прерванной командой SUSPEND. Эта пара команд используется в основном при отладке.

Синтаксис. RESUME

Использование. При использовании команды RESUME командный файл, исполнение которого было приостановлено командой SUSPEND, продолжит свою работу начиная с командной строки,

следующей за той командной строкой, на которой его выполнение было приостановлено (имеется в виду следующее: либо нажат Esc, что вызывает команду SUSPEND, либо эта команда явно присутствует в тексте программы).

Хорошей практикой является использование команды CLEAR перед выдачей RESUME. Это гарантия того, что команды, которые были выданы в интерактивном режиме, после того как программа была приостановлена командой SUSPEND, не будут интерферировать с последующими после RESUME командами вывода на экран (средствами продолжившей исполнение программы).

См. также CANCEL, RETURN, SUSPEND.

2.2.101. RETRY (только dBASE III Plus)

RETRY возвращает управление в программу, которая вызвала данную подпрограмму (или вызвала ситуацию, обрабатываемую данной подпрограммой) таким образом, что еще раз выполняется строка исходной программы, повлекшая вызов подпрограммы.

Синтаксис. RETRY

Использование. RETRY повторяет исполнение той же строки. Это команда, альтернативная команде RETURN, которая ведет к выполнению следующей строки (после вызова подпрограммы).

Команда RETRY преимущественно используется в ситуациях обнаружения и обработки ошибок, в которых позволяет повторять некоторую команду до тех пор, пока она успешно не выполнится (без возбуждения ошибочной ситуации), предпринимая в перерывах между повторами некоторые действия по устранению ошибочной ситуации.

RETRY также может использоваться в некоторых циклических операциях для повторного исполнения dBASE III Plus программы, пока требуемое задание не будет выполнено.

RETRY может использоваться в процедурах и в командных файлах. Когда RETRY используется в командном файле, она закрывает его.

RETRY устанавливает в нуль значение, возвращаемое функцией ERROR( ).

См. также ERROR( ), RETURN.

2.2.102. RETURN (dBASE III Plus и Clipper)

RETURN используется в программах для организации возврата управления в вызывающую программу или в интерактивный режим для dBASE III Plus. Когда управление возвращается, выполняются команды, следующие за командой вызова

Синтаксис. Для dBASE III Plus:

RETURN [TO MASTER]

Для Clipper:

RETURN

Использование. Если в программе dBASE III Plus используется RETURN с опцией TO MASTER, то управление передается в вызывающую программу самого высокого уровня, т. е. если программа A вызвала программу A1, а та, в свою очередь, — A11, то использование RETURN TO MASTER в A11 осуществит возврат в точку вызова программы A.

В dBASE III Plus RETURN может использоваться в процедурах и в командных файлах, при использовании в командном файле RETURN закрывает его.

Если команда RETURN опущена в программе, dBASE III Plus осуществляет выход из нее и закрывает программный файл по достижении последней команды либо по достижении команды CANCEL, затем передает управление вызвавшей программе либо (если таковой нет) осуществляет выход в интерактивный режим.

RETURN уничтожает все локальные переменные (PRIVATE), но не затрагивает глобальных (PUBLIC).

RETURN устанавливает в нуль значение, возвращаемое функцией ERROR( ).

См. также CALL, DO, ERROR( ), FUNCTION, PRIVATE, PROCEDURE, PUBLIC.

2.2.103. RUN/! (dBASE III Plus и Clipper)

RUN позволяет выполнить любую требуемую программу операционной системы из программ dBASE III Plus или Clipper, выполняющих роль «оболочки».

Синтаксис. RUN ⟨command⟩ или
! ⟨command⟩

Использование. RUN выполняет специфицированную программу DOS и возвращает управление в программу dBASE III Plus или Clipper по окончании заданной программы.

Для dBASE III Plus RUN требует дополнительную память (к тем 256K, которые необходимы для dBASE III Plus) порядка 20K для копии командного процессора плюс память, требуемую вызываемой программой. Если компьютер не имеет требуемого объема памяти, dBASE III Plus выдаст сообщение «Insufficient memory» («Недостаточно памяти») в ответ на команду RUN.

Файл COMMAND.COM должен находиться в месте (диск, директорий), определенном текущим значением параметра операционной среды COMSPEC (см. документацию по DOS). Для уточнения можно использовать команду DOS — SET. Например, SET COMSPEC = C:\COMMAND.COM.

Под ⟨command⟩ в синтаксисе команды понимается не только имя вызываемой программы, но и передаваемые ей параметры. При передаче параметров можно использовать макроаппарат dBASE III Plus / Clipper.

Предупреждение. Некоторые программы DOS остаются в памяти (возможно, частично) по окончании своей работы. Например: ASSIGN, PRINT и т. д. Такие программы лучше исполнять перед dBASE III Plus, так как по окончании RUN память должна вернуться к dBASE III Plus, чтобы дать возможность системе продолжить обработку информации.

2.2.104. SAVE SCREEN (только Clipper)

Употребляется в Clipper для сохранения образа экрана в оперативной памяти.

Синтаксис. SAVE SCREEN [TO ⟨memvar⟩]

Использование. Эта команда используется совместно с командой RESTORE SCREEN, чтобы избежать повреждения вида экрана,

которое может произойти при выдаче некоторой вспомогательной информации, графической информации или в других случаях.

Команда позволяет сохранить состояние экрана в символьную переменную, которая для Clipper может превосходить по длине 256 символов. Эта переменная может далее выступать в качестве аргумента различных функций обработки строк, что позволяет гибко манипулировать содержимым экрана. Затем экран может быть восстановлен с помощью команды RESTORE SCREEN FROM ⟨memvar⟩.

См. также RESTORE SCREEN.

2.2.105. SAVE TO (dBASE III Plus и Clipper)

Эта команда служит для сохранения всех или части переменных памяти в .mem файле.

Синтаксис. SAVE TO ⟨filename⟩ [ALL LIKE/EXCEPT ⟨skeleton⟩]

По умолчанию выбирается активный диск и расширение файла .mem.

Если опции ALL/LIKE/EXCEPT не используются, то сохраняются все переменные памяти.

Использование. В шаблоне (⟨skeleton⟩) знак вопроса «?» обозначает одиночный произвольный символ, а звездочка «*» — один или более символов.

В случае, если уже существует файл с именем, указанным в качестве параметра TO, dBASE III Plus не перепишет его без выдачи запроса на разрешение переписи в случае, если выполнена команда SET SAFETY ON.

См. также DECLARE, RESTORE FROM, SET SAFETY, STORE.

2.2.106. SEEK (dBASE III Plus и Clipper)

Команда осуществляет поиск по индексированному отношению (т. е. используя индексный механизм доступа) первого кортежа, у которого значение ключа индексирования совпадает со специфицированным в команде выражением.

Синтаксис. SEEK ⟨expression⟩

Если выражение — символьная строка, оно должно быть заключено в ограничители (' " [ ]). Если же это переменная памяти или числовое выражение, то ограничители не нужны.

Частичное задание ключа или задание подстроки работает только в случае определения выражения начиная с крайнего левого символа ключа индексирования. Например, если полное значение ключа — "Сидоров, Валерий", то "Сид" — значащее выражение для поиска, а "Валер" — нет.

Если поиск был безуспешным, то указатель текущего кортежа находится в конце отношения (EOF( ) = .T.) и, кроме того, для dBASE III Plus в режиме SET TALK ON выдается сообщение: «No find» («Не найден»). Для подавления выдачи этого сообщения используется SET TALK OFF.

В Clipper использование команды SEEK предпочтительнее использования FIND.

См. также FIND, INDEX, REINDEX, SET EXACT, SET DELETED, SET INDEX, USE.

2.2.107. SELECT (dBASE III Plus и Clipper)

Используется для переключения между рабочими областями; активизирует заданную рабочую область.

Синтаксис. Для dBASE III Plus:
SELECT ⟨work area/alias⟩/?
Для Clipper:
SELECT ⟨work area/alias⟩

Изначально по умолчанию активизируется первая рабочая область.

Значащие рабочие области — от 1 до 10 или от A до J. В dBASE III Plus в случае открытого каталога опция ? позволяет выбрать требуемую рабочую область из меню.

Для использования переменной в качестве параметра команды следует применить макрофункцию & (SELECT & ⟨memvar⟩).

Каждая рабочая область имеет номер (от 1 до 10 соответственно) и альтернативное имя (буква от A до J соответственно). Кроме того, при открытии отношения в некоторой рабочей области с последней можно связать и другое альтернативное имя (см. USE). Обычно команды манипулирования данными работают со значениями атрибутов текущего кортежа активного отношения, однако можно обрабатывать и данные из текущих кортежей любой рабочей области (где открыто отношение), используя для этого следующий синтаксис обращений:

⟨alias⟩ —⟩ ⟨fieldname⟩

Если используются команды SET FIELDS TO и SET FIELDS ON, то можно просматривать и изменять данные и в других рабочих областях. В каждой рабочей области может быть открыт свой форматный файл, но один и тот же форматный файл не может быть одновременно открыт в разных рабочих областях. Это касается не только форматных, но и всех типов файлов dBASE III Plus / Clipper. Если открыт каталог, то dBASE III Plus резервирует рабочую область 10 для работы с ним. Каждая рабочая область имеет свой указатель текущего кортежа. Переключение между рабочими областями не изменяет позиций указателей. Если RELATION (связь) не установлена, то команды, изменяющие положение указателя текущего кортежа, влияют только на активное отношение.

dBASE III Plus позволяет максимум 15 открытых одновременно файлов. Для эффективной работы с использованием многих рабочих областей (до 10) одновременно следует убедиться, что параметры FILES и BUFFERS в конфигурационном файле PC DOS — Config. sys установлены в значения 20 и 15 соответственно (см. документацию по операционной системе). В противном случае dBASE III Plus может отказаться открывать очередной файл.

См. также SET CATALOG, SET FIELDS, SET RELATION, SET VIEW, SKIP, USE, SELECT( ).

2.2.108. SET (только dBASE III Plus)

SET — интерактивный меню-управляемый монитор, предназначенный для просмотра/изменения многих параметров среды dBASE III Plus (SET-параметров).

Синтаксис. SET

Использование. Для выхода из SET меню следует ввести Esc. Такой выход сохраняет все сделанные в SET режиме изменения. Для

уточнения деталей по каждому параметру среды следует смотреть конкретную SET команду, относящуюся к данному параметру. Для изменения значения умолчания некоторого SET параметра нужно включить соответствующую SET команду в файл Config .db (имеется в виду использование dBASE III Plus).

SET меню содержит семь подменю.

1. Options (опции) — содержит SET команды вида бинарных переключателей ВКЛ/ВЫКЛ (ON/OFF), устанавливающие различные режимы работы, и SET команду SET DEVICE TO SCREEN/PRINT, имеющую лишь два возможных значения, например SET TALK ON/OFF, SET HELP ON/OFF.

2. Screen (экран) — это подменю позволяет менять цвет и атрибуты экрана.

3. Keys (клавиши) — позволяет перепрограммировать функциональные клавиши. Клавиша F1 резервируется для HELP, а остальные могут содержать определенные пользователем команды до 30 символов длиной.

4. Disk (диск) — позволяет определить диск по умолчанию и путь (директорий по умолчанию, где будут разыскиваться объекты, если они не найдены в текущем директории).

5. Files (файлы) — это меню позволяет открыть файл-протокол (ALTERNATE), а также требуемые форматные и индексные файлы. Для протокола следует определить имя, а выбор формата или индекса активизирует список возможных файлов из открытого каталога (либо текущего диска — директория), из которого надлежит сделать выбор.

6. Margin (границы) — позволяет установить левую границу печати в отчетах и ширину строки при выдаче значений memo-атрибутов.

7. Decimals (десятичные позиции) — определяет максимальное число десятичных позиций после запятой, выводимых как результат числовой функции или вычислений.

2.2.109. SET ALTERNATE (dBASE III Plus и Clipper)

Команда SET ALTERNATE записывает весь вывод, отличный от вывода полноэкранных интерактивных команд, в текстовый файл-протокол.

Синтаксис. SET ALTERNATE ON/OFF
SET ALTERNATE TO [⟨filename⟩]

Имя файла должно включать код диска, если файл находится не на диске умолчания. Если расширение не задано, то подразумевается .txt.

SET ALTERNATE TO закрывает открытый файл-протокол. CLOSE ALTERNATE — альтернативный синтаксис этой команды.

Использование. Эта команда состоит из двух частей: SET ALTERNATE TO ⟨filename⟩ создает текстовый файл-протокол. SET ALTERNATE ON записывает последовательный протокол ввода с клавиатуры и ответной реакции системы (выдачи информации на экран) в этот файл. Результат экранных команд, таких, как @. . . . . SAY. . . , EDIT, APPEND и т. п., не протоколируется.

Для протоколирования требуемых фрагментов сессии следует использовать команды SET ALTERNATE ON/OFF без использования команды CLOSE ALTERNATE. До тех пор пока не будет

дана команда CLOSE ALTEPNATE, данные добавляются в конец файла протокола.

Чтобы убедиться, что буфер записан на диск, нужно последовательно дать команды SET ALTERNATE OFF и CLOSE ALTERNATE (или SET ALTERNATE TO) перед использованием текстового файла протокола (например, для текстовой обработки).

Файл, создаваемый по SET ALTERNATE,— стандартный ASCII файл, и он может быть отредактирован любым текстовым редактором.

См. также CLOSE.

2.2.110. SET BELL (dBASE III Plus и Clipper)

Этот переключатель определяет, будет ли звуковая реакция (сигнал) в случае ошибочного ввода или достижения конца ввода (области переменной).

Синтаксис. SET BELL ON/OFF

Значение по умолчанию — ON. Сигнал предупреждения выдается в указанных случаях до получения команды SET BELL OFF.

См. также SET CONFIRM.

2.2.111. SET CARRY (только dBASE III Plus)

Этот переключатель определяет, будут ли копироваться данные из предыдущего кортежа в новый, добавляемый для команд APPEND, INSERT.

Синтаксис. SET CARRY ON/OFF

Значение по умолчанию — OFF.

Использование. Иногда бывает очень эффективным режим добавления новых кортежей, при котором в каждый добавляемый кортеж копируются данные из предыдущего (когда кортежи различаются незначительно).

Переключатель не влияет на работу команд INSERT BLANK или APPEND BLANK, результатом которых всегда является добавление пустого кортежа.

Добавляемый кортеж не сохраняется (не фиксируется) до тех пор, пока в нем не проделаны какие-либо изменения. При желании зафиксировать кортеж без изменений (фактически продублировать предыдущий) нужно просто перепечатать один символ.

В команде INSERT BEFORE содержимое кортежа, расположенного перед текущим, копируется в добавляемый кортеж. Например, если даны команды GO 4, INSERT BEFORE, то в добавленный кортеж скопируется содержимое кортежа # 3.

См. также @. . .SAY. . .GET. . . , APPEND, INSERT, READ, SET FORMAT.

2.2.112. SET CATALOG (dBASE III Plus)

Эта команда дает возможность создавать и активизировать отношения и другие объекты dBASE III Plus без добавления их к открытому каталогу, не закрывая каталог.

Синтаксис. SET CATALOG ON/OFF

Значение по умолчанию — OFF, однако команда SET CATALOG TO ⟨catalog name⟩ автоматически переключает SET CATALOG ON.

Использование. Когда выполняется SET CATALOG ON, открытый каталог обновляется автоматически при использовании таких команд, как CREATE, INDEX, CREATE/MODIFY REPORT и др.

SET CATALOG OFF предохраняет объекты, используемые в качестве параметров упомянутых выше команд, от внесения в открытый каталог. Однако в отличие от SET CATALOG TO команда SET CATALOG OFF оставляет каталог открытым и позволяет использовать опцию ? (запрос списка объектов из каталога) в командах. Это удобно в ряде случаев, например, может потребоваться создать какие-то временные промежуточные объекты без фиксации их в каталоге (не закрывая его при этом).

См. также SET CATALOG TO.

2.2.113. SET CATALOG TO (только dBASE III Plus)

Эта команда либо создает и открывает новый каталог, либо открывает существующий, либо закрывает открытый.

Синтаксис. SET CATALOG TO [⟨.cat filename⟩/?]

SET CATALOG TO закрывает открытый каталог. Каталоги — это специализированные отношения, имеющие расширение .cat в имени содержащего файла.

Использование. SET CATALOG TO ⟨.cat filename⟩ открывает названный каталог, если он существует. Иначе, такой каталог создается.

SET CATALOG TO ? активизирует меню существующих каталогов, из которых можно выбрать требуемый. Список каталогов берется из главного каталога.

Опция ? является опцией запроса к каталогу. Она запрашивает в главном каталоге список всех имеющихся каталогов. Как только некоторый каталог открыт, можно использовать эту опцию для запроса списка требуемых объектов в различных командах (например, USE?). При запросе списка индексов, форматов и т. п. выдается список только тех объектов запрошенного типа, которые относятся к активному отношению.

dBASE III Plus всегда проверяет, существует ли главный каталог. Если нет, то он создается. Главный каталог создается в корневом директории dBASE III Plus активного диска (имеется в виду тот директорий, откуда был загружен и где находится dBASE III Plus).

При создании нового каталога dBASE III Plus предлагает ввести некоторый заголовок, поясняющий назначение создаваемого каталога. Он может быть введен на русском языке и будет фигурировать в списке каталогов при выборе с помощью опции ?.

Когда открывается каталог, рабочая область 10 резервируется системой для его ведения (обновления — просмотра).

Все создаваемые или активизируемые объекты добавляются в открытый каталог, если SET CATALOG ON. При этом dBASE III Plus запрашивает о заглавии для каждого добавляемого объекта, которое может быть введено по-русски и используется в дальнейшем в командах при выборе с помощью опции ? .

При открытии каталога dBASE III Plus автоматически проверяет соответствие содержимого каталога и внешней памяти. Если

некоторых файлов, хранящих объекты, зарегистрированные в каталоге, не существует, то соответствующие объекты удаляются из каталога.

См. также SET CATALOG.

2.2.114. SET CENTURY (только dBASE III Plus)

Эта команда позволяет вводить и выводить столетие при работе с датами.

Синтаксис. SET CENTURY ON/OFF

Значение по умолчанию — OFF. т. е. столетия во вводимых/выводимых датах не фигурируют.

Значением по умолчанию при вводе дат является 20-е столетие, однако поскольку в поле значения типа дата есть место для столетия, то, если некоторые вычисления приводят к другому значению столетия, это значение сохраняется.

Хотя SET CENTURY ON выдает дату в виде 10 байт (ДД/ММ/ГГГГ или иначе — в зависимости от выбора способа представления), значения атрибута типа дата занимают всегда 8 байт.

Когда используется SET CENTURY OFF, то можно вводить только даты XX столетия.

См. также CTOD(), DATE(), DTOC().

2.2.115. SET COLOR (dBASE III Plus и Clipper)

Команда SET COLOR позволяет установить цвета на цветном мониторе или атрибуты экрана на монохромном мониторе.

Синтаксис. Для dBASE III Plus:

SET COLOR ON/OFF

SET COLOR TO[⟨standart⟩ [, ⟨enhanced⟩ [,⟨border⟩ [, ⟨background⟩]]]]

Для Clipper:

SET COLOR TO [⟨standart⟩ [, ⟨enhanced⟩ [,⟨border⟩]]]

Для dBASE III Plus команда SET COLOR ON/OFF осуществляет переключение между цветным и монохромным мониторами, если в конфигурации присутствуют оба. По умолчанию (вначале) используется тот монитор, с которого была осуществлена загрузка dBASE III Plus.

Команда SET COLOR TO устанавливает стандартные цвета: стандартный вывод — белым по черному; расширенный (GET-поля) — черным по белому; рамка экрана черная.

Параметры команды «стандартный вывод» и «расширенный вывод» задаются парой ⟨цвет символа⟩ — ⟨цвет фона⟩.

Clipper позволяет задавать цвета номерами, однако в одной команде следует пользоваться одним способом задания — или буквенным, или цифровым.

При задании цветов символов (а также цвета рамки) буквенным способом могут дополнительно использоваться символы: * — мерцание (только для символов), + — повышенная яркость. Это дает возможность расширить цветовой диапазон до 16 цветов с возможностью мерцания. Например, желтый цвет повышенной яркости — GR +.

Таблица цветов

Цвет	Буквенное обозначение	Цифровое обозначение (только для Clipper)	Примечание
Черный	N	0	
Синий	B	1	
Зеленый	G	2	
Бирюзовый	BG	3	
Красный	R	4	
Лиловый	RB	5	
Коричневый	GR	6	
Белый	W	7	
Пустой (без визуализации)	X		Только для dBASE III Plus

В dBASE III Plus, задавая цвет символов, можно выбрать значение «пусто» — X. Это удобно при вводе паролей — каждый символ выводится пробелом.

В dBASE III Plus для некоторых машин опция ⟨background⟩ (цвет фона) недоступна отдельно для разных видов вывода. В этом и только этом случае используется синтаксис команды с четырьмя параметрами, где 4-й параметр (⟨background⟩) задает цвет фона, общий для всех видов вывода. В этом случае при описании стандартного и расширенного выводов задается только цвет символов.

На монохромных мониторах dBASE III Plus позволяет специфицировать: U — атрибут подчеркивания, I — инверсный вывод.

На ПЭВМ Compaq следует употреблять эту команду так, как будто Compaq имеет цветной монитор.

См. также SET INTENSITY.

2.2.116. SET CONFIRM (dBASE III Plus и Clipper)

Этот параметр используется в режиме экранного редактирования. Он определяет, будет ли курсор автоматически перемещен в следующее поле ввода по достижении конца текущего, либо пользователь должен явно указывать конец ввода в данное поле нажатием Enter.

Синтаксис. SET CONFIRM ON/OFF

Значение по умолчанию — OFF.

Использование. С помощью SET CONFIRM OFF курсор автоматически перемещается к следующему вводному полю по заполнению текущего.

SET CONFIRM ON заставляет курсор оставаться на последнем знакоместе в текущем поле ввода до нажатия Enter.

См. также SET BELL.

2.2.117. SET CONSOLE (dBASE III Plus и Clipper)

SET CONSOLE включает/выключает выдачу информации на экран из программы.

Синтаксис. SET CONSOLE ON/OFF

По умолчанию — OFF.

Использование. SET CONSOLE влияет только на вывод, предназначенный для экрана. Вывод на принтер не управляется этой командой. Команда также не влияет на результат работы @... ...SAY...GET... команд.

В dBASE III Plus SET CONSOLE работает только из программы. В интерактивном режиме — всегда ON.

В режиме OFF ввод с клавиатуры допустим, хотя он невидим на экране.

Сообщения об ошибках и другая диагностика не подавляются этой командой. Для их отключения в dBASE III Plus следует использовать команду SET TALK OFF. В Clipper их нет.

2.2.118. SET DATE (dBASE III Plus и Clipper)

Эта команда определяет формат выдачи значений типа «дата». Хотя в документации по Clipper данная команда не описана, однако экспериментально установлено, что она работает (правда, с некоторыми ограничениями).

Синтаксис. SET DATE AMERICAN/ ANSI/ BRITISH/ITALIAN/ FRENCH/GERMAN

По умолчанию установлен формат AMERICAN.

Использование. Экспериментально установлено, что Clipper поддерживает форматы AMERICAN (по умолчанию) и BRITISH и не позволяет задавать FRENCH (хотя FRENCH и BRITISH форматы эквивалентны).

Рекомендуется задавать SET DATE BRITISH, что соответствует стандартному русскому представлению даты и допустимо как в dBASE III Plus, так и в Clipper.

В dBASE III Plus допустимы следующие форматы выдачи даты:

AMERICAN	— мм/дд/ гг
ANSI	— гг. мм. дд
BRITISH	— дд/мм/гг (соответствует русскому стандарту)
ITALIAN	— дд—мм—гг
FRENCH	— дд/мм/гг
GERMAN	— дд. мм. гг (соответствует русскому стандарту)

2.2.119. SET DEBUG (только dBASE III Plus)

SET DEBUG — это средство, используемое при отладке программы. Эта команда определяет, будет ли диагностический вывод, генерируемый в режиме SET ECHO, направляться на экран или на принтер.

Синтаксис. SET DEBUG ON/OFF

По умолчанию — OFF.

Использование. Когда SET DEBUG ON, вывод SET ECHO направляется на принтер. Это устраняет интерференцию (совмещение) между сообщениями программы и отладочными сообщениями.

Если SET DEBUG ON и SET ECHO ON, то следующая информация не выводится на принтер: TALK (например, число проиндексированных кортежей), LIST и DISPLAY, а также сообщения об ошибках.

См. также SET ECHO, SET TALK.

2.2.120. SET DECIMALS (dBASE III Plus и Clipper)

SET DECIMALS определяет минимальное число десятичных позиций (позиций после точки), которое выводится при выдаче результатов численных функций и вычислений.

Синтаксис. SET DECIMAL TO ⟨numeric expression⟩

По умолчанию выводится минимум 2 позиции.

Использование. SET DECIMALS влияет только на результат деления, SQRT( ), LOG( ) и EXP( ).

Для вычислений, не требующих умножения, число позиций после точки в результате совпадает с максимальным у операндов. Если умножение используется в операции, то число десятичных позиций после точки в результате равно сумме чисел позиций сомножителей.

См. также SET FIXED.

2.2.121. SET DEFAULT (dBASE III Plus и Clipper)

SET DEFAULT позволяет определить диск умолчания— диск, на котором исполняются все операции поиска по умолчанию (и др.) и запоминаются все файлы (если дисковый спецификатор не задан явно).

Синтаксис. SET DEFAULT TO ⟨drive⟩

Изначально диском по умолчанию является тот, с которого произошла загрузка.

SET DEFAULT не проверяет, существует или нет названный диск, и не изменяет активный диск PC DOS.

См. также SET PATH.

2.2.122. SET DELETED (dBASE III Plus и Clipper)

SET DELETED определяет, будут ли кортежи, которые помечены к удалению, приниматься во внимание при работе других команд.

Синтаксис. SET DELETED ON/OFF

По умолчанию — OFF.

Использование. Команды INDEX и REINDEX всегда рассматривают все кортежи независимо от статуса SET DELETED.

Если SET DELETED ON, то большинство команд работают так, как будто удаленных кортежей не существует. Так, например, команды LOCATE и LIST не работают с удаленными кортежами. Но некоторые команды игнорируют признак удаления и включают удаленные кортежи в рассмотрение: DISPLAY (текущий кортеж), DISPLAY RECORD ⟨n⟩ GO/GOTO ⟨n⟩ либо в любой команде с опцией RECORD или NEXT ⟨n⟩.

Если SET DELETED ON, то команда RECALL ALL не восстанавливает никакие кортежи (необходимо указать кортежи явно).

См. также DELETED( ).

2.2.123. SET DELIMITERS (dBASE III Plus и Clipper)

Команда определяет вид ограничителей (и их присутствие/ отсутствие) для полей ввода в режиме полноэкранного редактирования.

Синтаксис. SET DELIMITERS ON/OFF
SET DELIMITERS TO [⟨expC⟩ /DEFAULT]

По умолчанию — SET DELIMITERS OFF. Поля ввода выделяются инверсией или повышенной яркостью либо другой цветовой парой (см. SET COLOR, SET INTENSITY).

Команда SET DELIMITERS ON использует символ: для ограничения полей ввода, если другой ограничитель не задан командой SET DELIMITERS TO.

Использование. Команда SET DELIMITERS TO ⟨character string⟩ определяет символы, которые в дальнейшем используются в качестве ограничителей полей ввода. Символьная строка может содержать один или два символа. Если задан один символ, то он является и левым и правым ограничителем, иначе первый символ задает левый, а второй — правый ограничитель для полей ввода. При задании строки длиной более двух символов только первые два используются в команде.

Чтобы ограничители использовались SET DELIMITERS, должно быть ON.

Для возврата к стандартным ограничителям «:» используется команда SET DELIMITERS TO DEFAULT.

См. также @. . .SAY. . .GET. . ., APPEND, CHANGE, EDIT, INSERT, READ, SET COLOR, SET INTENSITY.

2.2.124. SET DEVICE (dBASE III Plus и Clipper)

Эта команда определяет, куда будут выводиться результаты @. . .SAY. . . команд — на экран или на принтер.

Синтаксис. SET DEVICE TO PRINT/SCREEN

По умолчанию команда SET DEVICE TO SCREEN определяет вывод на экран.

Использование. Если задано SET DEVICE TO PRINT, то весь вывод при выполнении команд @. . .SAY. . . идет на принтер, а команды @. . .GET. . . игнорируются (или GET. . . компоненты в командах @. . .SAY. . .GET. . .). Команда @, которая требует от принтера возврата назад (задан номер строки меньше предыдущего), ведет к переводу листа (page eject).

Для некоторых принтеров команды @. . .SAY. . . могут отрабатываться не мгновенно, а по заполнению буфера (происходит печать).

См. также @, SET FORMAT.

2.2.125. SET DOHISTORY (только dBASE III Plus)

Команда SET DOHISTORY определяет, будут или нет записываться команды программы (командного файла) в специальный архив (HISTORY) по мере их выполнения. Команда используется в основном при отладке программ.

Синтаксис. SET DOHISTORY ON/OFF

По умолчанию — OFF запись трассы выполнения программы в архив не ведется.

Использование. Когда SET DOHISTORY ON, команды из командных файлов запоминаются в архиве (HISTORY) в порядке их выполнения. Можно затем редактировать их в режиме экранного редактирования.

Заметим, что SET DOHISTORY не работает по отношению к командам, которые даются в интерактивном режиме. dBASE III Plus всегда запоминает их в архив (стек), который может быть использован при конструировании новых команд,— нажатие клавиши

«стрелка вверх» вызывает появление предыдущей команды и т. д., выбранная команда может быть отредактирована и вновь использована.

В сочетании с SUSPEND и DISPLAY HISTORY SET DOHISTORY позволяет организовать эффективный режим отладки. Однако ввиду того, что эта команда замедляет выполнение, ее использование целесообразно только при отладке.

См. также DISPLAY HISTORY, LIST HISTORY, RESUME, SET DEBUG, SET ECHO, SET HISTORY, SET STEP, SUSPEND.

2.2.126. SET ECHO (только dBASE III Plus)

SET ECHO позволяет выводить строки исполняемой программы (по мере исполнения) на экран и/или принтер. Это отладочное средство.

Синтаксис. SET ECHO ON/OFF

По умолчанию — OFF.

Использование. Обычно инструкции программы не выводятся в ходе выполнения. SET ECHO ON — одно из четырех отладочных средств dBASE III Plus. Три остальных — SET DEBUG, SET STEP и SET TALK.

Clipper имеет свой внутренний отладчик, который может быть «подлинкован» для отладки к разрабатываемому приложению.

См. также SET DEBUG, SET STEP, SET TALK.

2.2.127. SET ESCAPE (dBASE III Plus и Clipper)

Эта команда имеет некоторые различия в интерпретации в dBASE III Plus и в Clipper, однако в целом ее смысл сводится к тому, позволяется или нет пользователю прерывать нормальное выполнение программы.

Синтаксис. SET ESCAPE ON/OFF

По умолчанию — ON.

Использование. Для dBASE III Plus значение ON этого параметра позволяет с помощью нажатия клавиши Esc:

находясь в интерактивном режиме, прервать интерпретацию команд (например, выдачу информации по команде LIST) — в этом случае dBASE III Plus выдает сообщение ✳✳✳ INTERRUPTED ✳✳✳ (прервано) и возвращается в режим диалога (ожидания команды);

находясь в режиме исполнения программы, прервать исполнение, при этом dBASE III Plus выдает сообщение

Called from ⟨filename .prg⟩
what now — Cancel, Ignore, Suspend? (C, I, or S) и можно

выбрать Cancel (прекратить выполнение программы), Ignore (продолжить выполнение) либо Suspend (приостановить выполнение, выполнить несколько команд в интерактивном режиме, а затем, возможно, продолжить — по команде RESUME).

Если SET ESCAPE OFF, то нет возможности средствами dBASE III Plus прервать исполнение программы.

В Clipper эта команда разрешает/запрещает использование ключа Alt—C для прерывания выполнения программы. Если выполнена команда SET ESCAPE ON (по умолчанию), то по нажатию

клавиш Alt—C в правом верхнем углу экрана возникает сообщение (Q, A, I)?, где: Q = Quit (прекращение выполнения с закрытием всех открытых файлов); A = Abort to DOS (немедленный выход в PC DOS — сброс задания); I = Ignore and continue (продолжение выполнения программы).

Кроме того, при SET ESCAPE ON можно выйти из GET-инструкции по нажатию клавиши Esc, игнорируя опцию VALID.

В случае SET ESCAPE OFF эти возможности отсутствуют.

См. также ON ERROR/ESCAPE/KEY.

2.2.128. SET EXACT (dBASE III Plus и Clipper)

Эта команда определяет алгоритм сравнения двух символьных строк.

Синтаксис. SET EXACT ON/OFF

Значение по умолчанию — OFF.

Использование. Если SET EXACT OFF, то сравнение двух символьных строк начинается с крайнего левого символа в каждой и продолжается символ за символом до исчерпания строки, стоящей справа от знака =. Таким образом, если вторая строка совпадает с началом первой, то они считаются равными.

В случае SET EXACT ON обе строки должны совпадать и по длине. Исключением является наличие пробелов, в Clipper эта ситуация решается с помощью введения специального оператора ==, действие которого похоже, но не эквивалентно обычному сравнению в режиме SET EXACT ON.

2.2.129. SET FIELDS (только dBASE III Plus)

Эта команда позволяет использовать либо игнорировать список атрибутов, определенный ранее командой SET FIELDS TO.

Синтаксис. SET FIELDS ON/OFF

По умолчанию — OFF.

Использование. Когда SET FIELDS находится в состоянии OFF, все атрибуты активного отношения доступны. Атрибуты других открытых отношений также доступны для вывода путем использования их альтернативных имен (отношений) в качестве расширения имени ⟨alias⟩ — ⟨fieldname⟩.

Когда SET FIELDS ON, только те атрибуты, которые присутствуют в списке, заданном в команде SET FIELDS TO, могут быть выведены или введены. Это относится как к активному, так и ко всем открытым отношениям.

SET FIELDS ON позволяет активизировать виртуальное отношение, состоящее из атрибутов как активного отношения, так и других открытых отношений. Значения кортежей этого отношения могут просматриваться /редактироваться.

См. также SET FIELDS TO, SET VIEW TO.

2.2.130. SET FIELDS TO (только dBASE III Plus)

Команда SET FIELDS TO определяет список атрибутов, которые образуют виртуальное отношение. Это отношение затем может быть активизировано командой SET FIELDS ON. Атрибуты могут

быть взяты не только из активного отношения, но и из любых открытых отношений. Эта команда также переопределяет список атрибутов — выражений, используемый многими командами по умолчанию.

Синтаксис. SET FIELDS TO [⟨field list⟩ /ALL]

Команда SET FIELDS TO Enter удаляет из виртуального отношения те его атрибуты, которые присутствуют в активном отношении.

Команда SET FIELDS TO ALL включает в виртуальное отношение все атрибуты активного отношения.

Использование. Виртуальное отношение неактивно до выдачи команды SET FIELDS ON.

SET FIELDS TO никак не упорядочивает список атрибутов, а только ограничивает доступные атрибуты. Эта команда употребляется также совместно с командой CREATE VIEW FROM ENVIRONMENT для определения атрибутов, которые образуют внешнее представление. Дополнительные или последовательные команды SET FIELDS TO добавляют атрибуты к существующему списку.

Для включения в создаваемое виртуальное отношение атрибутов из других открытых отношений (не из активного) нужно переключиться на требуемую рабочую область с помощью команды SELECT либо использовать расширенное имя атрибута (с указанием альтернативного имени требуемого отношения — ⟨alias⟩ — ⟩ ⟨fieldname⟩). Можно сформулировать три правила управления списком атрибутов и возможной неоднозначностью их имен:

необходимо использовать расширенное имя (альтернативное имя отношения совместно с именем атрибута — ⟨alias⟩ —⟩ ⟨fieldname⟩) при ссылке на атрибут не связанного с активным отношения (см. SET RELATION) из другой рабочей области;

необходимо использовать аппарат расширенных имен для ликвидации неоднозначности в именах (одинаковое имя атрибута в различных отношениях);

не нужно использовать расширенное имя при ссылке на атрибут активного отношения или связанного отношения (если нет ситуации неоднозначности).

Команды LIST STRUCTURE и DISPLAY STRUCTURE помечают атрибуты активного отношения, входящие в список атрибутов виртуального отношения, специальным символом (код 10h) в тех случаях, когда SET FIELDS ON.

Замечание. SET FIELDS TO может создать виртуальное отношение из многих отношений, но в действительности не связывает их, т. е. не устанавливает соответствия между значениями атрибутов в образующих данное виртуальное отношение отношениях. Это соответствие следует установить с помощью команды SET RELATION, и тогда это будет действительно виртуальное отношение (ему соответствует объект dBASE III Plus — «внешнее представление» (VIEW) — см. CREATE/MODIFY VIEW), либо можно программно устанавливать виртуальные отношения, задавая алгоритм конструирования значащего кортежа непосредственно в dBASE программе.

Предупреждение. При использовании SET FIELDS ON без предварительного определения списка атрибутов командой SET FIELDS TO никакие атрибуты доступны не будут, т. е. получится виртуальное отношение с пустым списком атрибутов.

Команда SET FIELDS TO влияет на ряд других выдаваемых впоследствии команд. Среди них команды, содержащие FIELDS опцию как часть своего синтаксиса. SET FIELDS TO позволяет

этим командам поддерживать множество атрибутов из многих отношений одновременно.

Интерпретация следующих команд затрагивается SET FIELDS TO: AVERAGE, BROWSE, CHANGE, COPY TO, COPY STRUCTURE, DISPLAY, EDIT, JOIN, LIST, SUM, TOTAL.

Перед использованием SET FIELDS TO списком атрибутов по умолчанию для этих команд были все атрибуты активного отношения. SET FIELDS TO переопределяет этот список.

Есть также две команды — @...GET... и REPLACE, которые оперируют атрибутами и имеют возможность работы с атрибутами других открытых отношений (не только активного) посредством задания расширенных имен. Эти команды могут работать только с атрибутами, указанными в списке атрибутов виртуального отношения в случае SET FIELDS TO, SET FIELDS ON.

Пользуясь SET FIELDS TO, VIEWS и SET RELATION, следует отдавать себе отчет в том, что:

команды APPEND и INSERT добавляют кортежи только в активное отношение, а не во все отношения данного внешнего представления;

APPEND и INSERT используют все атрибуты активного отношения (если не задан некоторый форматный файл, исключающий некоторые атрибуты из рассмотрения APPEND).

Конечно, можно выполнить APPEND BLANK или INSERT BLANK в различных рабочих областях, открыть форматный файл с различными атрибутами из различных отношений, открытых в этих областях, и затем использовать READ для добавления значений атрибутов во много отношений одновременно. Однако если эти отношения связаны, то это порождает проблему наличия пустых ключей у отношений-потомков в связи. Если при установлении связи типа отец — сын — внук между отношениями с помощью команды SET RELATION более чем один кортеж в отношении-потомке имеет значение ключа, требуемое соответствующим кортежем отношения-родителя, то всегда выбирается первый кортеж. Поэтому описанная процедура будет позиционировать указатель текущего кортежа в отношении-потомке на первый кортеж с пустым ключом, а не на только что добавленный.

При редактировании кортежей во внешнем представлении dBASE III Plus такие команды, как EDIT, BROWSE, @...GET... ...и READ, не обновляют значения ключевых атрибутов в связанных отношениях при изменении ключа в отношении-родителе, т. е. изменение ключей может нарушить целостность базы данных и, следовательно, должно выполняться осторожно.

Поскольку можно связать отношения логически в программе, а не с помощью команды SET RELATION, SET FIELDS TO не проверяет связность отношений, образующих создаваемое виртуальное. Это следует делать самостоятельно.

Команды INDEX, LOCATE, SET FILTER и SET RELATION игнорируют список команды SET FIELDS TO независимо от статуса SET FIELDS ON/OFF. Поэтому им доступны и те атрибуты активного отношения, которых нет в списке виртуального. Используя расширенные имена, можно применять указанные команды и к атрибутам других активных отношений.

Рекомендуется определять перечисленные команды над атрибутами активного отношения, однако SET FILTER часто требует задания атрибутов в других рабочих областях (имеется в виду тот факт, что для каждого открытого отношения может быть задан свой фильтр). Это требует внимания при закрытии отношений в других

рабочих областях. dBASE III Plus не проверяет наличие открытых отношений в требуемых рабочих областях при использовании расширенных имен в перечисленных выше командах, так как это замедляет выполнение программ.

Для соблюдения целостности базы данных при использовании внешних представлений (VIEWS), SET FIELDS или SET RELATION рекомендуется добавлять кортежи к одному реальному отношению за один раз, обеспечивая при этом доступ ко всем его атрибутам.

Изменения в ключевой информации должны делаться с учетом того, что изменения в ключе отношения-родителя без соответствующих изменений в ключах связанных отношений ведут к разрыву связи между кортежами.

Следует избегать конструирования виртуальных отношений из атрибутов многих несвязанных отношений, если нет веских причин делать это.

Использование многих связанных отношений ставит ряд вопросов неоднозначности и доступности определенных атрибутов. Существуют следующие правила управления доступом:

отношения, не затронутые SET FIELDS TO, сохраняют свои списки атрибутов по умолчанию, и атрибуты в таких отношениях доступны по расширенным именам (⟨alias⟩ —⟩ ⟨fieldname⟩);

атрибуты активного отношения доступны, только если они присутствуют в списке;

атрибуты в связанных отношениях доступны в случае, если они присутствуют в списке и указывается расширенное имя;

если активно отношение, имеющее атрибуты, входящие в виртуальное отношение (отношение, затронутое SET FIELDS TO), то отношения, не затронутые SET FIELDS TO, недоступны из активной рабочей области (их атрибуты);

если отношение не имеет связанных с ним отношений-потомков, то его собственные (внутренние) атрибуты не выводятся.

При использовании нерасширенных имен атрибутов dBASE III Plus осуществляет поиск по такому алгоритму: прежде всего просматривается активное отношение; в активном отношении вначале просматривается структура (порядок следования), а затем собственно список атрибутов; затем, если атрибут не найден в активном отношении, просматриваются связанные с ним отношения.

Замечание. Аппарат конструирования виртуальных отношений (имеются в виду SET FIELDS TO, SET FIELDS ON/OFF и VIEWS— внешние представления) не поддерживается в среде Clipper, поэтому его использование порождает определенную несовместимость разрабатываемых приложений. Кроме того, использование этого аппарата недостаточно хорошо документировано, что может послужить причиной появления непредсказуемых результатов.

Следовательно, данный аппарат нуждается в изучении и уточнении.

См. также @...SAY...GET..., BROWSE, CHANGE, COPY TO, COPY STRUCTURE, CREATE/MODIFY VIEW, CREATE VIEW FROM ENVIRONMENT, DISPLAY, DISPLAY STRUCTURE, EDIT, JOIN, LIST, REPLACE, SET FIELDS, SET RELATION, SET VIEW, TOTAL.

2.2.131. SET FILTER (dBASE III Plus и Clipper)

Эта команда позволяет наложить на отношение горизонтальный фильтр — представить его состоящим только из кортежей, удовлетворяющих условию фильтрации.

Синтаксис. Для dBASE III Plus:

SET FILTER TO [FILE ⟨filename⟩/?]
[⟨condition⟩]

Для Clipper:

SET FILTER TO [⟨condition⟩]

SET FILTER TO без задания условия фильтрации отключает (деактивирует) текущий фильтр.

Для dBASE III Plus SET FILTER обновляет (пополняет) каталог, если он открыт.

Использование. SET FILTER относится только к активному отношению, поэтому можно для каждого открытого отношения задать независимое условие фильтрации.

Все команды, требующие наличия активного отношения, такие, как AVERAGE, BROWSE, EDIT, REPORT, учитывают наличие фильтрующего условия.

Команда SET FILTER TO FILE (для dBASE III Plus) считывает фильтрующее условие из файла запроса (.qry), созданного с помощью CREATE/MODIFY QUERY. Если каталог открыт, то можно использовать опцию ? в команде (запрос к каталогу), вызывающей выдачу списка всех файлов запросов для активного отношения, хранящихся в каталоге, в виде меню.

Фильтр не активизируется до тех пор, пока не будет инициировано перемещение указателя текущего кортежа в отношении.

См. также CREATE/MODIFY QUERY.

2.2.132. SET FIXED (dBASE III Plus и Clipper)

SET FIXED определяет, будет ли для всех десятичных чисел зафиксировано одно (одинаковое) число позиций после десятичной точки при их выводе.

Синтаксис. SET FIXED ON/OFF

Значение по умолчанию — OFF.

Если SET DECIMALS не использовалось, то число десятичных позиций равно двум.

Использование. Когда SET FIXED OFF, число десятичных позиций, выводимых при выдаче того или иного результата, в общем случае различно и определяется в описании команды SET DECIMALS. Когда же SET FIXED ON, число десятичных позиций для любого вывода десятичных чисел фиксировано выданной ранее командой SET DECIMALS (либо равно 2 по умолчанию).

Са. также SET DECIMALS.

2.2.133. SET FORMAT (dBASE III Plus и Clipper)

Команда позволяет активизировать форматный файл (.fmt) — некоторую форму (внешний вид), используемую для организации интерфейса пользователя.

Синтаксис. Для dBASE III Plus:

SET FORMAT TO [⟨filename⟩//?]

Для Clipper:

SET FORMAT TO [⟨filename⟩]

Если расширение в имени форматного файла не задано, то подразумевается расширение .fmt. Команда SET FORMAT TO (либо CLOSE FORMAT) закрывает открытый формат. Команда SET FORMAT для dBASE III Plus обновляет (пополняет) каталог, если он открыт.

Использование. Если SET FORMAT не используется, то команды APPEND, CHANGE, EDIT и INSERT используют свою стандартную (системную) форму.

Команда SET FORMAT TO ? в случае открытого каталога (dBASE III Plus) активизирует меню всех форматных файлов, относящихся к открытому отношению.

В dBASE III Plus имеется генератор экранных форм CREATE/MODIFY SCREEN, позволяющий легко и эффективно сконструировать требуемую форму независимо от того, где затем будет использоваться соответствующий форматный файл: в dBASE III Plus или в Clipper.

Форматные файлы позволяют создавать в разрабатываемых приложениях полноэкранный выразительный интерфейс пользователя. Они очень удобны при решении задач вида заполнение бланков, анкет, форм и др.

Замечания по программированию. При необходимости использовать многостраничный форматный файл, в котором @. . .SAY. . . GET. . . продолжаются на 2—32 страницах, следует включать READ в местах, где надо организовать перевод страницы. Клавиши PgUp и PgDn вызывают листание страниц. Многостраничный формат работает только в случае, если форматный файл открывается командой SET FORMAT TO.

Clipper не очищает экран перед выполнением форматного файла. Он допускает наличие команд внутри форматного файла. Это позволяет более гибко использовать аппарат форматов.

См. также @. . .SAY. . .GET. . . . , APPEND, CHANGE, EDIT, INSERT, READ, SET DEVICE.

2.2.134. SET FUNCTION (dBASE III Plus и Clipper)

Эта команда используется для программирования функциональных клавиш. Нажатие каждой функциональной клавиши может генерировать последовательность символов (специфичную для данной клавиши) длиной до 30 символов. Команда SET FUNCTION позволяет установить соответствие между некоторой функциональной клавишей и последовательностью символов, поступающих в компьютер по ее нажатию.

Синтаксис. SET FUNCTION ⟨expN⟩ TO ⟨expC⟩ [;]

Использование. Для dBASE III Plus используется стандартное назначение клавиш:

Клавиша	Значение	Клавиша	Значение
F1	help	F6	display status
F2	assist	F7	display memory
F3	list	F8	display
F4	dir	F9	append
F5	display structure	F10	edit

Функциональная клавиша F1 не может быть перепрограммирована ни в dBASE III Plus, ни в Clipper. В Clipper с ней ассоциируется вызов процедуры HELP. PRG (если такая процедура включена в состав приложения) — см. HELP.

В Clipper количество функциональных клавиш расширено до 40:
с 1-й по 10-ю активизируются нажатием соответственно клавиш F1—F10;
с 11-й по 20-ю активизируются нажатием клавиши Shift и одновременно соответствующей функциональной клавиши (F1—F10);
с 21-й по 30-ю активизируется нажатием клавиши Ctrl и одновременно соответствующей функциональной клавиши (F1—F10);
с 31-й по 40-ю активизируются нажатием клавиши Alt и одновременно соответствующей функциональной клавиши (F1—F10).

Символьная строка в Clipper может иметь внутри себя управляющие символы (такие, как Ctrl — C — эквивалентно PgDn) для завершения READ команды. dBASE III Plus не позволяет вставлять управляющие символы в строку подстановки.

Символ ; обозначает возврат каретки (Enter), генерируемый в заданном месте, что позволяет, например, не только задать команду в dBASE III Plus, но и послать ее на выполнение.

2.2.135. SET HEADING (только dBASE III Plus)

SET HEADING определяет, будут или нет выводиться заголовки столбцов по каждому полю вывода в командах DISPLAY, LIST, SUM и AVERAGE.

Синтаксис. SET HEADING ON/OFF (по умолчанию — OFF).

Использование. Команды DISPLAY, LIST, SUM и AVERAGE выводят заголовки столбцов по каждому атрибуту, переменной памяти или выражению.

Ширина столбца — большее из двух: ширины заголовка и собственно выводимых значений.

2.2.136. SET HELP (только dBASE III Plus)

Эта команда определяет, будет ли появляться вопрос dBASE III Plus «Do you want some help (Y/N)?» («Нуждаетесь ли Вы во вспомогательной информации?») при выявлении ошибок в командах, данных в интерактивном режиме.

Синтаксис. SET HELP ON/OFF (по умолчанию — ON)

Использование. В случае ошибки и ответе «Y» на запрос о помощи dBASE III Plus выводит вспомогательную информацию по выданной команде (независимо от вида ошибки), переходя в режим HELP.

См. также HELP.

2.2.137. SET HISTORY (только dBASE III Plus)

Позволяет включать/отключать командный архив.

Синтаксис. SET HISTORY ON/OFF (по умолчанию — ON)

Использование. Архив команд по умолчанию включен, позволяя тем самым воспроизводить выданные ранее команды, редактировать и повторно выполнять их.

Команда SET HISTORY OFF аннулирует архив.

См. также DISPLAY HISTORY, LIST HISTORY, SET DOHISTORY, SET HISTORY TO.

2.2.138. SET HISTORY TO (только dBASE III Plus)

Эта команда позволяет специфицировать размер командного архива (стека) — максимальное количество команд, которое может там храниться, после чего первая сохраненная команда выталкивается из стека (уничтожается), давая возможность запомнить последнюю введенную.

Синтаксис. SET HISTORY TO ⟨expN⟩

По умолчанию число запоминаемых команд равно 20. Допустимый диапазон — от 0 до 16000.

Использование. Для вычисления объема памяти, требуемого под архив команд, следует прибавить 9 байт к числу байт в каждой команде.

Если HISTORY переустанавливается к числу команд, меньшему числа команд, уже лежащих в архиве, то архив очищается. Иначе он сохраняется, увеличивая свой объем.

Параметр MAXMEM в Config. db должен иметь значение, большее 256 Кбайт, если HISTORY выбирается большим числом. Иначе при использовании команды RUN могут быть потеряны команды, лежащие в архиве.

См. также DISPLAY HISTORY, LIST HISTORY, SET DOHISTORY.

2.2.139. SET INDEX (dBASE III Plus и Clipper)

Открывает поименованные индексные файлы.

Синтаксис. Для dBASE III Plus:

SET INDEX TO [⟨list of .ndx files⟩/?]

Для Clipper:

SET INDEX TO [⟨list of .ntx files⟩]

По умолчанию dBASE III Plus предполагает расширение .ndx, а Clipper — .ntx в именах индексных файлов. Индексы dBASE III Plus и Clipper различаются, однако механизм индексного доступа одинаков. В Clipper имеется специальная утилита INDEX, позволяющая проиндексировать по требуемому ключу отношение, кроме того, это можно сделать в программе (см. команду INDEX).

В случае открытого каталога в dBASE III Plus допустим синтаксис SET INDEX TO?, что вызывает выдачу меню всех допустимых для данного отношения индексов.

Использование. Этой командой может быть открыто до семи индексов (ранее созданных для этого отношения).

Первый индекс в списке является главным, или мастер-индексом. Он управляет механизмом доступа к отношению (порядком следования кортежей). Однако все открытые индексы автоматически модифицируются (и это является целью одновременного открытия более одного индекса), когда производимые в отношении изменения затрагивают соответствующие ключи индексирования.

Команда INSERT эквивалентна команде APPEND при использовании индексного механизма доступа.

Указатель текущего кортежа. После выполнения этой команды он устанавливается в начало отношения (в представлении мастер-индекса).

Clipper позволяет использовать макроаппарат при задании имен индексных файлов, но при этом каждый индексный файл должен представляться отдельной макропеременной.

2.2.140. SET INTENSITY (dBASE III Plus и Clipper)

SET INTENSITY определяет, выделять или нет поля ввода в командах, активизирующих режим экранного редактирования, таких, как APPEND, EDIT, @. . . SAY . . . GET . . . , @. . . . . .PROMPT. . .MESSAGE. . . и др.

Синтаксис. SET INTENSITY ON/OFF (по умолчанию — ON, что соответствует выделению полей ввода).

Использование. Эта команда является переключателем для двух наборов экранных атрибутов: стандартного и расширенного. Эти наборы определяются с помощью команды SET COLOR.

Когда INTENSITY установлено в ON, стандартный и расширенный атрибуты могут быть различными (как и по умолчанию). В случае INTENSITY OFF значения набора, соответствующего стандартному атрибуту, используются и для расширенного.

См. также SET COLOR.

2.2.141. SET KEY (только Clipper)

Эта команда позволяет выполнить некоторую процедуру по нажатию пользователем определенной клавиши во время ожидания ввода.

Синтаксис. SET KEY ⟨expN⟩ TO [⟨proc⟩]

Использование. Состояние ожидания ввода возбуждается одной из следующих команд: WAIT, READ, ACCEPT, INPUT, MENU TO.

⟨expN⟩ — числовое значение, возвращаемое функцией INKEY( ) при нажатии требуемой клавиши. Изначально система предполагает:

SET KEY ⟨Fl-code⟩ TO help

Как и в случае Help. prg (см. HELP), вызываемой процедуре передаются три параметра, которые должны быть в ней обязательно объявлены (даже если она их не использует).

SET KEY ⟨key-code⟩ TO отменяет назначение процедуры клавише.

2.2.142. SET MARGIN (dBASE III Plus и Clipper)

Команда SET MARGIN позволяет установить левую границу (отступ) для всего последующего вывода на печать. На дисплейный вывод эта команда не влияет.

Синтаксис. SET MARGIN TO ⟨expN⟩

По умолчанию левая граница (отступ) — 0.

2.2.143. SET MEMOWIDTH (только dBASE III Plus)

Эта команда позволяет регулировать ширину вывода значений атрибутов типа memo (текст). В Clipper для их редактирования/просмотра существует специальная функция — MEMOEDIT( ), кроме того, там допустимы и переменные памяти типа memo.

Синтаксис. SET MEMOWIDTH TO ⟨expN⟩

По умолчанию ширина вывода memo-атрибутов — 50.

2.2.144. SET MENU (только dBASE III Plus)

Эта команда определяет, будет или нет выдано меню клавиш перемещения курсора (меню навигации) в командах, инициирующих режим экранного редактирования.

Синтаксис. SET MENU ON/OFF (по умолчанию — ON).

Использование. В режиме экранного редактирования клавиша F1 служит переключателем — есть /нет меню навигации.

2.2.145. SET MESSAGE (dBASE III Plus и Clipper)

Эта команда имеет различный смысл (и синтаксис) в dBASE III Plus и Clipper.

В dBASE III Plus эта команда позволяет задать некоторую строку-сообщение (подсказку), возникающую в нижней строке экрана (строке сообщений).

В Clipper эта команда имеет совсем другой смысл. Она задает номер строки экрана, куда будет выводиться информация, поясняющая позиции некоторого меню, описываемого командами @. . . . . .PROMPT. . .MESSAGE. . . и активизированного командой MENU TO.

Синтаксис. Для dBASE III Plus:

SET MESSAGE TO [⟨cstring⟩]

Для Clipper:

SET MESSAGE TO [⟨expN⟩]

Использование. В dBASE III Plus команда SET MESSAGE TO (без параметра) убирает существующее сообщение, и в строке сообщений выводятся только сообщения dBASE III Plus.

Строка сообщений очень удобна при использовании экранных форм (создании своего режима экранного редактирования), если @. . . SAY. . .GET. . . команды не используют эту строку. Она может применяться для организации подсказки и других целей.

В Clipper SET MESSAGE TO позволяет определить номер строки экрана, в которую будет выводиться сообщение, поясняющее ту позицию меню, на которой находится курсор, и определяемое соответствующей командой @. . .PROMPT. . .MESSAGE. . . Команда SET MESSAGE TO 0 или SET MESSAGE TO отключает выдачу вспомогательных сообщений.

См. также SET STATUS.

2.2.146. SET ORDER (только dBASE III Plus)

Эта команда позволяет назначить главным (MASTER) индексом новый индекс из числа открытых или отключить индексный механизм доступа, не закрывая при этом открытых индексов.

Синтаксис. SET ORDER TO [⟨expN⟩]

Использование. В зависимости от числа открытых индексных файлов числовое выражение должно принимать значение от 0 до 7. Если оно равно 0, то отношение рассматривается в своем естественном (физическом) порядке — без использования индексного механизма доступа.

Команда SET ORDER позволяет улучшить временные характеристики при работе с отношением со многими индексами одновременно — при переопределении доступа к отношению через другой

индекс, так как она не закрывает и не открывает индексные файлы и не передвигает указатель текущего кортежа.

См. также NDX( ).

2.2.147. SET PATH (dBASE III Plus и Clipper)

Эта команда определяет путь, по которому dBASE III Plus или Clipper будут осуществлять поиск файлов, если они не найдены в текущем директории.

Синтаксис. SET PATH TO [⟨path list⟩]

Команда SET PATH TO без параметров ограничивает поиск текущим директорием активного диска.

Использование. SET PATH дает возможность доступа к требуемым файлам в древовидных структурах оглавлений DOS. Она определяет последовательность директориев, которую dBASE III Plus или Clipper должен просмотреть в поисках требуемого файла, если он не найден в текущем директории.

Не следует смешивать эту команду с командой PATH операционной системы, поскольку список директориев, задаваемый последней, не используется в среде dBASE III Plus/Clipper для поиска файлов.

Путь — это список директориев, разделенных запятыми либо символом ; . Символ ; , используемый в dBASE III Plus/Clipper в качестве признака переноса команды на новую строку, употреблять в качестве разделителя не рекомендуется. Clipper вообще запрещает переносить части этой команды, кроме того, этот символ в данном контексте вносит неоднозначность в синтаксис команды.

Замечание. SET PATH не оказывает влияния на вывод команды DIR, которая всегда выводит файлы в текущем либо в заданном директории. SET PATH влияет только на команды, осуществляющие поиск файлов. Если команда должна создать файл и необходимо, чтобы она создала его не в текущем директории, то следует указать путь явно в имени файла.

См. также DIR, SET DEFAULT.

2.2.148. SET PRINT (dBASE III Plus и Clipper)

SET PRINT ON вызывает дублирование всего неформатного вывода на принтер (имеется в виду весь вывод, кроме @. . .SAY. . .).

Синтаксис. SET PRINT ON/OFF (по умолчанию — OFF).

Использование. В интерактивном режиме все, кроме вывода экранного режима (@. . .SAY. . . , APPEND, EDIT и др.), дублируется на принтер.

Для направления потока @. . .SAY. . . на принтер следует использовать SET DEVICE TO PRINT.

Специальные случаи. Некоторые принтеры буферизуют вывод, т. е. принтер может начать выдавать информацию не сразу по выдаче SET PRINT ON, а через несколько команд.

См. также SET DEVICE, SET FORMAT.

2.2.149. SET PRINTER (только dBASE III Plus)

Эта команда позволяет выбрать одно из штатных устройств DOS в качестве принтера.

Синтаксис. SET PRINTER TO ⟨DOS device⟩

По умолчанию используется назначение LPT1. dBASE III Plus направляет весь принтерный вывод на это устройство, пока не будет сделано иное назначение командой SET PRINTER.

Допустимыми именами устройств являются параллельные порты печати LPT1, LPT2, LPT3 и последовательные порты связи COM1 и COM2. Иными словами, если компьютер оснащен надлежащим образом, то можно организовать вывод из dBASE III Plus на различные устройства вывода (в том числе и переслать некоторую информацию на другой компьютер через порты связи).

См. также SET PRINT.

2.2.150. SET PROCEDURE (dBASE III Plus и Clipper)

Открывает поименованный файл, содержащий процедуры.

Синтаксис. SET PROCEDURE TO [⟨procedure filename⟩]

Если код диска не задан, подразумевается активный, а расширением по умолчанию считается .prg.

Использование. SET PROCEDURE TO или CLOSE PROCEDURE закрывает открытый процедурный файл.

В dBASE III Plus процедурный файл — это контейнер для процедур, который может содержать до 32 процедур. Начало каждой процедуры идентифицируется командой PROCEDURE. Одновременно может быть открыто не более одного процедурного файла. Механизм процедурных файлов в dBASE III Plus позволяет структуризовать разрабатываемую программу и улучшить ее временные характеристики путем выделения набора подпрограмм-процедур, которые могут содержать формальные параметры (см. PARAMETERS), в отдельный файл (файлы) с последующей активизацией такого файла (загрузкой содержащегося в нем набора процедур в память) и вызовом требуемых процедур из тела программы (возможно, с передачей фактических параметров).

Замечание. В dBASE III Plus при открытом процедурном файле не следует открывать новый, не закрыв этот. В этом случае новый файл откроется и команда будет проинтерпретирована правильно, однако счетчик открытых файлов увеличится (вместо того чтобы остаться неизменным: один файл закрылся, другой открылся), и повторные команды могут привести к превышению максимально допустимого в dBASE III Plus числа одновременно открытых файлов (15).

В Clipper эта команда имеет несколько другой смысл — она указывает компилятору файл, в котором содержится описание используемых в программе процедур, а возможно, и функций (так как Clipper позволяет определять новые функции — см. команду FUNCTION). Указанный файл будет открыт компилятором, и он будет разыскивать там (а также в основном компилируемом файле и в файлах, вызываемых по DO) используемые в программе процедуры и функции. Число процедур и/или функций, содержащихся в таком файле, не ограничено. Здесь можно открывать новые процедурные файлы, не закрывая (либо закрывая) предыдущие.

Замечание. Clipper требует, чтобы имя процедурного файла не использовалось в качестве имени одной из содержащихся в нем процедур и/или функций.

Вызов процедуры в dBASE III Plus выполняется с помощью команды DO . . . [WITH. . .], а в Clipper возможен и такой вызов, и вызов с помощью команды CALL. . . [WITH. . .].

См. также CALL, DO, FUNCTION, PARAMETERS, PROCEDURE.

2.2.151. SET RELATION (dBASE III Plus и Clipper)

Эта команда устанавливает связь между двумя отношениями в соответствии с ключевым условием, которое является общим для обоих отношений. В Clipper эта команда позволяет связать одно отношение со многими.

Синтаксис. Для dBASE III Plus:

SET RELATION TO [⟨key exp⟩/RECNO ( )/ ⟨expN⟩ INTO ⟨alias⟩]

Для Clipper:

SET RELATION TO [⟨key exp1⟩ /RECNO ( )/ ⟨expN1⟩ INTO ⟨alias1⟩] [, TO ⟨key exp2⟩/RECNO( )/ ⟨expN2⟩ INTO ⟨alias2⟩. . .]

SET RELATION TO без параметров разрывает связь текущей рабочей области (с другими: сыном и/или отцом).

Использование. SET RELATION связывает активное отношение с любым открытым в другой рабочей области отношением. dBASE III Plus позволяет связывание только с одним, а Clipper — одновременно со многими.

INTO опция. Отношения идентифицируются альтернативными именами (т. е. можно ссылаться на буквенный идентификатор требуемой рабочей области — от A до J соответственно либо собственно имя отношения).

Замечание. Следует помнить, что в dBASE III Plus с помощью команды CREATE/MODIFY VIEW можно связывать отношения, пользуясь удобным меню-управляемым интерфейсом и, кроме того, можно сохранять созданные связи в виде внешних представлений (.vue файлов — см. п. 1.5).

Опции. Связь можно осуществлять одним из трех способов: по ключевому выражению; по номеру кортежа; по числовому выражению.

При связывании по ключевому выражению отношение, с которым устанавливается связь, должно быть проиндексировано, причем возможные значения ключевого выражения как раз и будут выступать в качестве значений ключа индексирования при поиске (связывании кортежей). Алгоритм связывания следующий:

при перемещении указателя текущего кортежа в активном отношении вычисляется ключевое выражение (по значениям, взятым из кортежа, на который указывает указатель);

затем полученное значение используется в качестве ключа индексирования и делается попытка найти в отношении-сыне кортеж с этим ключом;

если такой кортеж не найден, то указатель текущего кортежа в отношении-сыне позиционируется в конец отношения — EOF( )= =.T. (пустой кортеж).

При связывании по номеру записи отношение, с которым устанавливается связь, не должно быть проиндексировано. В этом случае, когда в активном отношении смещается указатель текущего кортежа, в отношении-сыне указатель также смещается на кортеж с тем же номером. Если номер превышает количество кортежей, содержащихся в отношении-сыне, возникает ситуация «конец файла» (EOF ( ) = .T.), и указатель указывает на пустой кортеж.

При связывании по числовому выражению отношение-сын не должно быть индексировано. При смещении указателя текущего кортежа в активном отношении номер кортежа в отношении-сыне

равен значению числового выражения, вычисленному по значениям из нового кортежа в отношении-родителе (активном отношении).

В dBASE III Plus можно воспользоваться командой CREATE VIEW ⟨.vue filename⟩ FROM ENVIRONMENT для сохранения установленных связей в виде внешнего представления.

См. также CREATE/MODIFY VIEW, SET FIELDS, SET FIELDS TO, SET VIEW.

2.2.152. SET SAFETY (только dBASE III Plus)

Эта команда позволяет установить определенный уровень защиты от случайной переписи или уничтожения файлов. Это осуществляется путем выдачи пользователю сообщения-запроса на подтверждение при выполнении подобных операций.

Синтаксис. SET SAFETY ON/OFF (по умолчанию — ON).

Использование. SET SAFETY ON обеспечивает выдачу сообщения вида «⟨filename⟩ already exists, overwrite it? (Y/N)», требующего подтверждения очередной операции, связанной с уничтожением/переписью файлов. Это важно, например, при наличии большого количества отношений и индексов: можно, создавая новый объект, попытаться назвать его именем уже существующего и т. п.

SET SAFETY OFF отменяет режим выдачи такого сообщения.

См. также COPY, COPY FILE, CREATE, JOIN, SAVE, SORT, TOTAL, UPDATE, ZAP.

2.2.153. SET SCOREBOARD (только dBASE III Plus)

SET SCOREBOARD определяет, будут ли вспомогательные сообщения dBASE III Plus появляться в статусной строке. Статусной считается строка 0 в случае STATUS OFF и строка 22, если STATUS ON.

Синтаксис. SET SCOREBOARD ON/OFF (по умолчанию — ON)

Использование. В случае SCOREBOARD ON и STATUS OFF можно видеть некоторые сообщения dBASE III Plus в нулевой строке экрана. Это индикация отметки текущего кортежа как удаленного в режиме экранного редактирования (Del), а также указание на заполнение поля ввода в команде @. . .SAY. . .GET. . . значением, выходящим из специфицированного в команде диапазона.

Если же SCOREBOARD ON и STATUS ON, то dBASE III Plus выдает расширенную диагностическую информацию, часть которой появляется в статусной строке (22), а часть — в строке навигации и ошибок (23). Статус помеченности кортежа к удалению, статус текущего режима редактирования вставка — замена — (Ins/), индикатор переписи и другие появляются в статусной строке, а ошибки ввода данных — в строке навигации и ошибок.

При SET SCOREBOARD OFF и SET STATUS OFF dBASE III Plus не выводит указанных индикаторов и сообщений.

См. также SET STATUS.

2.2.154. SET STATUS (только dBASE III Plus)

Определяет, будет ли статусная строка выводиться внизу экрана при работе в интерактивном режиме и в режиме экранного редактирования, генерируемом такими командами, как APPEND, EDIT и READ.

Синтаксис. SET STATUS ON/OFF (по умолчанию — ON)

Использование. Когда STATUS ON, статусная строка (22-я строка экрана) содержит информацию о текущем режиме (интерактивный, меню-управляемый, экранного редактирования и т. п.), об активном диске, имени активного отношения, текущем кортеже в нем и общем количестве кортежей, статусе редактирования вставка — замена — (Ins/), состоянии клавиши Num Lock на клавиатуре.

Когда STATUS OFF, некоторая информация выводится в нулевой строке в верхнем правом углу экрана в случае, если SCOREBOARD ON; если же последний режим OFF, то ничего не выводится.

SET STATUS OFF не имеет эффекта, если используется меню-управляемый монитор, такой, как ASSIST, SET или CREATE/MODIFY REPORT. Там статусная строка всегда выводится. Однако после возврата в программу или интерактивный режим STATUS принимает свое истинное значение.

Если @. . .SAY. . .GET. . .не использует статусную строку, то можно использовать SET STATUS ON совместно с SET FORMAT и READ.

Если STATUS ON, то при выполнении из программы команды типа APPEND или EDIT, инициирующей режим экранного редактирования, возникает статусная строка.

См. также SET MESSAGE TO, SET SCOREBOARD.

2.2.155. SET STEP (только dBASE III Plus)

Эта команда инициирует/отменяет режим покомандного исполнения программы. Она является мощным отладочным средством.

Синтаксис. SET STEP ON/OFF (по умолчанию — OFF).

Использование. Если установлен режим покомандного исполнения (SET STEP ON), то после выполнения каждой команды выводится ее результат и выдается сообщение.

Press SPACE to step, S to suspend, or Esc to cancel. . . (Нажмите «пробел» для следующего шага (выполнение очередной команды), S — для приостановления (ряд команд будет дан в интерактивном режиме, затем выполнение программы возобновится), Esc — для снятия выполнения программы и перехода в интерактивный режим).

См. также SET DEBUG, SET ECHO, SET TALK.

2.2.156. SET TALK (только dBASE III Plus)

Эта команда определяет, будет ли выводиться реакция dBASE III Plus на выполняемые команды.

Синтаксис. SET TALK ON/OFF (по умолчанию — ON)

Использование. Обычно в интерактивном режиме используется TALK ON. В этом режиме удобно, когда по каждой команде dBASE III Plus выводит результаты ее выполнения и некоторые вспомогательные сообщения. В программе, как правило, первой идет команда SET TALK OFF, при которой программа сама управляет видом экрана.

Режим TALK позволяет dBASE III Plus выводить информацию о количестве обработанных кортежей, значениях переменных, полученных в результате присвоения, о результатах команд, таких, как APPEND FROM, COPY, INDEX, PACK, SUM и др.

См. также SET DEBUG, SET ECHO.

2.2.157. SET TITLE (только dBASE III Plus)

Эта команда включает/выключает запрос на расширенное имя (пояснение) по каждому добавляемому в каталог файлу.

Синтаксис. SET TITLE ON/OFF (по умолчанию — ON).

Использование. Когда открыт каталог (он открывается в 10-й рабочей области — см. п. 1.5), при добавлении в него нового объекта (с точки зрения хранения это всегда файл, будь то отношение, индекс, формат или др.) dBASE III Plus просит ввести для этого объекта некоторый заголовок (расширенное имя до 80 символов), поясняющий назначение этого объекта.

В случае SET TITLE OFF этот режим подавляется, и если необходимо добавить в каталог пояснения по какому-либо объекту, то это можно сделать обычным редактированием каталога как отношения (SELECT 10, EDIT. . .).

См. также EDIT, SET CATALOG.

2.2.158. SET TYPEAHEAD (только dBASE III Plus)

Позволяет специфицировать размер буфера ввода.

Синтаксис. SET TYPEAHEAD TO ⟨expN⟩

Размер буфера по умолчанию — 20 символов. Допустимый диапазон значений — от 0 до 32K.

Использование. dBASE III Plus имеет специальный буфер ввода, куда последовательно складываются все символы и управляющие коды, вводимые пользователем с клавиатуры; затем содержимое этого буфера используется при интерпретации команд, требующих ввода.

Команда SET TYPEAHEAD работает только в случае SET ESCAPE ON. Для полного отказа от буферизации ввода нужно установить размер буфера 0. Это деактивирует ON KEY команду и INKEY( ) функцию.

Специальные случаи. Если исполняется командный файл, использующий ON ERROR обработку, целесообразно включение SET TYPEAHEAD TO 0 в качестве первой команды файла. Так как ошибка — результат непредвиденной ситуации, то подобный отказ от буферизации ввода будет хорошей мерой предосторожности.

При попытке ввести большее количество символов, чем позволяет размер буфера, все «лишние» символы введены не будут (потеряются), а в случае SET BELL ON при каждой попытке переполнения буфера будет выдаваться звуковой сигнал.

См. также CLEAR TYPEAHEAD, INKEY( ), ON, SET BELL, SET ESCAPE.

2.2.159. SET UNIQUE (dBASE III Plus и Clipper)

Определяет, будет ли включена в индексный файл информация о всех кортежах с одинаковым значением ключа индексации или только о первом (что делает индексный файл намного компактнее).

Синтаксис. SET UNIQUE ON/OFF (по умолчанию — OFF)

Использование. В случае создания нового индекса в режиме UNIQUE ON наличие нескольких кортежей с одинаковым значением ключа индексации приводит к тому, что только первый из них, обнаруженный dBASE III Plus, будет включен в формируемый индекс.

Команда SET UNIQUE ON эквивалентна команде INDEX ON ⟨key expression⟩ TO ⟨filename⟩ UNIQUE.

В случае реиндексации индексного файла, который был создан в режиме UNIQUE, индекс вновь будет сделан UNIQUE независимо от текущего режима UNIQUE — ON или OFF.

Добавление новых кортежей или редактирование существующих в отношении с подключенным компактным (unique) индексом сохраняет UNIQUE статус этого индекса. Это означает, что при добавлении кортежа (APPEND) с уже имеющимся значением ключа индексации этот кортеж не будет зарегистрирован в индексе, но добавится к отношению. Аналогично при редактировании кортежа (EDIT) и изменении при этом значения его ключа индексации на уже имеющееся в индексе значение исключается информация о нем из индексного файла (если она была там раньше с некоторым уникальным значением).

Замечание. Для восстановления полного индекса, включающего информацию по всем кортежам отношения вне зависимости от уникальности значений ключа индексирования для каждого из них, нужно просто перестроить индекс с помощью команд SET UNIQUE OFF, INDEX ON.

См. также FIND, INDEX, REINDEX, SET INDEX, USE.

2.2.160. SET VIEW (только dBASE III Plus)

Эта команда активизирует заданное внешнее представление (открывает .vue файл — см. п. 1.5).

Синтаксис. SET VIEW TO ⟨.vue filename⟩/?

В случае открытого каталога эта команда может обновить его, фиксируя в нем новое внешнее представление (если его там не было).

SET VIEW TO открывает .vue файл только временно и закрывает его автоматически. Для закрытия отношений, образующих внешнее представление, следует использовать команды CLEAR или CLOSE.

Использование. Можно либо непосредственно задать имя .vue файла, либо использовать опцию запроса к каталогу — ? в случае, если каталог активен.

Внешнее представление состоит из всех отношений, образующих виртуальное отношение, совместно с их индексами (для тех отношений, у которых они были использованы при конструировании внешнего представления), каждое отношение открывается в своей, определенной во внешнем представлении рабочей области; всех связей между отношениями; номера рабочей области, которая должна стать текущей при активизации внешнего представления; списка атрибутов виртуального отношения; условия фильтрации; форматного файла (формата), если он определен для этого внешнего представления.

Все это открывается, устанавливается и активизируется по команде SET VIEW TO. Таким образом, одна эта команда заменяет порой очень значительную последовательность повторяющихся команд: USE, SELECT, SET INDEX, SET FIELDS TO, SET FIELDS ON, SET RELATION, SET FILTER, SET FORMAT. Это очень эффективное средство работы с внешними представлениями.

Для изменения содержимого внешнего представления или для создания нового следует использовать команду CREATE/MODIFY VIEW.

Clipper не имеет подобных средств. Конструирование внешних представлений там осуществляется программно.

См. также CREATE VIEW FROM, CREATE/MODIFY VIEW, SET FIELDS, SET FIELDS TO, SET FILTER, SET FORMAT, SET RELATION.

2.2.161. SKIP (dBASE III Plus и Clipper)

Осуществляет относительное перемещение указателя текущего кортежа (для dBASE III Plus — только в активном отношении, для Clipper — в любом из открытых).

Синтаксис. Для dBASE III Plus:

SKIP [⟨expN⟩]

Для Clipper:

SKIP [⟨expN⟩] [ALIAS ⟨selection⟩]

Использование. Если активен индексный механизм доступа, то SKIP осуществляет перемещение указателя текущего кортежа на указанное число кортежей вперед или назад в последовательности, задаваемой индексом. Если же индекс не используется, то перемещение указателя текущего кортежа осуществляется в соответствии с физическим порядком расположения кортежей в отношении.

Указатель смещается вперед, если не задан знак — в выражении.

Если выражение опущено, то по умолчанию осуществляется сдвиг вперед на один кортеж, т. е. выполняется операция «следующий кортеж». Если SKIP используется, когда указатель указывает на последний кортеж в отношении, то значение функции RECNO( ) становится на единицу больше номера последнего кортежа в отношении, а функция EOF( ) принимает значение .T. (истина). Если SKIP-1 используется, когда указатель стоит на первом кортеже отношения, то RECNO( ) остается равным 1, но BOF( ) принимает значение .T. (истина).

Clipper позволяет осуществлять перемещение указателя не только в активном, но и в любом открытом отношении.

См. также BOF( ), EOF ( ), RECNO ( ).

2.2.162. SORT (dBASE III Plus и Clipper)

Команда SORT создает новое отношение, куда копирует кортежи активного отношения, удовлетворяющие условию, и осуществляет упорядочивание кортежей в новом отношении в алфавитном, хронологическом или числовом порядке в соответствии с заданными атрибутами сортировки (имеется в виду физическое упорядочивание). При этом может осуществляться вложенное упорядочение, т. е. разбиение групп на подгруппы и т. д.

Синтаксис. SORT ⟨scope⟩ TO [⟨newfile⟩] ON ⟨field1⟩ [/A] [/C] [/D]

[, ⟨field2⟩ [/A] [/C] [/D] . . .]

[WHILE [⟨condition⟩] [FOR ⟨condition⟩]

TO отношение имеет расширение .dbf (если иное не задано явно).

Сортировка выполняется в порядке возрастания (/A), если иное не задано (имеется в виду возрастание в соответствии с кодовой таблицей ASCII).

Для dBASE III Plus, если открыт каталог, SORT обновляет его.

Использование. Нельзя назначать сортировку по логическим или текстовым атрибутам (memo).

SORT не работает с подстроками и другими выражениями (только по значениям атрибутов). При необходимости использовать некоторый сложный ключ следует использовать индексирование (см. INDEX).

Можно сортировать максимум по десяти атрибутам одновременно (т. е. до девяти вложений).

Отношение не может быть отсортировано само в себя или в любое из других открытых отношений.

Опции. Ключевые слова ASCENDING (по возрастанию) и DESCENDING (по убыванию) являются альтернативами для /A и /D. При опции /C не учитываются различия между прописными и строчными буквами при сортировке по данному атрибуту.

Не следует использовать буквы от A до J при задании имени результирующего отношения, так как эти буквы зарезервированы в качестве альтернативных имен по умолчанию. Например, AA — допустимое имя, а A — нет.

Можно комбинировать /C с одним из /A или /D. Когда используется такая комбинация, можно задавать просто /DC (или/ AC).

При сортировке по многим ключам наиболее значащий должен задаваться первым и т. д.

При сортировке по многим ключам вначале результирующее отношение упорядочивается по наиболее значащему ключу, затем в образовавшихся группах (интервалах постоянства наиболее значащего ключа) производится упорядочение по следующему ключу— разбиение групп на подгруппы, затем в подгруппах — следующее упорядочение и т. д.

См. также INDEX.

2.2.163. STORE (dBASE III Plus и Clipper)

Создает и инициализирует (присваивает тип и значение) одну или более переменных памяти.

Синтаксис. STORE ⟨expression⟩ TO ⟨memory variable list⟩

Альтернативный синтаксис для STORE (чаще используемый):

⟨memory variable⟩ = ⟨expression⟩

Использование. Если переменная памяти существовала ранее, то она переписывается (при этом, возможно, изменяя свой тип).

Обычно для одиночных присвоений используется альтернативный синтаксис=. Основной синтаксис (STORE. . .) используется при групповых присвоениях: одно выражение соответствует списку переменных.

При создании переменной посредством альтернативного синтаксиса нельзя использовать наименование команды dBASE III Plus (или Clipper) в качестве имени, иначе система неправильно проинтерпретирует команду. В dBASE III Plus/Clipper, вообще говоря, нет зарезервированных имен, все понимается по контексту. Используя основной синтаксис команды, можно присвоить некоторой переменной имя какой-либо команды dBASE III Plus/Clipper.

Числовые переменные могут быть добавлены к себе (a = a+ a), для символьных допустима операция конкатенации с собой (b = «string» + b + b).

Максимальное число активных переменных для dBASE III Plus — 256, и максимум 6000 байт может быть выделено под хранение их значений, если это число не изменено в Config. db.

Для Clipper максимальное число активных переменных — 2048, причем массив считается одной переменной, а он может иметь до 2048 элементов. Ограничением на объем памяти, выделяемой под значения переменных, является лишь физический объем памяти компьютера.

Специальные случаи. Если имя атрибута совпадает с именем переменной, то атрибут имеет больший приоритет во всех операциях (где такая неоднозначность допустима). Для точного указания переменной (а не атрибута) нужно использовать конструкцию m — ⟩ ⟨memvar⟩.

2.2.164. SUM (dBASE III Plus и Clipper)

Эта команда позволяет суммировать значения числовых атрибутов (или некоторых определенных на них выражений) для кортежей активного отношения, удовлетворяющих условию выбора, заданному в команде, получая, таким образом, некоторые итоговые (интегральные) результаты.

Синтаксис. Для dBASE III Plus:

SUM [⟨scope⟩] [⟨expression list⟩]
TO [⟨memvar list⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]

Для Clipper:

SUM [⟨scope⟩] ⟨field list⟩ TO ⟨memvar list⟩
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]

Если иное не задано опциями ⟨scope⟩, FOR. . . , WHILE. . . , суммирование проводится по всем кортежам отношения.

Clipper запрещает умолчание списка суммируемых атрибутов и списка переменных памяти. Это диктуется требованиями компилятора.

Если (в случае dBASE III Plus) список выражений опущен, то суммирование производится по всем числовым атрибутам, если список соответствующих переменных для помещения результатов также опущен, то в режиме TALK ON значения просто выводятся на экран.

См. также AVERAGE.

2.2.165. SUSPEND (только dBASE III Plus)

SUSPEND — это в большинстве случаев отладочное средство, позволяющее оператору приостановить программу в процессе ее исполнения. Затем оператор может ввести ряд команд с клавиатуры либо из архива (HISTORY — см. п. 2.12), а затем продолжить выполнение командой RESUME.

Синтаксис. SUSPEND

Использование. При нажатии клавиши Esc во время выполнения программы dBASE III Plus (в режиме SET ESCAPE ON) прервет исполнение программы и выдаст сообщение-запрос, предоставляющее выбор одной из трех возможностей:

Called from ⟨filename .prg⟩ What now — Cancel, Ignore, Suspend? (C, I, or S)

Можно выбрать Cancel (прекратить выполнение программы), Ignore (продолжить выполнение) либо Suspend (приостановить выполнение, выполнить несколько команд в интерактивном режиме, а затем, возможно, продолжить — по команде RESUME).

При ответе S (SUSPEND) можно изменить командные строки из архива и затем выполнить их и/или выполнить некоторую произвольную команду (команды). Нельзя изменять прерванный программный файл или другие открытые программные файлы: при попытке сделать это dBASE III Plus ответит, что файл уже открыт (File is already open). По окончании обработки прерывания нужно дать команду RESUME, и dBASE III Plus вернется к интерпретации программы с места прерывания.

Можно также прервать исполнение путем включения непосредственно команды SUSPEND в нужное место (места) в программе.

Если созданы какие-либо переменные памяти в процессе обработки прерывания, то они получат статус локальных (PRIVATE) на том уровне вложения, на котором программа была прервана.

Команда CANCEL прекращает (снимает) все DO файлы (подпрограммы), включая и те, которые приостановлены (по SUSPEND). Необходимо, однако, закрыть процедурный файл (если он открыт) командой CLOSE PROCEDURE.

См. также CANCEL, DISPLAY HISTORY, SET ESCAPE, SET STEP, RESUME, RETURN.

2.2.166. TEXT...ENDTEXT (dBASE III Plus и Clipper)

TEXT используется в программе для вывода текстового фрагмента (блока, кадра) на экран или принтер. Команда дает простой и удобный способ осуществления такого вывода. Clipper, кроме того, позволяет осуществить вывод в определенный файл.

Синтаксис. Для dBASE III Plus:

```
TEXT
⟨text characters. . .⟩
ENDTEXT
```

Для Clipper:

```
TEXT [TO PRINT/TO FILE ⟨filename⟩]
⟨text characters. . .⟩
ENDTEXT
```

Использование. Текст выводится точно так, как он внесен в программу.

Макрофункция (&) не распознается внутри текста.

2.2.167. TOTAL (dBASE III Plus и Clipper)

Эта команда создает отношение, каждый кортеж которого содержит в качестве значений своих числовых атрибутов суммарные значения по всем кортежам активного отношения, имеющим одинаковое значение заданного в команде ключевого атрибута.

Синтаксис. TOTAL ON ⟨key field⟩ TO ⟨filename⟩
[⟨scope⟩] [FIELDS ⟨field list⟩]
[WHILE ⟨condition⟩] [FOR ⟨condition⟩]

В имени ТО отношения следует задать код диска, если он отличен от активного. По умолчанию подразумевается расширение .dbf.

Если иное не задано опциями ⟨scope⟩, FOR. . . , WHILE. . . , то все кортежи активного отношения подвергаются обработке (анализу) данной командой. Если открыт каталог, то создаваемое ТО отношение добавляется к нему.

Использование. Активное отношение должно быть проиндексировано (индекс должен быть активным) или отсортировано по значениям ключевого атрибута.

Если ТО отношение существует, TOTAL не перезаписывает его (не создает нового) без запроса при SET SAFETY ON.

Всем кортежам активного отношения с одинаковым значением ключевого атрибута соответствует один кортеж ТО отношения. Все его числовые атрибуты, объявленные в параметре FIELDS, будут иметь суммарные значения. Все остальные его атрибуты (структура ТО отношения соответствует структуре активного) будут иметь значения первого из кортежей активного отношения с данным значением ключевого атрибута.

Специальные случаи. Не следует использовать одиночные буквы от A до J и M в качестве имени ТО отношения, так как они зарезервированы в качестве альтернативных имен (ALIAS). Например, AA — значащее имя, A — нет.

Если длины атрибута не хватает, чтобы вместить суммарное значение, то dBASE III Plus размещает там звездочки «*» вместо значения, индицируя переполнение. Избежать этого можно, выполнив команду MODIFY STRUCTURE для активного отношения с целью увеличения длины требуемых числовых атрибутов, а затем повторив TOTAL.

Структура ТО отношения идентична структуре активного отношения, за исключением memo-атрибутов, которые не копируются в ТО отношение (опускаются).

2.2.168. TYPE (dBASE III Plus и Clipper)

Эта команда служит для вывода содержимого текстового файла.

Синтаксис. Для dBASE III Plus:

```
TYPE ⟨filename⟩ [TO PRINT]
```

Для Clipper:

```
TYPE ⟨filename⟩ [TO PRINT]
     [TO FILE ⟨filename⟩]
```

Имя файла должно включать расширение, а если файл не на активном диске,— то и код диска.

Использование. TYPE осуществляет вывод только ASCII файлов. Отношения, индексы, .dbt файлы и др. не могут быть выведены с помощью этой команды.

TYPE не может выводить открытые файлы. Другими словами, программа не может включать инструкцию «распечатать саму себя».

2.2.169. UPDATE (dBASE III Plus и Clipper)

Позволяет комплексную модификацию информации в одном отношении на основе данных другого отношения.

Команда UPDATE использует данные из некоторого открытого отношения в качестве базиса для модификации кортежей активного отношения. Модификация происходит при совпадении ключевого условия в кортежах обоих отношений.

Синтаксис. UPDATE ON ⟨key field⟩ FROM ⟨Alias⟩
REPLACE ⟨field1⟩ WITH ⟨exp2⟩
[⟨field2⟩ WITH ⟨exp2⟩ . . .]
[RANDOM]

Использование. Модифицируемое отношение должно быть активным (в текущей рабочей области), FROM отношение должно быть открыто в какой-либо другой рабочей области.

Ключевой атрибут (атрибут, по значениям которого устанавливается связь между отношениями) должен иметь одинаковое имя в обоих отношениях.

Если ключевой атрибут в модифицируемом отношении содержит неуникальные значения, то только первый кортеж (среди кортежей с совпадающими значениями ключа) подвергнется модификации.

Опции. Оба отношения должны быть отсортированы или проиндексированы по ключевому атрибуту, если не задана опция RANDOM (произвольный). Эта опция требует, чтобы обновляемое (активное) отношение было проиндексировано по ключевому атрибуту, а кортежи FROM отношения могут идти в произвольном порядке.

Если REPLACE выражение включает атрибут из FROM отношения (обычно так и бывает), то этот атрибут следует указывать расширенным именем: ⟨alias⟩ —⟩ ⟨field⟩.

2.2.170. USE (dBASE III Plus и Clipper)

USE — центральная команда dBASE III Plus/Clipper. Она активизирует указанное отношение (открывает его в текущей рабочей области) и может дополнительно открыть до семи индексных файлов в текущей рабочей области. Если отношение содержит атрибуты, то соответствующий .dbt файл открывается автоматически.

Синтаксис. Для dBASE III Plus:
USE [⟨.dbf filename⟩/?]
[INDEX ⟨.ndx file list⟩]
[ALIAS ⟨alias name⟩]
Для Clipper:
USE [.⟨dbf filename⟩]
[INDEX ⟨.ntx file list⟩]
[ALIAS ⟨alias name⟩]

USE без параметров закрывает активное отношение и индексные файлы в текущей рабочей области.

Если иного не задано, то подразумевается .dbf расширение в имени файла, .ndx расширение для индексных файлов dBASE III Plus и .ntx — для индексных файлов Clipper.

Если открыт каталог (в случае dBASE III Plus) и активизируемое отношение в нем еще не зарегистрировано, то оно добавляется в каталог.

Использование. Опция ? (запрос к каталогу) позволяет (в случае dBASE III Plus) получить список всех зарегистрированных в каталоге отношений в качестве меню и выбрать из него требуемое отношение. Затем можно получить меню индексов, если для выбранного отношения они зафиксированы в каталоге.

Если альтернативное имя (ALIAS) не указывается, то оно считается именем отношения или буквой от A до J в соответствии с номером рабочей области.

Указатель текущего кортежа. Если отношение активизируется без индексов, то указатель текущего кортежа позиционируется на

первый кортеж в отношении. Если же отношение активизируется с открытием одного или более индексов, то указатель позиционируется на логически первый кортеж в соответствии с индексным механизмом доступа. В качестве индекса берется первый в списке индекс (он называется главным, или мастер-индексом). Остальные индексы в списке автоматически корректируются при модификации отношения, и любой из них может быть сделан мастер-индексом в любой момент с помощью команды SET INDEX, которая также позволяет сменить список открытых индексов.

См. также CLOSE, INDEX, SELECT, SELECT(), SET INDEX, SET VIEW.

2.2.171. WAIT (dBASE III Plus и Clipper)

Команда WAIT приостанавливает программу, ожидая от пользователя нажатия любого ключа на клавиатуре. Затем выполнение возобновляется.

Синтаксис. WAIT [⟨prompt⟩] [TO ⟨memvar⟩]

Если введен Enter либо другой невизуализируемый символ, то значение переменной, которая может указываться в WAIT, — ноль (ASCII 0).

Использование. ⟨prompt⟩ (сообщение) может быть символьной переменной или литеральной строкой. Если ⟨prompt⟩ не задается, то по умолчанию выдается сообщение: «Press any key to continue...» («Нажмите любую клавишу для продолжения»).

Опции. Если используется опция TO ⟨memvar⟩, то код нажатой клавиши запоминается в создаваемую символьную переменную с указанным именем.

Специальные случаи. Если (в dBASE III Plus) используется конструкция ON KEY WAIT TO ⟨memvar⟩, то нажатая клавиша, которая возбудила ON ситуацию, попадает в указанную переменную, не ожидая от пользователя нового нажатия клавиши.

См. также ON KEY.

2.2.172. ZAP (dBASE III Plus и Clipper)

Эта команда удаляет все кортежи из активного отношения.

Синтаксис. ZAP

Использование. ZAP эквивалентна паре команд DELETE ALL, PACK, но выполняется значительно быстрее.

Если SAFETY ON, то dBASE III Plus запросит подтверждение этой операции перед удалением: ZAP ⟨filename⟩? (Y/N).

Любой открытый в текущей рабочей области индекс автоматически очищается.

dBASE III Plus/Clipper возвращает в операционную систему всю освободившуюся от удаления кортежей, значений индексов и memo-атрибутов внешнюю память.

См. также DELETE, PACK, SET SAFETY.

Глава 3

ФУНКЦИИ dBASE III PLUS, CLIPPER

3.1. Использование

Функции предназначены для конструирования выражений и соответственно расширения возможностей языка dBASE III Plus/ Clipper.

Функции возвращают значения, являющиеся результатом их работы, поэтому употребление функций допустимо внутри выражений, когда необходимо использовать возвращаемое функцией значение. Тип возвращаемого значения должен быть уместен в данном контексте (т. е., например, если результат функции используется в качестве одного из операндов некоторой арифметической операции, то функция должна возвращать числовое значение).

Результат некоторой функции может быть использован в качестве аргумента другой функции, т. е. функции могут вкладываться одна в другую в соответствии с алгоритмом получения требуемого результата. Например: CMONTH (DATE ( )) — текущий месяц. Вначале вычисляется самая внутренняя функция, затем следующая — та, у которой результат вычисленной является одним из аргументов, и т. д.

dBASE III Plus содержит около 70 функций. Clipper имеет средства, позволяющие разработчику самому конструировать новые функции, пополняя их состав в соответствии со спецификой создаваемых приложений. Разработка функций может вестись в Clipper как на языке dBASE, так и на языках программирования СИ, Ассемблер. В составе поставки Clipper часть функций (44) описана в основной документации и входит в библиотеку Clipper .lib, вторая часть представлена в описании основных отличий версии Winter' 85 (рабочая версия) в файле Read _ me .lst (14) и также содержится в библиотеке Clipper .lib. Кроме того, 27 функций приводятся в исходных текстах на языке dBASE, 6 — на языке СИ и 2 — на языке Ассемблер (файлы: Extend .doc, Extenddb .prg, Extend. h, Extende. c, Extenda. asm). Таким образом, набор функций Clipper включает в себя практически все функции dBASE III Plus (кроме двух: ERROR и MESSAGE), а также много других функций и может быть пополнен при необходимости.

Общее число рассматриваемых в настоящем описании функций — 96.

3.2. Классификация

Функции dBASE III Plus / Clipper могут быть разбиты на следующие восемь классов (функционально): & — макроподстановка; Д — функции даты и времени; С — символьные (строчные) манипуляции; М — математические функции; П — функции, осуществляющие преобразования типов; Т — тестовые функции (проверки различного рода); И — идентификации объектов; В — ввода.

Ниже перечислены функции каждого класса.

В последующих таблицах указание * в качестве типа параметра обозначает отсутствие параметра.

3.2.1. & — макроподстановка

Имя функции	Тип параметра	Тип результата	dBASE	Clipper: основной	Clipper: W&B 5	Clipper: dBase	Clipper: C	Clipper: Asm	Отличия	Описание
&	C	C, D, N, L	+	+					+	Макроподстановка

3.2.2. Д — функции даты и времени

Имя функции	Тип параметра	Тип результата	dBASE	Clipper: основной	Clipper: W&B 5	Clipper: dBase	Clipper: C	Clipper: Asm	Отличия	Описание
CDOW	D	C	+	+						День недели
CMONTH	D	C	+	+						Календарный месяц
DATE	*	D	+	+						Системная дата
DAY	D	N	+	+						День месяца
DOW	D	N	+	+						День недели
ELAPTIME	N, N	C	+	+		+				Количество секунд в заданном интервале времени
MONTH	D	N	+	+	+					Месяц года
SECONDS	*	N	+	+						Системное время в виде количества прошедших с полночи секунд
TIME	*	C	+	+						Системное время
YEAR	D	N	+	+						Год

3.2.3. C — символьные (строчные) манипуляции

Имя функции	Тип		dBASE	Clipper					Отлич	Описание
	параметра	результата		осн	W85	dB	C	A		
ALLTRIM	C	C	—			+				Удаление ведущих и хвостовых пробелов в строке
AMPM	C	C	—			+				Время в американском формате (a.m./p.m.)
AT	C	C	+	+						Выделение подстроки в строке
LEFT	C	C	+			+				Выбор подстроки от левого конца строки
LTRIM	C	C	+		+					Удаление ведущих пробелов
MEMOEDIT	M	M	—		+					Редактирование переменных или значений атрибутов типа memo
REPLICATE	C	C	+	+						Повторитель символьного выражения
RIGHT	C	C	+			+				Выбор подстроки от правого конца строки
RTRIM	C	C	+	—						Удаление хвостовых пробелов
SPACE	N	C	+	+						Генерация строки пробелов заданной длины
STUFF	C	C	+			+				Замена части строки
SUBSTR	C	C	+	+					+	Выделение подстроки в строке
TRANSFORM	C, N	C	+		+					Символьная строка или число в формате, определяемом PICTURE шаблоном
TRIM	C	C	+	+						Удаление хвостовых пробелов

3.2.4. M — математические функции

Имя функции	Тип		dBASE	Clipper					Отлич	Описание
	параметра	результата		осн	W85	dB	C	A		
ABS	N	N	+			+				Абсолютное значение (модуль)
EXP	N	N	+	+						Экспонента
INT	N	N	+	+						Преобразование в целое
LOG	N	N	+	+						Логарифм
MAX	N	N	+			+				Определение большего из двух чисел
MIN	N	N	+			+				Определение меньшего из двух чисел
MOD	N	N	+			+				Вычисление остатка от деления по модулю
ROUND	N	N	+	+						Округление чисел
SQRT	N	N	+	+						Извлечение квадратного корня

3.2.5. П — функции, осуществляющие преобразования типов

Имя функции	Тип параметра	Тип результата	dBASE	Clipper: осн	Clipper: W85	Clipper: dB	Clipper: C	Clipper: A	Отлич	Описание
AMPM	C	C	–			+				Время в американском формате (a.m./p.m.)
ASC	C	N	+	+						Получение ASCII кода символа
CHR	N	C	+	+						Получение символа по ASCII коду
CTOD	C	D	+	+						Преобразование символьной переменной в дату
DAYS	N	N	–			+				Преобразование секунд в целое число дней
DTOC	D	C	+	+						Преобразование даты в символьную строку: мм/дд/гг
DTOS	D	C	–		+					Преобразование даты в символьную строку: ггггммдд
LOWER	C	C	+	+						Преобразование прописных латинских букв в строчные
SECS	C	N	–			+				Преобразование строки времени в число секунд
SOUNDEX	N	C	–			+				Преобразование числа к «научной» форме записи
STR	N	C	+	+						Преобразование числа в строку
STRZERO	N	C	–			+				Преобразование числа в строку заданной длины с ведущими нулями вместо пробелов
TSTRING	N	C	–			+				Преобразование числа секунд в строку времени
UPPER	C	C	+	+						Преобразование строчных латинских букв в прописные
VAL	C	N	+	+						Преобразование символьной строки в число
WORD	N	N	–		+					Преобразование чисел при передаче в качестве параметров к короткой форме

3.2.6. Т — тестовые функции (проверки различного рода)

Имя функции	Тип параметра	Тип результата	dBASE	Clipper: осн	Clipper: W85	Clipper: dB	Clipper: C	Clipper: A	Отлич	Описание
BOF	*	L	+	+						Анализ признака начала файла
DELETED	*	L	+	+						Анализ признака удаления для текущего кортежа
EMPTY	N/C/D/L	L	–		+					Проверка на нулевое значение
EOF	*	L	+	+						Анализ признака конца файла
ERROR	*	N	+	–						Номер ошибки dBASE III Plus
FILE	C	L	+	+						Проверка существования файла
FOUND	*	L	+		+					Результат последней операции поиска в отношении
IF	C, D, N	C, D, N	–	+						Выбор выражения по условию
IIF	C, D, N	C, D, N	+			+				Выбор выражения по условию
ISALPHA	C	L	+			+				Проверка на букву
ISCOLOR	*	L	+					+		Проверка наличия цветного дисплея
ISLOWER	C	L	+			+				Проверка на строчную букву
ISPRINTER	*	L	–					+		Проверка готовности принтера
ISUPPER	C	L	+			+				Проверка на прописную букву
UPDATED	*	L	–		+					Показывает: были ли изменены данные в процессе экранного редактирования

3.2.7. И — идентификация объектов

Имя функции	Тип параметра	Тип результата	dBASE	Clipper осн	Clipper W85	Clipper dB	Clipper C	Clipper A	Отлич	Описание
COL	*	N	+	+						Определение номера столбца текущего положения экранного курсора
DBF	*	C	+			+			+	Имя активного отношения
DISKSPACE	*	N	+				+			Свободное пространство на диске (в байтах)
FIELD	N	C	+	−						Имя атрибута в активном отношении
FIELDNAME	N	C	−	+						Имя атрибута в активном отношении
FKLABEL	N	C	+			+			+	Имена функциональных клавиш
FKMAX	*	N	+			+			+	Максимальный номер функциональной клавиши
GETE	C	C	−				+			Значение переменных операционного окружения
GETENV	C	C	+	−						Значение переменных операционного окружения
HEADER	*	N	−							Размер заголовка в отношении
LASTREC	*	N	−	+						Число кортежей в активном отношении
LEN	C	N	+	+						Длина символьной строки
LENNUM	N	N	−			+				Длина числа
LUPDATE	*	D	+				+			Дата последнего обновления отношения
MESSAGE	*	C	+	−						Текст ошибочного сообщения dBASE III Plus
NDX	N	C	+			+			+	Имена открытых индексных файлов
OS	*	C	+			+				Наименование операционной системы
PCOL	*	N	+	+						Определение текущего столбца на устройстве печати
PROCLINE	*	N	−	+						Возвращает номер строки исходного текста исполняемой программы или процедуры
PROCNAME	*	C	−	+						Имя исполняемой программы или процедуры
PROW	*	N	+	+						Определение текущей строки на устройстве печати
READVAR	*	C	−		+					Возвращает имя текущей GET/MENU переменной или пустую строку
RECCOUNT	*	N	+	−						Число кортежей в активном отношении
RECNO	*	N	+	+						Номер текущего кортежа
RECSIZE	*	N	+	−						Размер (длина) кортежа
ROW	*	N	+	+						Определение номера строки текущего положения экранного курсора
SELECT	*	N	−		+					Номер выбранной (активной) рабочей области
TYPE	C	C	+	+						Определяет тип выражения
VERSION	*	C	+			+				Версия dBASE III Plus/Clipper

3.2.8. B — ввод

Имя функции	Тип параметра	Тип результата	dBASE	Clipper осн	Clipper W85	Clipper dB	Clipper C	Clipper A	Отлич	Описание
INKEY	N	N	+		+				+	Код нажатой клавиши
LASTKEY	*	N	—	+						Код последней нажатой клавиши
READKEY	*	N	+			+				Возвращает код клавиши, нажатой для выхода из режима экранного редактирования, а также показывает, были ли сделаны изменения в ходе редактирования

3.3. Типы параметров и возвращаемых значений

Ниже приведен полный список функций с параметрами и возвращаемыми значениями в алфавитном порядке.

Имя функции	Тип параметра	Тип результата	Класс	dBASE	Clipper осн	Clipper W85	Clipper dB	Clipper C	Clipper A	Отлич	Описание
&	C	C, D, N, L	&	+	+					+	Макроподстановка
ABS	N	N	M	+			+				Абсолютное значение (модуль)
ALLTRIM	C	C	C	—			+				Удаление ведущих и хвостовых пробелов в строке
AMPM	C	C	П	—			+				Время в американском формате (a.m./p.m.)
ASC	C	N	П	+	+						Получение ASCII кода символа
AT	C	C	С	+	+						Выделение подстроки в строке
BOF	*	L	Т	+	+						Анализ признака начала файла
CDOW	D	C	Д	+	+						День недели
CHR	N	C	П	+	+						Получение символа по ASCII коду
CMONTH	D	C	Д	+	+						Календарный месяц
COL	*	N	И	+	+						Определение номера столбца текущего положения экранного курсора
CTOD	C	D	П	+	+						Преобразование символьной переменной в дату
DATE	*	D	Д	+	+						Системная дата
DAY	D	N	Д	+	+						День месяца
DAYS	N	N	П	—			+				Преобразование секунд в целое число дней
DBF	*	C	И	+			+			+	Имя активного отношения
DELETED	*	L	Т	+	+						Анализ признака удаления для текущего кортежа
DISKSPACE	*	N	И	+				+			Свободное пространство на диске (в байтах)
DOW	D	N	Д	+	+						День недели
DTOC	D	C	П	+	+						Преобразование даты в символьную строку: мм/дд/гг
DTOS	D	C	П	—		+					Преобразование даты в символьную строку: ггггммдд
ELAPTIME	N, N	C	Д	—			+				Возвращает символьную строку (количество секунд в заданном интервале времени)

Продолжение

Имя функции	Тип параметра	Тип результата	Класс	dBASE	Clipper: осн	Clipper: W&B	Clipper: dB	Clipper: C	Clipper: A	Отлич	Описание
EMPTY	N/C/D/L	L	Т	–		+					Проверка на нулевое значение
EOF	*	L	Т	+	+						Анализ признака конца файла
ERROR	*	N	Т	+	–						Номер ошибки dBASE III Plus
EXP	N	N	М	+	+						Экспонента
FIELD	N	C	И	+	–						Имя атрибута в активном отношении
FIELDNAME	N	C	И	–	+						Имя атрибута в активном отношении
FILE	C	L	Т	+	+						Проверка существования файла
FKLABEL	N	C	И	+			+			+	Имена функциональных клавиш
FKMAX	*	N	И	+			+			+	Максимальный номер функциональной клавиши
FOUND	*	L	Т	+		+					Результат последней операции поиска в отношении
GETE	C	C	И	–				+			Значение переменных операционного окружения
GETENV	C	C	И	+	–						Значение переменных операционного окружения
HEADER	*	N	И	–							Размер заголовка в отношении
IF	C, D, N	C, D, N	Т	–	+						Выбор выражения по условию
IIF	C, D, N	C, D, N	Т	+							Выбор выражения по условию
INKEY	N	N	В	+			+				Код нажатой клавиши
INT	N	N	М	+		+				+	Преобразование в целое
ISALPHA	C	L	Т	+	+						Проверка на букву
ISCOLOR	*	L	Т	+			+				Проверка наличия цветного дисплея
ISLOWER	C	L	Т	+					+		Проверка на строчную букву
ISPRINTER	*	L	Т	–			+				Проверка готовности принтера
ISUPPER	C	L	Т	+					+		Проверка на прописную букву
LASTKEY	*	N	В	–	+		+				Код последней нажатой клавиши
LASTREC	*	N	И	–	+						Число кортежей в активном отношении
LEN	C	N	И	+	+						Длина символьной строки
LENNUM	N	N	И	–			+				Длина числа
LOG	N	N	М	+	+						Логарифм
LOWER	C	C	П	+	+						Преобразование прописных латинских букв в строчные
LTRIM	C	C	С	+		+					Удаление ведущих пробелов
LUPDATE	*	D	И	+				+			Дата последнего обновления отношения
MAX	N	N	М	+			+				Определение большего из двух чисел
MEMOEDIT	M	M	С	–		+					Редактирование переменных или значений атрибутов типа memo
MESSAGE	*	C	И	+	–						Текст ошибочного сообщения dBASE III Plus
MIN	N	N	М	+			+				Определение меньшего из двух чисел
MOD	N	N	М	+			+				Вычисление остатка от деления по модулю

Продолжение

Имя функции	Тип параметра	Тип результата	Класс	dBASE	Clipper осн	Clipper W85	Clipper dB	Clipper C	Clipper A	Отлич	Описание
MONTH	D	N	Д	+	+						Месяц года
NDX	N	C	И	+			+			+	Имена открытых индексных файлов
OS	*	C	И	+			+				Наименование операционной системы
PCOL	*	N	И	+	+						Определение текущего столбца на устройстве печати
PROCLINE	*	N	И	–	+						Возвращает номер строки исходного текста исполняемой программы или процедуры
PROCNAME	*	C	И	–	+						Имя исполняемой программы или процедуры
PROW	*	N	И	+	+						Определение текущей строки на устройстве печати
READKEY	*	N	В	+			+				Определяет код клавиши, нажатой для выхода из режима экранного редактирования, а также были ли сделаны изменения в ходе редактирования
READVAR	*	C	И	–		+					Возвращает имя текущей GET/MENU переменной или пустую строку
RECCOUNT	*	N	И	+	–						Число кортежей в активном отношении
RECNO	*	N	И	+	+						Номер текущего кортежа
RECSIZE	*	N	И	+	–						Размер (длина) кортежа
REPLICATE	C	C	С	+	+						Повторитель символьного выражения
RIGHT	C	C	С	+			+				Выбор подстроки от правого конца строки
ROUND	N	N	М	+	+						Округление чисел
ROW	*	N	И	+	+						Определение номера строки текущего положения экранного курсора
RTRIM	C	C	С	+	–						Удаление хвостовых пробелов
SECONDS	*	N	Д	–		+					Системное время в виде количества прошедших с полночи секунд
SECS	C	N	П	–			+				Преобразование строки времени в число секунд
SELECT	*	N	И	–		+					Номер выбранной (активной) рабочей области
SOUNDEX	N	C	П	–			+				Преобразование числа к «научной» форме записи
SPACE	N	C	С	+	+						Генерирует строку пробелов заданной длины
SQRT	N	N	М	+	+						Извлечение квадратного корня
STR	N	C	П	+	+						Преобразование числа в строку
STRZERO	N	C	П	–			+				Преобразование числа в строку заданной длины с ведущими нулями вместо пробелов
STUFF	C	C	С	+			+				Замена части строки
SUBSTR	C	C	С	+	+					+	Выделение подстроки в строке
TIME	*	C	Д	+	+						Системное время

Продолжение

Имя функции	Тип параметра	Тип результата	Класс	dBASE	Clipper: осн	Clipper: WB5	Clipper: dB	Clipper: C	Clipper: A	Отлич	Описание
TRANSFORM	C, N	C	C	+		+					Символьная строка или число в формате, определяемом PICTURE шаблоном
TRIM	C	C	C	+	+						Удаление хвостовых пробелов
TSTRING	N	C	П	–			+				Преобразование числа секунд в строку времени
TYPE	C	C	И	+–	+	+					Определяет тип выражения
UPDATED	*	L	Т								Показывает, были ли изменены данные в процессе экранного редактирования
UPPER	C	C	П	+	+						Преобразование строчных латинских букв в прописные
VAL	C	N	П	+	+						Преобразование символьной строки в число
VERSION	*	C	И	+			+				Версия dBASE III Plus/Clipper
WORD	N	N	П	–		+					Преобразование чисел при передаче в качестве параметров к короткой форме
YEAR	D	N	Д	+	+						Год
Итого				7 0	4 5	1 1	2 7	6	2		

3.4. Список функций

3.4.1. & — макрофункция (dBASE III Plus и Clipper)

Класс: &

Макроподстановка не является функцией в истинном смысле и в этом разделе приводится для удобства. Она осуществляет подстановку содержимого переменной памяти в качестве части командной строки dBASE III Plus/Clipper. dBASE III Plus позволяет осуществлять макроподстановку всей команды в целом или произвольной ее части. Clipper накладывает ряд ограничений на аппарат макроподстановки в силу особенностей компиляции.

Эта функция может использоваться только с символьными переменными.

Примеры.

```
x = "a = 5"
&x          && справедливо только для dBASE III Plus
? a
  5
```

Пусть необходимо произвести в отношении поиск по индексу, причем ключ индексирования задается в переменной памяти:

```
mname = "Lotus"
USE Clients INDEX Cnames
FIND & mname
```

dBASE III Plus обычно рассматривает символьную последовательность в команде FIND как заданную литерально. Если бы не была использована & функция, то поиск производился бы по ключу «mname», а не «Lotus».

В dBASE III Plus результатом макроподстановки может быть целая команда либо некоторая (произвольная) ее часть. При программировании в среде Clipper компилятор должен явно «видеть» все синтаксические элементы команды (чтобы он мог ее скомпилировать). Поэтому макроопределения в Clipper не могут содержать команду (либо ее часть), но макросредства могут быть использованы для задания параметров команды. Они обычно используются в конструкциях типа FOR, WHILE и др. Далее приводится ряд примеров, поясняющих данное замечание.

Примеры.

Clipper скомпилирует

```
file = "ABC"
USE &file
comp = "nmbr2)nmbr1"
LIST name, city FOR
&comp
condition = [name=
= "Jones"]
LIST name FOR & con-
dition
ma = "A"
mx = "A [1]"
```

Clipper выдаст ошибку (а в dBASE III Plus допустим о)

```
file = "USE ABC"
&file
comp = "name, city FOR nmbr2)
nmbr1" LIST & comp
```

```
DECLARE & ma[1]
& mx = 100
```

```
DECLARE &mx  && в dBASE III Plus
             && нет массивов
```

```
? & ma [1], & mx
& ma [1] = 100
n = «1»
DO P& n
```

В каждом конкретном случае наилучшей проверкой является эксперимент.

3.4.2. ABS — абсолютное значение (dBASE III Plus и Clipper)

Класс: М

Синтаксис. ABS (⟨expN⟩)

Входные параметры: числовое выражение

Функция возвращает: абсолютное значение числового выражения.

Эта функция осуществляет вычисление абсолютной величины числового выражения. Она используется обычно, когда необходимо получить разность двух чисел вне зависимости от того, которое из них больше.

Результат работы функции ABS в dBASE III Plus всегда положителен.

В Clipper эта функция реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION ABS
* Синтаксис. ABS( ⟨expN⟩ )
* Функция возвращает: абсолютное значение числового выражения
*
PARAMETERS cl_n
RETURN IF (cl_n) = 0, cl_n, —cl_n)
```

Примеры.

```
j = 15
i = 60
? ABS (i — j)
45
? ABS (j—i)
45
```

3.4.3. ALLTRIM — удаление ведущих и хвостовых пробелов в строке (только Clipper)

Класс: С

Синтаксис. ALLTRIM(⟨expC⟩)

Входные параметры : символьная строка

Функция возвращает : преобразованную символьную строку

В dBASE III Plus в составе стандартных функций не реализована. В Clipper реализована на языке dBASE и содержится в файле Extenddb.prg:


```
FUNCTION ALLTRIM
* Синтаксис. ALLTRIM( ⟨expC⟩ )
* Функция возвращает: ⟨expC⟩ без ведущих и хвостовых пробелов
*
PARAMETERS cl_string
RETURN LTRIM(TRIM(cl_string))
```

Примеры.

```
a = " Москва "
b = ALLTRIM(a)
? b
Москва
? LEN (a)
10
? LEN (b)
6
```

3.4.4. AMPM — время в американском формате (a. m./p. m.) (только Clipper)

Класс: П

Синтаксис. AMPM(⟨time string⟩)

Входные параметры : символьная строка формата времени (чч : мм : сс)

Функция возвращает : преобразованную символьную строку

В dBASE III Plus в составе стандартных функций не реализована. В Clipper реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION AMPM
* Синтаксис. AMPM( ⟨time string⟩ )
* Возвращает: 11-байтовую символьную строку времени в американском 12-часовом формате (am/pm).
*
PARAMETERS cl_time
RETURN IF  ( VAL (cl_time) < 12, cl_time+ "am", ;
           IF (  VAL (cl_time = 12,  cl_time+" pm" , ;
                 STR( VAL (cl_time — 12,2 + SUBSTR ;
                 (cl_time, 3) +" pm"
```

Примеры.

```
a="23:11:34 "
? AMPM(a)
11 : 11 : 34 pm
```

3.4.5. ASC — получение ASCII кода символа (dBASE III Plus и Clipper)

Класс: П

Синтаксис. ASC (⟨expC⟩

Входные параметры : символьная строка

Функция возвращает : число (ASCII код первого символа)

Эта функция возвращает ASCII код первого символа в символьной строке.

Примеры.

```
? ASC ("Nestle")
78                    && ASCII код для символа "N" —78.
num = "123"
? ASC (num)
49                    && ASCII код для "1" —49
```

3.4.6. AT — выделение подстроки в строке (dBASE III Plus и Clipper)

Класс: С
Синтаксис. AT (⟨expC1⟩, ⟨expC2⟩)
Входные параметры : символьная строка1 (подстрока)
: символьная строка 2
Функция возвращает : число
Эта функция осуществляет поиск подстроки в строке. Возвращает номер позиции, начиная с которой заданная подстрока расположена в исходной строке. Если заданная подстрока не включена в исходную строку, то функция AT возвращает 0.

Примеры.

```
? AT ("is a", "This is a cat")
6
```

* подстрока "is a" содержится в строке "This is a cat" начиная
* с 6 позиции.

См. также SUBSTR( ), LEFT( ), RIGHT( ).

3.4.7. BOF — анализ признака начала файла (dBASE III Plus и Clipper)

Класс: Т
Синтаксис. BOF( )
Входные параметры : нет
Функция возвращает : логическую величину
Эта функция осуществляет идентификацию признака начала файла, содержащего активное отношение; возвращает логическую величину .Т. ,если курсор установлен перед первым кортежем. Во всех остальных случаях возвращает .F.
Используется, как правило, в прикладных программах при просмотре отношения покортежно в обратном порядке.

Примеры.

```
USE Clients
? BOF ( )
.F.                   && курсор установлен на первом кортеже
SKIP —1
? BOF ( )
.T.                   && курсор установлен перед первым кортежем
```

Эта команда удобна в циклах:

```
USE Travel
GO BOTTOM
DO WHILE .NOT. BOF ( )
```



```
   IF Agent="JT"
      ? RECNO ( )
   ENDIF
   SKIP —1
ENDD
```

См. также EOF ( ).

3.4.8: CDOW — день недели (dBASE III Plus и Clipper)

Класс: Д
Синтаксис. CDOW (⟨expD⟩)
Входные параметры : выражение типа date
Функция возвращает : символьную строку
Эта функция определяет название дня недели заданной переменной типа date.
Выражение типа date может быть сконструировано из переменных памяти этого типа, значений атрибутов типа date, результатов функций, возвращающих дату (например, функция DATE( ) — возврат системной даты).

Примеры.

```
Пусть системная дата 05/15/84
? CDOW (DATE( ) )
Tuesday
```

См. также DOW ( ), DAY ( ).

3.4.9. CHR — получение символа по ASCII коду (dBASE III Plus и Clipper)

Класс: П
Синтаксис. CHR (⟨expN⟩)
Входные параметры : числовое выражение
Функция возвращает : соответствующий символ
Функция CHR осуществляет преобразование числового выражения, рассматривая его как код ASCII, в соответствующий символ либо, если на клавиатуре нет эквивалентна этому коду, преобразование с помощью функции CHR приводит к специальным эффектам (например, звонок).
Выражение ⟨expN⟩ должно быть целым в диапазоне от 1 до 256.
CHR (0) рассматривается как эквивалент пустой символьной строки. Так как dBASE III Plus и Clipper оценивают символьные выражения начиная справа, использование функции CHR(0) возможно только в левой части равенства. Использование CHR (0) в правой части равенства приведет к некорректным результатам, так как dBASE III Plus при оценке такого равенства всегда будет возвращать .Т.

Примеры.

```
? CHR (65)
A
? CHR (7)
звонок
mem = "abc"
```

```
? mem = CHR (0)
.T.
? CHR (0) = mem
.F.                          && правильный ответ
```

См. также ASC ( ), INKEY ( ).

3.4.10. CMONTH — календарный месяц (dBASE III Plus и Clipper)

Класс: Д
Синтаксис. CMONTH (⟨expD⟩)
Входные параметры : выражение типа date
Функция возвращает : символьную строку

Эта функция определяет название месяца для заданного выражения типа date.

Выражение типа date может быть сконструировано из переменных памяти этого типа, значений атрибутов типа date, результатов функций, возвращающих дату (например, функция DATE ( ) — возврат системной даты).

Примеры.

Пусть системная дата 05/15/87

```
? CMONTH (DATE ( ) )
May
mmonth = CMONTH (DATE( ) )
? mmonth
May
```

См. также MONTH ( ).

3.4.11. COL — номер столбца текущего положения экранного курсора (dBASE III Plus и Clipper)

Класс: И
Синтаксис. COL ( )
Входные параметры : нет
Функция возвращает : число

Эта функция идентифицирует номер столбца текущего положения курсора. Позволяет контролировать положение курсора из тела программы.

Функция COL может быть использована для относительной адресации. Так, например, команда @ 1, COL ( ) + 5 спозиционирует курсор на экране на пять позиций вправо от его первоначального положения. С помощью функции COL из тела программы можно эффективно контролировать положение курсора на экране.

Примеры.

Если необходимо в программе окончить цикл по достижении курсором 70-го столбца, то фрагмент программы будет выглядеть следующим образом:

```
x = 0
DO WHILE COL ( ) <= 70
@ 5, x . . .
. . .
```



```
x = x + 1
. . .
ENDD
```

См. также ROW ( ), PROW ( ), PCOL ( ).

3.4.12. CTOD — преобразование символьного выражения в дату (dBASE III Plus и Clipper)

Класс: П
Синтаксис. CTOD (⟨expC⟩)
Входные параметры : символьное выражение
Функция возвращает : значение типа дата

Эта функция осуществляет преобразование даты, запомненной как символьная строка, в значение типа дата (т. е. для значения становятся допустимыми все операции и функции, определенные над датами). Обычно формат символьной строки следующий: mm/dd/yy, но он может быть переустановлен с помощью команд SET DATE и SET CENTURY.

По умолчанию подразумевается XX столетие, и поэтому год идентифицируется двумя цифрами.

Символьная строка ⟨expC⟩, задаваемая в функции CTOD, должна быть в интервале от "1/1/100" до "12/31/9999" (при этом год может быть задан лишь двумя последними цифрами, если SET CENTURY OFF, что имеет место по умолчанию).

Примеры.

Необходимо запомнить символьную строку 01/01/80 в переменной типа date:

```
testdate = CTOD ("01/01/80")
? testd te
01/01/80                     & & testdate переменная типа date
```

См. также команды SET CENTURY, SET DATE;
функцию DTOC ( ).

3.4.13. DATE — системная дата (dBASE III Plus и Clipper)

Класс: Д
Синтаксис. DATE ( )
Входные параметры : нет
Функция возвращает : текущую дату

Функция служит для определения системной даты в формате mm/dd/yy. Этот формат может быть переустановлен командами SET DATE или SET CENTURY.

Системная дата устанавливается на уровне DOS.

Примеры.

Необходимо запомнить системную дату в переменной памяти:

```
? DATE ( )
05/04/84
mdate = DATE ( )
? mdate
05/04/84
```

См. также команды SET DATE, SET CENTURY;
функцию TIME ( ).

3.4.14. DAY — день месяца (dBASE III Plus и Clipper)

Класс: Д
Синтаксис. DAY (⟨expD⟩)
Входные параметры : выражение типа date
Функция возвращает : число

Эта функция возвращает день месяца из заданного выражения типа date. Выражение типа date может быть сконструировано из переменных памяти этого типа, значений атрибутов типа date, результатов функций, возвращающих дату (например, функция DATE ( ) — возврат системной даты).

Примеры.

Пусть системная дата 05/15/84 :

```
mday = DAY (DATE ( ) )
? mday
15
```

См. также CDOW ( ), DOW ( ).

3.4.15. DAYS — преобразование секунд в целое число дней (только Clipper)

Класс: П
Синтаксис. DAYS (⟨seconds⟩)
Входные параметры : числовое выражение
Функция возвращает : числовое выражение

В dBASE III Plus в составе стандартных функций не реализована. В Clipper эта функция реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION DAYS
* Синтаксис. DAYS( ⟨seconds⟩ )
* Функция возвращает : целое число дней по количеству секунд
* Остаток возвращается функцией TSTRING ( ).
PARAMETERS cl_secs
RETURN INT( cl_secs / 86400 )
```

3.4.16. DBF — определение имени активного отношения (dBASE III Plus и Clipper)

Класс: И
Синтаксис. DBF ( )
Входные параметры : нет
Функция возвращает : символьную строку

Эта функция позволяет определить имя активного отношения (т. е. USE отношения) в текущей выбранной рабочей зоне.

Если нет активного отношения в выбранной рабочей зоне, то функция DBF возвращает пустую строку.

В Clipper эта функция реализована на языке dBASE, носит несколько фиктивный характер и содержится в файле Extenddb .prg:


```
FUNCTION DBF
* Синтаксис. DBF ( )
* Функция возвращает:строку "DBF", если есть активное отно-
*                    шение, иначе — пустую строку
*          В dBASE III Plus возвращает имя активного отношения.
RETURN IF ([ ]⟨FIELDNAME (1), "DBF", [ ])
```

Использование этой функции позволяет программе работать с отношениями, которые были активизированы до начала работы программы.

Примеры.

Чтобы определить в интерактивном режиме, какое отношение является активным, необходимо выполнить последовательность команд:

```
SET DEFAULT TO B
USE Clients
? DBF ( )
B: Clients .dbf
CLOSE DATABASE
? DBF ( )                    && возврат пустой строки
```

Рассмотрим пример, когда из тела программы необходимо определить, есть ли активное отношение:

```
Null = " "
IF Null ⟨DBF ( )          && сохранение имени активного
Oldfile = DBF ( )            отношения
USE Clients                && открытие нового отношения
⟨команды⟩
USE & Oldfile              && открытие старого отношения
ENDIF
```

См. также FIELD( ), FIELDNAME( ), LASTREC( ), LEN( ), LENNUM( ), LUPDATE( ), NDX( ), RECCOUNT( ), RECSIZE( ).

3.4.17. DELETED — анализ признака удаления для текущего кортежа (dBASE III Plus и Clipper)

Класс: Т
Синтаксис. DELETED( )
Входные параметры : нет
Функция возвращает : логическую величину

Эта функция служит для идентификации тех кортежей отношения, которые помечены признаком удаления. Возвращает логическую величину .T., если кортеж помечен к удалению и .F.— в противном случае.

Примеры.

Необходимо высветить все кортежи, помеченные к удалению:

```
DISPLAY FOR DELETED( )
```

См. также команду SET DELETED ON.

3.4.18. DISKSPACE — свободное пространство на диске (в байтах) (dBASE III Plus и Clipper)

Класс: И

Синтаксис. DISKSPACE ( )

Входные параметры : нет (для Clipper это число — код диска)

Функция возвращает : число

Эта функция позволяет определить свободное дисковое пространство (в байтах). В Clipper эта функция может иметь параметр — код диска: 1 — A, 2 — B и т. д. В Clipper эта функция реализована на языке СИ и содержится в файле Extendc. c:

```
#include «extend. h»
#define DEFAULT   0
#define FALSE     0
#define TRUE      1
#define NULL      " "
/*************************
* DISKSPACE ( )
* Синтаксис. DISKSPACE( [⟨expN⟩] )
* Функция возвращает: число байт свободного пространства на
                      диске
* Замечание. ⟨expN⟩ — код диска: 1 — A, 2 — B, 3 — C и т. д.
*/           по умолчанию — текущий диск.
DISKSPACE( )
{
  struct        /* структура для хранения дисковой информа-
                   ции */
        {
        unsigned no_clusts; /* число свободных кластеров */
        unsigned secs_clusts; /* количество секторов на кла-
                                 стер */
        unsigned clusts_drv; /* количество кластеров на диск */
        } drv_info;
  /* если есть параметр и он числовой */
if (PCOUNT==1 && ISNUM (1))
  {                          /* _dspace — функция Lattice */
    _dspase(—parni (1), & drv_info); */ заданный диск */
  }
else
    _dspace (DEFAULT, & drv_info);   /* диск умолчания */
  /* Возврат длинного целого в Clipper
  * байты: : = число кластеров *  число секторов/кластер *
  512 */ _retnl (512L * (long) drv_info.secs_clusts * (long)
  drv_info.no_clusts);
  }
```

Функция DISKSPACE используется совместно с функциями RECCOUNT ( ) (LASTREC ( )) и RECSIZE ( ) для автоматического создания копий отношений. Использование этих функций позволяет правильно оценить и распределить свободное дисковое пространство. В качестве примера такого использования этих функций см. пример в описании функции RECSIZE ( ).

Примеры.

Необходимо создать новое отсортированное отношение на диске B из уже имеющегося на диске отношения, которое занимает 1000000 байт:


```
mfilesize = 1000000
IF DISKSPACE ( ) > mfilesize*2
    SORT ON Lastname TO Nuwname
ELSE
    @22,0 SAY «недостаточно дискового пространства»
ENDIF
```

См. также GETE ( ), GETENV ( ), LASTREC ( ), RECCOUNT ( ), RECSIZE ( ), VERSION ( ).

3.4.19. DOW — день недели (dBASE III Plus и Clipper)

Класс: Д

Синтаксис. DOW (⟨expD⟩)

Входные параметры : выражение типа date

Функция возвращает : число

Эта функция возвращает порядковый номер дня недели из заданного выражения типа date начиная отсчет с воскресенья.

Выражение типа date может быть сконструировано из переменных памяти этого типа, значений атрибутов типа date, результатов функций, возвращающих дату (например, функция DATE ( ) — возврат системной даты).

Примеры.

Пусть системная дата 05/15/84.

```
? DOW  (DATE ( ) )
3                    && 3 — порядковый номер вторника
```

Пусть необходимо определить, какой день недели будет в будущем, например через 197 дней после системной даты:

```
? DOW (DATE ( ) + 197)
4                    && 4 — порядковый номер среды.
```

См. также CDOW ( ), DAY ( ).

3.4.20. DTOC — преобразование даты в символьную строку: мм/дд/гг (dBASE III Plus и Clipper)

Класс: П

Синтаксис. DTOC ((⟨expD⟩))

Входные параметры: выражение типа «дата»

Функция возвращает : символьную строку

Эта функция преобразует выражение типа date в символьную строку. По своему действию она обратна функции CTOD ( ).

Выражение типа date может быть сконструировано из переменных памяти этого типа, значений атрибутов типа date, результатов функций, возвращающих дату (например, функция DATE ( ) — возврат системной даты).

Примеры.

Пусть необходимо системную дату использовать как часть текста: testdate=DATE ( )

```
? testdate
05/15/84
```

```
? TYPE ("testdate")
D                          && тип переменной testdate — date
text= "Сегодняшняя дата :" +DTOC (testdate)
? text
Сегодняшняя дата : 05/15/84
```

См. также функцию CTOD( ),
команду SET CENTURY.

3.4.21. DTOS — преобразование даты в символьную строку: ггггммдд (только Clipper)

Класс: П
Синтаксис. DTOS(⟨expD⟩)
Входные параметры : выражение типа «дата»
Функция возвращает : символьную строку
В dBASE III Plus в составе стандартных функций не реализована.

Возвращает символьную строку вида ггггммдд. Это может быть использовано для индексации по дате как по некоторому символьному выражению.

Примеры.

```
xmas = CTOD ("12/25/85")
? DTOS (xmas)
19851225
```

3.4.22. ELAPTIME — возвращает символьную строку: количество секунд в заданном интервале времени (только Clipper)

Класс: Д
Синтаксис. ELAPTIME(⟨start time⟩, ⟨end time⟩)
Входные параметры : числовое выражение, числовое выражение
Функция возвращает : символьную строку времени
В dBASE III Plus в составе стандартных функций не реализована. В Clipper реализована на языке dBASE и содержится в файле Extenddb. prg:

```
FUNCTION ELAPTIME
* Синтаксис. ELAPTIME ( ⟨start time⟩, ⟨end time⟩ )
* Функция возвращает : строку времени, показывающую разницу между начальным и конечным значениями (секунд)
* Замечание. Если начальное значение больше конечного, то предполагается, что сутки сменились в полночь.
*            Имеет смысл только для временных интервалов в пределах 24 часов.
*            86400 — число секунд в 24 часах.
PARAMETERS cl_start, cl_end
RETURN TSTRING( IF (cl_end ⟨cl_start, 86400,0) +;
                    SECS (cl_end) — SECS (cl_start) )
```


3.4.23. EMPTY — проверка на нулевое (пустое) значение (только Clipper)

Класс: Т
Синтаксис. EMPTY (⟨exp⟩)
Входные параметры : выражение одного из типов:
символьное
числовое
логическое
дата
Функция возвращает : логическое значение
Функция возвращает значение «Истина», если:
параметром является символьная строка и она пустая либо содержит одни пробелы;
параметром является числовое выражение и оно равно нулю;
параметром является выражение типа «дата» и оно содержит пустую дату;
параметром является логическое выражение и оно ложно.
В dBASE III Plus в составе стандартных функций не реализована.

3.4.24. EOF — анализ признака конца файла (dBASE III Plus и Clipper)

Класс: Т
Синтаксис. EOF ( )
Входные параметры : нет
Функция возвращает : логическое значение
Эта функция осуществляет идентификацию признака конца файла, содержащего активное отношение. Возвращает логическую величину .T., если курсор установлен после последнего кортежа. Во всех остальных случаях возвращает .F.

Используется, как правило, в прикладных программах, при просмотре отношения покортежно.

Когда конец файла достигнут, т. е. функция EOF вернула величину .T., то RECNO ( ) = RECCOUNT ( ) + 1. При этом указатель текущего кортежа установлен на кортеже с номером на 1 больше, чем количество кортежей в отношении.

Следует отметить также, что некоторые команды (например, SKIP) по достижении конца файла непредвиденно выдают сообщение об ошибке вида «Конец файла найден неожиданно».

Примеры.

Пусть необходимо провести тестирование достижения конца файла:

```
USE Tours
GO BOTTOM
? EOF ( )
.F.                        && указатель кортежей не на конце файла
SKIP
? EOF ( )
.T.                        && указатель кортежей на конце файла
```

Эта команда удобна в циклах:

```
USE Trevel
GO TOP
```

```
DO WHILE .NOT. EOF ()
   IF Agent = "JT"
      ? RECNO ()
   ENDIF
   SKIP
ENDD
```

См. также BOF (), FOUND (), ERROR ().

3.4.25. ERROR — номер ошибки dBASE III Plus (только dBASE III Plus)

Класс: T
Синтаксис. ERROR ()
Входные параметры : нет
Функция возвращает : число

Эта функция возвращает номер ошибки dBASE III Plus. В Clipper функция не реализована, так как подобные ошибки там, как правило, выявляются в процессе компиляции, а не выполнения.

Эта функция позволяет средствами программы зафиксировать и обработать стандартную ошибочную ситуацию dBASE III Plus. Обрабатывающие действия, предпринимаемые в этом случае, могут включать в себя закрытие файлов, удаление некоторых файлов для создания свободного места во внешней памяти, повторный вывод меню для выбора других опций и т. п.

Для нормального функционирования ERROR () (возврата номера ошибки) должна быть активизирована ситуация ON ERROR (см. описание команды ON). Команды ETRTURN, RETRY сбрасывают номер в ноль (нет ошибки). Функция MESSAGE () возвращает связанное с данной ошибкой сообщение.

Примеры.

```
* главная программа MAIN.PRG
*
  ON ERROR DO Err_Prog WITH ERROR ()
.......
*конец MAIN.PRG
*ERR_PROG. PRG
PARAMETERS Error_no
    IF Error_no=54
        .........
        @ 24,0 SAY "Ошибка: " +SUBSTR(MESSAGE (), 4)
    ENDIF
    IF Error_no=5
        .........
    ENDIF
.........
```

См. также команды ON ERROR, RETRY, функцию MESSAGE ().

3.4.26. EXP — экспонента (dBASE III Plus и Clipper)

Класс: M
Синтаксис. EXP (⟨expN⟩)
Входные параметры : числовое выражение

Функция возвращает : число

Эта функция вычисляет экспоненту для числового выражения.

Примеры.

```
                1.000
```

Пусть требуется вычислить e:

```
? EXP (1.000)
2.718
Y = 1.000
? EXP (Y)
2.718
```

См. также SET DECIMALS.

3.4.27. FIELD (FIELDNAME) — имя атрибута в активном отношении (dBASE III Plus и Clipper)

Класс: И
Синтаксис. FIELD (⟨expN⟩) — для dBASE III Plus
FIELDNAME(⟨expN⟩) — для Clipper
Входные параметры : числовое выражение (номер атрибута)
Функция возвращает : символьную строку (его имя)

Эта функция служит для получения имени атрибута по задаваемому номеру атрибута в структуре активного отношения.

Функция возвращает пустую строку в том случае, когда номер атрибута задан некорректно, а именно: 1) номер атрибута не лежит в диапазоне от 1 до 128 для dBASE III Plus и 1024 для Clipper; 2) атрибута с таким номером нет в структуре отношения.

Так как функция FIELD адресуется к атрибутам по их номерам, то программа может обращаться к атрибутам как к элементам массива.

Функция FIELD возвращает все имена атрибутов только в символах верхнего регистра клавиатуры.

Примеры.

```
USE Clients
? FIELD (3)
ADDRESS
? FIELD (3) = Address
.F.                    && только символы верхнего регистра
number = 2
? FIELD (number)
LASTNAME
? FIELD (number + 1)
ADDRESS
```

См. также DBF (), LUPDATE (), NDX (), RECCOUNT (), RECSIZE ().

3.4.28. FILE — проверка существования файла (dBASE III Plus и Clipper)

Класс: T
Синтаксис. FILE (⟨filename⟩)
Входные параметры : символьная строка (имя файла)

Функция возвращает : логическое значение

С помощью этой функции возможно определить, существует ли файл с заданным именем. Если он существует, то функция FILE возвращает логическую величину .T.

Имя файла должно задаваться вместе с расширением и с явным указанием пути и дисковода, если искомый файл расположен не в текущем директории и/или не на активном диске. Имя файла либо должно быть символьной строкой в кавычках, либо может быть содержимым переменной памяти.

Примеры.

Пусть активным является диск A и файл Clients расположен на диске B.

```
? FILE ("Clients. dbf")
.F.
? FILE ("B: Clients")
.F.
? FILE ("B: Clients. dbf")
.T.
test="B : Clients. dbf"
? FILE (test)
.T.
```

3.4.29. FKLABEL — имена функциональных клавиш (dBASE III Plus и Clipper)

Класс: И

Синтаксис. FKLABEL (⟨expN⟩)

Входные параметры : числовое выражение

Функция возвращает : символьную строку

Эта функция возвращает символьное имя функциональной клавиши, соответствующее задаваемому номеру функциональной клавиши.

Различные терминалы определяют функциональные клавиши различными способами. Некоторые используют цифры от 1 до 10, другие — символические имена, такие, как SETUP.

В Clipper эта функция носит фиктивный характер, реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION FKLABEL
* Синтаксис. FKLABEL( ⟨expN⟩ )
* Функция возвращает. : имя ⟨expN⟩-й программируемой функ-
*                        циональной клавиши
* Замечание. F1 резервирован, поэтому первой «программируемой»
*            функциональной клавишей является F2.

PARAMETERS cl_1
RETURN IF (cl_1 < 10, "F" + LTRIM(STR(cl_1 + 1) ), [ ])
```

Функция FKLABEL позволяет программисту переопределить с помощью команды SET FUNCTION программируемые функциональные клавиши, которые имеют нецифровые маркировки.

Допускается любое числовое выражение в диапазоне от 1 до FKMAX( ).

Следует отметить, что FKLABEL(1) соответствует второй функциональной клавише на клавиатуре, что делает невозможным использование подсказки — HELP KEY.

Примеры.

Для того чтобы определить число программируемых функциональных клавиш на клавиатуре и установить их соответственно, можно включить следующий фрагмент программы:

```
i = 1
DO WHILE i <= FKMAX( )
   ACCEPT "SET" + FKLABEL (i) + "TO" TO String
   SET FUNCTION FKLABEL (i) TO String
   i = i + 1
ENDD
```

См. также функцию FKMAX ( ),
команду SET FUNCTION.

3.4.30. FKMAX — максимальный номер функциональной клавиши (dBASE III Plus и Clipper)

Класс: И

Синтаксис. FKMAX ( )

Входные параметры : нет

Функция возвращает : число

Эта функция позволяет установить функциональные клавиши, чтобы сделать более удобным использование терминала.

В Clipper эта функция носит фиктивный характер, реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION FKMAX
* Синтаксис. FKMAX ( )
* Функция возвращает: максимальное количество программируе-
*                     мых функциональных клавиш
* Замечание. F1 резервирована, поэтому первой «программируе-
*            мой» функциональной клавишей является F2.
RETURN 9    && специфика IBM
```

Пример использования см. в описании функции FKLABEL.

См. также команду SET FUNCTION.

3.4.31. FOUND — результат последней операции поиска в отношении (dBASE III Plus и Clipper)

Класс: Т

Синтаксис. FOUND ( )

Входные параметры : нет

Функция возвращает : логическое значение

Эта функция анализирует результат действия команд поиска: FIND, SEEK, LOCATE, CONTINUE.

Если поиск был произведен успешно, функция FOUND возвращает логическую величину .T., в противном случае — .F.

Обычно эта функция используется в программах для определения, по какой ветви программы идти дальше в зависимости от результата проведенного перед этим поиска в отношении.

Для каждой рабочей области результат функции FOUND ( ) уникален.

Если отношения связаны по условию командой SET RELATION, то при передвижении в активном отношении (открытом в ак-

тивной рабочей зоне) в связанной по условию зоне устанавливается FOUND ( ), так как при передвижении в активном отношении происходит неявный поиск и в связанном с ним отношении.

Если поиск в отношении производится командой, отличной от FIND, SEEK, LOCATE или CONTINUE, то функция FOUND возвращает логическую величину .F.

Примеры.

Пусть требуется проверить результат поиска в непроиндексированном отношении:

```
LOCATE FOR Agent = "JT"
DO WHILE FOUND ( )
   ? Agent
   CONTINUE
ENDDO
```

Пусть теперь требуется провести поиск в отношении, проиндексированном по атрибуту Agent:

```
SEEK "JT"
IF FOUND ( )
   DO WHILE Agent = "JT"
      ? Agent
      SKIP
   ENDDO
ENDIF
```

См. также команды CONTINUE, LOCATE, SEEK, SET RELATION; функции BOF ( ), FOF ( ).

3.4.32. GETENV (GETE) — значение переменных операционного окружения (dBASE III Plus и Clipper)

Класс: И

Синтаксис. GETENV (⟨expC⟩) — для dBASE III Plus
GETE (⟨expC⟩) — для Clipper

Входные параметры : символьная строка

Функция возвращает : символьную строку

Эта функция возвращает содержимое переменной операционной среды в виде символьной строки. Переменная операционной среды устанавливается SET командами DOS (см. документацию по DOS). Функция GETENV позволяет программе проверить содержимое переменных операционной среды. Если задаваемое символьное выражение не найдено, dBASE III Plus возвращает пустую строку.

Для более подробной информации о SET командах DOS и переменных операционной среды следует обратиться к документации по DOS.

В Clipper эта функция реализована на языке СИ и содержится в файле Extendc. c:

```
#include "extend.h"
#define DEFAULT     0
#define FALSE       0
#define TRUE        1
#define NULL        " "
/*************************
* GETE ()
```



```
* Синтаксис. GETE( ⟨expC⟩ )
* Функция возвращает: символьную строку из зоны переменных
*                окружения операционной системы, соответствующую заданной в ⟨expC⟩ переменной окружения.
* Замечание.  Аналог GETENV ( ) в dBASE III Plus. Имя изменено, так как оно конфликтует с функцией getenv( ) Lattice C, используемой в этой программе.
* Пример.     GETE( «PATH») возвращает текущий путь, установленный в операционной системе.
*/
GETE( )
{
 char *getenv( );
 char *str;
 /* если параметр один и он символьный */
 if (PCOUNT = = 1 && ISCHAR (1))
   {
   _UPPER ( );              /* преобразует параметр к заглавным буквам, используя внутреннюю функцию Clipper */
   str = getenv (_parc  (1)); /* получение содержимого переменной от DOS — функция Lattice C */
   _retc (str  ? str : NULL); /* возврат значения, если оно есть, либо пустой строки */
   }
 else
   _retc (NULL);            /* пустая строка, если такой переменной нет */
}
```

См. также DISKSPACE ( ), OS ( ), VERSION ( ).

3.4.33. IIF (IF) — выбор одного из выражений по условию (dBASE III Plus и Clipper)

Класс: Т

Синтаксис. IIF (⟨expL⟩, ⟨exp1⟩, ⟨exp2⟩) — для dBASE III Plus
IF (⟨expL⟩, ⟨exp1⟩, ⟨exp2⟩) — для Clipper

Входные параметры : логическое условие и два произвольных (однотипных) выражения

Функция возвращает : одно из выражений, указанных в качестве входных параметров

Эта функция позволяет строить условные выражения без конструкции IF . . . ENDIF. Она используется как в интерактивном режиме, так и в теле программы. Принцип работы не отличается от принципа работы условной конструкции IF . . .ENDIF. Если выражение ⟨expL⟩ возвращает логическую величину .T. , то функция IIF возвратит ⟨exp1⟩, в противном случае — ⟨exp2⟩.

Выражения ⟨exp1⟩ и ⟨exp2⟩ могут быть символьными, числовыми или выражениями типа date, но обязательно типы обоих этих выражений должны быть одинаковы. Тип возвращаемого результата будет такой же.

Можно использовать функцию IIF в MODIFY REPORT и MODIFY LABEL для условного определения содержимого атрибутов.

В Clipper эта функция носит наименование IF, но для совместимости с dBASE III Plus вводится фиктивная функция IIF, написанная на языке dBASE и содержащаяся в файле Extenddb .prg:

```
FUNCTION IIF
* Замечание. (exp1) и (exp2) должны быть однотипными.
*
PARAMETERS cl_if1, cl_if2, cl_if3
RETURN IF (cl_if1, cl_if2, cl_if3)
```

Примеры.

Рассмотрим фрагмент программы, использующий условную конструкцию:

```
IF Sex = "F"
   Mname = "Ms." + Lastname
ELSE
   Mname = "Mr. " + Lastname
ENDIF
```

Этот же фрагмент можно записать с использованием функции IIF:
Mname = IIF (Sex = "F", "Ms. ", "Mr. ") + Lastname

Рассмотрим еще один пример. Предположим, что имеется отношение с атрибутами Start_date и End_date и необходимо продлить End_date на 30 дней, если она уже прошла: LIST IIF (End_date > DATE ( ), End_date, End_date + 30)

3.4.34. INKEY — код нажатой клавиши (dBASE III Plus и Clipper)

Класс: B

Синтаксис. INKEY( ) — для dBASE III Plus
INKEY((expN)) — для Clipper

Входные параметры : нет — для dBASE III Plus,
числовое выражение — для Clipper

Функция возвращает : число (ASCII код нажатой клавиши)

Для dBASE III Plus эта функция возвращает целое число в диапазоне 0 — 255, соответствующее ASCII коду нажатой пользователем клавиши. Если в буфере ввода были некоторые символы, то INKEY ( ) возвращает код первого (наиболее раннего) из них и этот символ удаляется из буфера. Если ничего нажато не было, то INKEY ( ) вернет 0. Выполнение программы при вызове функции INKEY не прерывается.

В Clipper функция INKEY( ) реализована значительно более удобно. Эта функция приостанавливает выполнение программы до нажатия пользователем произвольной клавиши на клавиатуре (кроме клавиш-регистров) либо до истечения некоторого временного интервала, заданного в секундах в качестве параметра функции (возможно задание дробного числа). Вызов INKEY (0) приводит к произвольно долгому ожиданию нажатия клавиши. INKEY ( ) всегда возвращает собственно значение нажатой клавиши (или комбинации) или 0. Установка функциональных ключей (см. команду SET KEY) не меняет INKEY ( ) результат нажатия этих ключей. Задаваемый временной интервал отсчитывается по таймеру и не зависит от скорости центрального процессора.

Функция INKEY возвращает коды клавиш управления — таких, как стрелки или Ctrl— Home, что позволяет использовать

эти специальные клавиши для конструирования пользовательского интерфейса в программе.

Клавиши и соответствующие им INKEY величины представлены ниже:

Специальные клавиши	Альтернативные комбинации	INKEY результат
Вправо	Ctrl—D	4
Влево	Ctrl—S	19
Вверх	Ctrl—E	5
Вниз	Ctrl—X	24
Ctrl — вправо	Ctrl—B	2
Ctrl — влево	Ctrl—Z	26
Ins	Ctrl—V	22
Del	Ctrl—G	7
Home	Ctrl—A	1
End	Ctrl—F	6
PgUp	Ctrl—R	18
PgDn	Ctrl—C	3
Ctrl — Home	Ctrl—]	29
Ctrl — End	Ctrl—W	23
Ctrl — PgUp	Ctrl——	31
Ctrl — PgDn	Ctrl—^	30

Следует отметить, что в dBASE III Plus нажатие клавиши «влево» или Ctrl — S может интерпретироваться системой как сигнал к прекращению скролинга. Чтобы избежать этого, необходимо использовать команду SET ESCAPE OFF.

Примеры.

```
DO WHILE .T.
   *
   (меню)
   *
   i = 0
   DO WHILE i = 0
      @ 1,72 SAY TIME( )
      i = INKEY ( )              && INKEY (1) — для Clipper
   ENDDO
   *
   DO CASE                       && обработка INKEY( ) результата
      CASE CHR (i) $ "Qq0"
           RETURN                  выход
      CASE CHR (i) $ "Aa1"
           DO (procedure1)
      CASE CHR (i) $ "Bb2"
           DO (procedure1)
      . . . . . .
   ENDCASE
ENDDO
```

вывод времени не чаще одного раза в секунду, сопровождающийся параллельным отслеживанием ввода с клавиатуры

```
m_inkey = 0
DO WHILE m_inkey = 0
   @ 1,72 SAY TIME ( )
```

```
m_time = TIME ()
DO WIHLE m_time=TIME( ) .AND. m_inkey = 0
                          && здесь проявляется
                       * неоднозначность языка dBASE,
                       * заключающаяся в том, что знак
                       * «=» используется и в присвоении
                       * и в проверке условия равенства
                       * (в данном случае — это условие
                       * равенства)
     m_inkey =INKEY ( )  && INKEY (0.5) — для Clipper
                        (например)
     .....
   ENDDO
ENDDO
* для Clipper
? [ Авторское право : НПО «Горсистемотехника», 1987 ]
i = INKEY (3)                  && задержка на 5 секунд
CLEAR
```

См. также команды ON KEY, SET TYPEAHEAD;
функции CHR ( ), LACTKEY ( ), READKEY ( ).

3.4.35. INT — преобразование в целое (dBASE III Plus и Clipper)

Класс: M
Синтаксис. INT (⟨ expN⟩)
Входные параметры : числовое выражение
Функция возвращает : целое число
Эта функция осуществляет преобразование задаваемого числового выражения в целое число, отбрасывая все знаки справа от десятичной точки.

Примеры.

Необходимо преобразовать к целому число 10.23 :

```
? INT (10.23)
10
t = 10.23
? t
10.23
? INT (t)
10
```

3.4.36. ISALPHA — проверка на букву (dBASE III Plus и Clipper)

Класс: T
Синтаксис. ISALPHA (⟨expC⟩)
Входные параметры : символьное выражение
Функция возвращает : логическую величину
Функция ISALPHA анализирует символьное выражение на начальный символ. Если выражение начинается с латинской буквы верхнего или нижнего регистров, функция ISALPHA возвращает

логическую величину .T. Для всех других начальных символов она возвращает .F.

В Clipper эта функция реализована на языке dBASE и содержится в файле Extenddb. prg:

```
FUNCTION ISALPHA
* Синтаксис. ISALPHA( ⟨expC⟩ )
* Функция возвращает : истину, если первый символ в ⟨expC⟩
                       латинская буква
PARAMETERS cl_string
RETURN UPPER (SUBSTR  (cl_string, 1,1) ) $ [ABCDEFGHIJ
                                    KLMNOPQRSTUVWXYZ]
```

Примеры.

```
? ISALPHA ("abc123")
.T.
? ISALPHA ("ABC123")
.T.
? ISALPHA ("123ABC")
.F.
```

См. также ISLOWER ( ), ISUPPER( ), LOWER( ), UPPER ( ).

3.4.37. ISCOLOR — проверка наличия цветного дисплея (dBASE III Plus и Clipper)

Класс: T
Синтаксис. ISCOLOR ( )
Входные параметры : нет
Функция возвращает : логическую величину
Эта функция анализирует цветовой режим, который установлен в настоящее время. Если режим цветовой, то функция возвращает логическую величину .T. , если же режим монохромный, то возвращается величина .F.

Использование функции ISCOLOR позволяет строить прикладную программу таким образом, что она не зависит от того, в каком режиме она будет выполняться, а также позволяет выбирать цвета с помощью команды SET COLOR.

В Clipper эта функция реализована на языке Ассемблер и содержится в файле Extenda .asm:

```
NAME EXTENDA
PUBLIC ISCOLOR
PUBLIC ISPRINTER
; Вызовы возвратов управления в Clipper
EXTRN _RETC : FAR       ; возврат символьной строки
EXTRN _RETDS : FAR      ; возврат даты по строке типа
                          «ггггммдд»
EXTRN _RETL : FAR       ; возврат логической величины
EXTRN _RETNI : FAR      ; возврат слова как числа
EXTRN _RETNL : FAR      ; возврат двойного слова как числа
EXTRN _RETND : FAR      ; возврат с плавающей точкой как
                          числа

_PROG SEGMENT
ASSUME CS: — PROG
;-------------------------------------------------------
```

```
; ISCOLOR ( )
; Синтаксис: ISCOLOR ( )
; Функция возвращает: "Истину", если есть адаптер цветного дис-
                      плея, иначе — "Ложь"
ISCOLOR PROC FAR
MOV         AH, 15              ; функция текущего видеостатуса
INT         10H                 ; чтение видеостатуса
XOR         BX, BX              ; ложь
CMP         AL,07               ; монохромный 80 × 25
JE          RET_ISCOLOR         ; возврат "ложь", если да
MOV         BX, 1               ; истина
RET_ISCOLOR:
PUSH        BX                  ; положить возвращаемое значе-
                                  ние в стек
CALL        _RETL               ; вернуть логическое значение
                                  в Clipper
POP         BX                  ; восстановить стек
RET
ISCOLOR ENDP
_PROG ENDS
END
```

Примеры.

Для того чтобы установить желаемые цвета из программы независимо от того, в цветном или монохромном режиме она работает, можно использовать следующую группу команд:

```
IF ISCOLOR( )
   SET COLOR TO GR/B, W/R, GR
ELSE
   SET COLOR TO W+
ENDIF
```

См. также SET COLOR.

3.4.38. ISLOWER — проверка на строчную букву (dBASE III Plus и Clipper)

Класс: T
Синтаксис. ISLOWER (⟨expC⟩)
Входные параметры : символьная строка
Функция возвращает : логическую величину

Эта функция возвращает логическую величину .T., если заданное символьное выражение начинается с латинской буквы в нижнем регистре клавиатуры. Для всех остальных символов клавиатуры функция ISLOWER возвращает логическую величину .F.

В Clipper эта функция реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION ISLOWER
* Синтаксис. ISLOWER( ⟨expC⟩ )
* Функция возвращает: "Истину", если первый символ в ⟨expC⟩
                      строчная латинская буква
PARAMETERS cl_string
RETURN SUBSTR (cl_string, 1,1) $ [abcdefghijklmnopqrstuvwxyz]
```


Примеры.

```
? ISLOWER ("abc123")
.T.
? ISLOWER ("ABC123")
.F.
? ISLOWER ("123abc")
.F.
```

См. также ISALPHA( ), ISUPPER( ), LOWER ( ), UPPER ( ).

3.4.39. ISPRINTER — проверка готовности принтера (только Clipper)

Класс: T
Синтаксис. ISPRINTER( )
Входные параметры : нет
Функция возвращает : логическую величину

Функция возвращает «истину», если принтер «online» и готов, иначе — «ложь». Это очень удобно при необходимости вывода на принтер в Clipper — пока не будет подключен принтер и не будет обеспечена его готовность, программа не начнет вывод, что исключает «зависание» программы при попытке вывода на неготовый принтер, характерное для dBASE III Plus.

В dBASE III Plus в составе стандартных функций не реализована.

В Clipper эта функция реализована на языке Ассемблер и содержится в файле Extenda .asm:

```
NAME  EXTENDA
PUBLIC ISCOLOR
PUBLIC ISPRINTER
; Вызовы возвратов управления в Clipper
EXTRN     _RETC : FAR   ; возврат символьной строки
EXTRN     _RETDS : FAR  ; возврат даты по строке типа
                          «ггггммдд»
EXTRN     _RETL : FAR   ; возврат логической величины
EXTRN     _RETNI : FAR  ; возврат слова как числа
EXTRN     _RETNL : FAR  ; возврат двойного слова как числа
EXTRN     _RETND : FAR  ; возврат с плавающей точкой как
                          числа
_PROG SEGMENT
ASSUME CS:_PROG
;-----------------------------------------------------------
; ISPRINTER ( )
; Синтаксис. ISPRINTER ( )
; Функция возвращает: "Истину", если принтер "online" и готов,
;                     иначе — "Ложь"
ISPRINTER PROC FAR
MOV         AH, 2H              ; функция статуса принтера
MOV         DX, OH              ; какой принтер проверять
INT         17H                 ; прочесть статус принтера
XOR         BX, BX              ; ложь
CMP         AH, 90H             ; не занят или не выбран (90h =
                                  = 10010000)
```

```
JNE         RET_ISPRINTER   ; возврат «ложь»
MOV         BX, 1           ; истина
RET_ISPRINTER:
PUSH        BX              ; поместить возвращаемое значение в стек
CALL        _RETL           ; возврат логической величины в Clipper
POP         BX              ; восстановить стек
RET
ISPRINTER ENDP
_PROG ENDS
END
```

Примеры.

```
@ 20,0 SAY [Включите, пожалуйста, принтер. . .]
DO WHILE ! ISPRINTER ( )
ENDDO
@ 20,0
. . .
```

3.4.40. ISUPPER — проверка на прописную букву (dBASE III Plus и Clipper)

Класс: T

Синтаксис. ISUPPER (⟨expC⟩)

Входные параметры : символьная строка

Функция возвращает : логическую величину

Эта функция возвращает логическую величину .T., если задаваемое символьное выражение начинается с латинской буквы в верхнем регистре клавиатуры. Для всех остальных символов клавиатуры функция ISUPPER возвращает логическую величину .F.

В Clipper эта функция реализована на языке dBASE и содержится в файле Extenddb .prg:

```
FUNCTION ISUPPER
* Синтаксис. ISUPPER( ⟨expC⟩ )
* Функция возвращает: "Истину", если первый символ в ⟨expC⟩— строчная латинская буква

PARAMETERS cl_string
RETURN SUBSTR (cl_string, 1,1) $ [ABCDEFGHIJKLMNOPQ
                                  RSTUVWXYZ]
```

Примеры.

```
? ISUPPER ("abc123")
.F.
? ISUPPER ("ABC123")
.T.
? ISUPPER ("123abc")
.F.
```

См. также ISALPHA( ), ISLOWER( ), LOWER( ), UPPER( ).

3.4.41. LASTKEY (READKEY) — код последней нажатой клавиши (dBASE III Plus и Clipper)

Класс: B

Синтаксис. LASTKEY ( ) — для Clipper
READKEY ( ) — для dBASE III Plus и Clipper

Входные параметры : нет

Функция возвращает : число

Функция READKEY( ) возвращает целое, представляющее собой ASCII код клавиши, нажатой для того, чтобы выйти из режима экранного редактирования, инициированного одной из команд типа APPEND, BROWSE, CHANGE, CREATE, EDIT, INSERT, MENU, MODIFY, READ. Она также анализирует, были ли модифицированы какие-либо данные.

Функция LASTKEY ( ), входящая в стандартный набор функций Clipper, отличается от функции READKEY ( ) dBASE III Plus. Эта функция возвращает ASCII код нажатой ранее клавиши. Коды, возвращаемые по нажатию специальных клавиш, приводятся в описании функции INKEY ( ). Вообще говоря, функция Clipper LASTKEY( ) по своему смыслу совпадает с dBASE-овской интерпретацией функции INKEY ( ).

В связи с различиями LASTKEY( ) и READKEY( ) в Clipper на языке dBASE реализована функция READKEY ( ), содержащаяся в файле Extenddb.prg:

```
FUNCTION READKEY
* Синтаксис. READKEY ( )
* Функция возвращает: число, представляющее клавишу, нажатую для выхода из режима экранного
*                     редактирования
* Различия между dBASE-овским READKEY( ) и Clipper-овским LASTKEY( ):
*
```

*	Клавиша выхода	dBASE	Clipper
*	Backspace	0	Нет выхода
*	^D, ^L	1	Нет выхода
*	Lt arrow	2	Нет выхода
*	Rt arrow	3	Нет выхода
*	Up arrow	4	Нет выхода
*	Dn arrow	5	Нет выхода
*	PgUp	6	18
*	PgDn	7	3
*	Esc, ^Q	12	27 (только Esc)
*	^End, ^W	14	23 (только ^W)
*	Исчерпание поля ввода	15	ASCII код последнего введенного символа
*	Enter	15	13
*	^Home	33	Нет выхода
*	^PgUp	34	31
*	^PgDn	35	30
*	F1	36	Нет выхода

```
* dBASE III добавляет 256 к коду возврата в случае, если пользователь изменил что-либо
* Clipper использует свою функцию UPDATED ( ) для
```

```
*
*        определения, были ли сделаны изменения
DO CASE
   CASE LASTKEY ( ) = 18                                  && PgUp
        RETURN 6 + IF (UPDATED ( ), 256 , 0)
   CASE LASTKEY ( ) = 3                                   && PgDn
        RETURN 7 + IF (UPDATED ( ), 256,0)
   CASE LASTKEY ( ) = 27                                  && Esc
        RETURN 12 + IF (UPDATED ( ), 256,0)
   CASE LASTKEY ( ) = 23                                  &&^W
        RETURN 14 + I F (UPDATED ( ), 256,0)
   CASE LASTKEY ( ) = 13                                  && Enter
        RETURN 15 + IF (UPDATED ( ), 256,0)
   CASE LASTKEY ( ) = 31                                  &&^PgUp
        RETURN 34 + IF (UPDATED ( ), 256,0)
   CASE LASTKEY ( ) = 30                                  &&^PgDn
        RETURN 35 + IF (UPDATED ( ), 256,0)
   CASE LASTKEY ( ) > = 32                                && type past end
        RETURN 15 + IF (UPDATED ( ), 256,0)
ENDCASE
```

Функция READKEY ( ) помогает определить необходимые действия после того, как оператор вышел из одного из режимов экранного редактирования: APPEND, BROWSE, CHANGE, CREATE, EDIT, INSERT, MODIFY, READ.

Можно использовать специальные переменные, имена которых соответствуют возможным вариантам выхода из режима экранного редактирования, соответствующие коду нажатой клавиши, которые облегчают построение и читабельность программ. При этом важно помнить, что возможны два варианта выхода из режима экранного редактирования: если пользователь не менял данных в ходе экранного редактирования (коды выхода 0—36) и если он обновлял данные — возвращаемый код увеличивается на 256.

Приводимые ниже имена переменных относятся только к кодам 0—36; если требуется идентифицировать определенный выход с обновлением данных, то необходимо к соответствующей переменной прибавить переменную rc_update (или код 256). Список этих имен, соответствующих им клавиш и вызывающих выход действий приведен ниже (только для dBASE III Plus, для Clipper — см. текст функции READKEY ( ) — не все клавиши выхода dBASE III Plus задействованы в Clipper).

Имя переменной	Код	Код с обновлением (+rc_update)	Нажатые клавиши	Значение клавиш
rc_update	—	256	—	—
rc_bakchr	0	256	Ctrl—H Ctrl—S Backspace	Назад на символ
rc_forchr	1	257	Ctrl—D Ctrl—L	Вперед на символ
rc_bakwrd	2	258	Ctrl—A	Назад на слово
rc_forwrd	3	259	Ctrl—F	Вперед на слово
rc_bakfld	4	260	Ctrl—E Ctrl—K	Назад на атрибут
rc_forfld	5	261	Ctrl—J Ctrl—X	Вперед на атрибут
rc_bakscr	6	262	Ctrl—R	Назад на экран
rc_forscr	7	263	Ctrl—C	Вперед на экран
rc_bakpan	8	264	Ctrl—Z	
rc_forpan	9	265	Ctrl—B	
rc_delete	10	266	Ctrl—U	Удаление
rc_insert	11	267	Ctrl—N	Вставить
rc_quit	12	268	Ctrl—Q	Продолжать без сохранения
	13		Esc	Не используется
rc_write	14	270	Ctrl—W Ctrl—End	Продолжать с сохранением
rc_filled	15	271	Ctrl—M	Переполнение поля ввода
rc_return	16	272	Enter	Клавиша ввода
rc_menu	33	289	Ctrl—Home	Переключатель включающий/выключающий меню
rc_zomout	34	290	Ctrl—PgUp	Общий вид (в целом)
rc_zomin	35	291	Ctrl—PgDn	Детализация (фрагмент)
rc_help	36	292	F1	Функциональная клавиша подсказки

Специальные клавиши клавиатуры и соответствующие им величины, возвращаемые функцией READKEY ( ), приведены ниже.

Примеры.

Для того чтобы определить, были ли сделаны изменения в переменной памяти, и выполнить REPLACE в этом случае, можно выполнить следующие команды:

```
. . . . . .
```

Специальные клавиши	Альтернативный вариант	Значение READKEY
Вправо	Ctrl—D	1
Влево	Ctrl—S	0
Вверх	Ctrl—E	4
Вниз	Ctrl—X	5
Ctrl — вправо	Ctrl—B	9
Ctrl — влево	Ctrl—Z	8
Ins	Ctrl—V	Нет
Del	Ctrl—G	Нет
Home	Ctrl—A	2
End	Ctrl—F	3
PgUp	Ctrl—R	6
PgDn	Ctrl—C	7
Ctrl — Home	Ctrl—J	33
Ctrl — End	Ctrl—W	14
Ctrl — PgUp	Ctrl——	34
Ctrl — PgDn	Ctrl—^	35

```
Mname = name
Msal = salary
@ 5,10 say [Ф. И. О. служащего]
@ 5,30 get Mname
@ 6,10 say [Оклад]
@ 6,30 get Msal
```

```
@ 22,5 say [Нажмите ESC для сброса изменений]
READ
IF READKEY ( ) < > rc_quit + rc_update    &&  или LASTKEY
                                              ( ) < > 27
                                          *   для Clipper
   REPLACE name WITH Mname, salary
   WITH Msal
ENDIF
```

См. также функцию INKEY ( );
команды ON KEY, READ.

3.4.42. LASTREC (RECCOUNT) — определение числа записей в активном отношении (dBASE III Plus и Clipper)

Класс: И

Синтаксис. LASTREC ( ) — для Clipper
RECCOUNT ( ) — для dBASE III Plus и Clipper

Входные параметры : нет

Функция возвращает : число

Функции LASTREC ( ) в Clipper и RECCOUNT ( ) в dBASE III Plus производят подсчет логических записей (кортежей) в активном отношении. По своему результату использование этих функций в dBASE III Plus и в Clipper аналогично выполнению команды GO BOTTOM с последующим определением порядкового номера последней записи непроиндексированного отношения с помощью функции RECNO ( ), но производится гораздо быстрее.

Если в данный момент нет активного отношения, то результатом функций LASTREC ( ) и RECCOUNT ( ) будет 0.

Функции LASTREC ( ) и RECCOUNT ( ) при подсчете учитывают все кортежи отношения даже в случае активности режимов SET DELETE и SET FILTER ( ).

Рекомендуется использовать функции LASTREC ( ) либо RECCOUNT ( ) (соответственно в Clipper либо dBASE III Plus) совместно с функциями RECSIZE( ) и DISKSPASE( ) для определения свободного пространства на диске при необходимости получения автоматических копий файлов (см. пример в описании функции RECSIZE ( )).

Для совместимости Clipper и dBASE III Plus в Clipper введена фиктивная функция RECCOUNT ( ), написанная на языке dBASE и содержащаяся в Extenddb.prg:

```
FUNCTION RECCOUNT
*Синтаксис. RECCOUNT ( )
* Функция возвращает : число кортежей в активном отношении
*
RETURN LASTREC ( )
```

Примеры.

```
USE Tours
? RECCOUNT ( )
20
USE
```

См. также DBF ( ), DISKSPACE ( ), FIELD ( ), NDX ( ), RECSIZE ( ).

3.4.43. LEFT — выбор подстроки от левого конца строки (dBASE III Plus и Clipper)

Класс: С

Синтаксис. LEFT (⟨expC⟩, ⟨expN⟩)

Входные параметры : символьное выражение, число

Функция возвращает : символьную строку

Эта функция выделяет заданное количество символов в заданной символьной строке начиная с первого символа. Эквивалентна функции SUBSTR с заданной позицией начала — первый символ.

Если числовое выражение, задающее количество выделяемых символов, равно нулю или отрицательно, то результатом действия функции LEFT будет пустая строка.

Если ⟨expN⟩ больше длины выражения ⟨expC⟩, то функция LEFT возвращает исходную строку.

В Clipper функция LEFT реализована в файле Extenddb.prg:

```
FUNCTION LEFT
* Синтаксис : LEFT ( ⟨expC⟩, ⟨expN⟩ )
* Функция возвращает :⟨expN⟩ символов выражения ⟨expC⟩ на-
*                     чиная слева
PARAMETERS cl_string, cl_len
RETURN SUBSTR (cl_string, 1, cl_len)
```

Примеры.

```
? LEFT ("abc123", 3)
abc
```

См. также AT ( ), LTRIM ( ), RIGHT ( ), RTRIM ( ), STUFF( ), SUBSTR ( ), TRIM ( ).

3.4.44. LEN — определение длины символьной строки (dBASE III Plus и Clipper)

Класс: С

Синтаксис. LEN (⟨expC⟩)

Входные параметры : символьная строка

Функция возвращает : число

Эта функция определяет длину символьной строки. Если задана пустая строка, то функция LEN возвращает 0. Если символьным выражением является имя атрибута, то функция LEN возвратит длину атрибута, заданную в структуре отношения, а не длину содержащейся в этом атрибуте информации.

Примеры.

```
? LEN (" ЭТО строка ")
12
text = " Это строка"
? LEN (text)
12
```

Пусть в отношении есть атрибут Address с заданной длиной, равной 25 символам, и в этом атрибуте содержится следующий адрес: «Киев Красноармейская». Тогда, чтобы определить длину истинного адреса, необходимо выполнить команды

```
? LEN (Address)
25
? LEN (TRIM (Address))
20
```

3.4.45. LENNUM — определение длины числа (только Clipper)

Класс: И
Синтаксис. LENNUM (⟨expN⟩)
Входные параметры : число
Функция возвращает : число
Функция LENNUM определяет длину числа, т. е. количество цифр.
В состав стандартных функций dBASE III Plus не входит.
В Clipper функция LENNUM реализована на языке dBASE и находится в файле Extenddb.prg.

```
FUNCTION LENNUM
* Синтаксис : LENNUM ( ⟨expN⟩ )
* Функция возвращает : длину числа ⟨expN⟩
*
PARAMETERS cl_number
RETURN LEN ( TRIM ( STR (cl_number) ) )
```

3.4.46. LOG — вычисление натурального логарифма числа (dBASE III Plus и Clipper)

Класс: М
Синтаксис. LOG (⟨expN⟩)
Входные параметры : число
Функция возвращает : число
Эта функция вычисляет натуральный логарифм заданного числа.

Примеры.

Пусть требуется вычислить натуральный логарифм для числа 2.71828:

```
? LOG (2.71828)
1.0000
x = 4
y = 2
? LOG (x * y)
2.08
```

См. также EXP ( ), SET DECIMALS, SET FIXED.

3.4.47. LOWER — преобразование прописных латинских букв в строчные (dBASE III Plus и Clipper)

Класс: С
Синтаксис. LOWER (⟨expC⟩)
Входные параметры : символьная строка
Функция возвращает : символьную строку

Эта функция преобразует в заданном символьном выражении прописные буквы латинского алфавита в строчные.

Примеры.

```
? LOWER ( "THIS IS A NICE DAY")
this is a nice day
STORE 'THIS IS A CAT' TO A
? A
THIS IS A CAT
? LOWER (A)
this is a cat
```

См. также ISALPHA( ), ISLOWER ( ), ISUPPER ( ), UPPER( ).

3.4.48. LTRIM — подавление ведущих пробелов в строке (dBASE III Plus и Clipper)

Класс: С
Синтаксис. LTRIM (⟨expC⟩)
Входные параметры : символьная строка
Функция возвращает : символьную строку
Эта функция уничтожает ведущие пробелы в заданном символьном выражении. Рекомендуется использовать функцию LTRIM для подавления ведущих пробелов, возникающих вследствие функции STR.

Примеры.

```
? STR (95.11, 8, 2)
    95.11
? LTRIM (STR (95.11, 8, 2))
95.11
```

См. также LEFT ( ), RIGHT ( ), RTRIM ( ), SUBSTR ( ), TRIM ( ), STR ( ).

3.4.49. LUPDATE — определение даты последней модификации отношения (dBASE III Plus и Clipper)

Класс: Т
Синтаксис. LUPDATE ( )
Входные параметры : нет
Функция возвращает : переменную типа дата
Эта функция служит для определения даты последней модификации активного в данный момент отношения. В Clipper функция LUPDATE реализована на языке СИ и содержится в файле Extende. c:

```
/*************************************
* LUPDATE ( )
* Синтаксис. LUPDATE ( )
* Функция возвращает : дату последней записи в активное отно-
                       шение
*/
LUPDATE( )
{
  unsigned year, month, day;
```

```
? LEN (Address)
25
? LEN (TRIM (Address))
20
```

3.4.45. LENNUM — определение длины числа (только Clipper)

Класс: И
Синтаксис. LENNUM (⟨expN⟩)
Входные параметры : число
Функция возвращает : число
Функция LENNUM определяет длину числа, т. е. количество цифр.
В состав стандартных функций dBASE III Plus не входит.
В Clipper функция LENNUM реализована на языке dBASE и находится в файле Extenddb.prg.

```
FUNCTION LENNUM
* Синтаксис : LENNUM ( ⟨expN⟩ )
* Функция возвращает : длину числа ⟨expN⟩
*
PARAMETERS cl_number
RETURN LEN ( TRIM ( STR (cl_number) ) )
```

3.4.46. LOG — вычисление натурального логарифма числа (dBASE III Plus и Clipper)

Класс: М
Синтаксис. LOG (⟨expN⟩)
Входные параметры : число
Функция возвращает : число
Эта функция вычисляет натуральный логарифм заданного числа.

Примеры.

Пусть требуется вычислить натуральный логарифм для числа 2.71828:

```
? LOG (2.71828)
1.0000
x = 4
y = 2
? LOG (x * y)
2.08
```

См. также EXP ( ), SET DECIMALS, SET FIXED.

3.4.47. LOWER — преобразование прописных латинских букв в строчные (dBASE III Plus и Clipper)

Класс: С
Синтаксис. LOWER (⟨expC⟩)
Входные параметры : символьная строка
Функция возвращает : символьную строку

Эта функция преобразует в заданном символьном выражении прописные буквы латинского алфавита в строчные.

Примеры.

```
? LOWER ( "THIS IS A NICE DAY")
this is a nice day
STORE 'THIS IS A CAT' TO A
? A
THIS IS A CAT
? LOWER (A)
this is a cat
```

См. также ISALPHA( ), ISLOWER ( ), ISUPPER ( ), UPPER( ).

3.4.48. LTRIM — подавление ведущих пробелов в строке (dBASE III Plus и Clipper)

Класс: С
Синтаксис. LTRIM (⟨expC⟩)
Входные параметры : символьная строка
Функция возвращает : символьную строку
Эта функция уничтожает ведущие пробелы в заданном символьном выражении. Рекомендуется использовать функцию LTRIM для подавления ведущих пробелов, возникающих вследствие функции STR.

Примеры.

```
? STR (95.11, 8, 2)
     95.11
? LTRIM (STR (95.11, 8, 2))
95.11
```

См. также LEFT ( ), RIGHT ( ), RTRIM ( ), SUBSTR ( ), TRIM ( ), STR ( ).

3.4.49. LUPDATE — определение даты последней модификации отношения (dBASE III Plus и Clipper)

Класс: Т
Синтаксис. LUPDATE ( )
Входные параметры : нет
Функция возвращает : переменную типа дата
Эта функция служит для определения даты последней модификации активного в данный момент отношения. В Clipper функция LUPDATE реализована на языке СИ и содержится в файле Extendc. c:

```
/*************************
* LUPDATE ( )
* Синтаксис. LUPDATE ( )
* Функция возвращает : дату последней записи в активное отно-
                       шение
*/
LUPDATE( )
{
  unsigned year, month, day;
```

```
  char date_str [9];
if (DBf_OPEN)                    /* дата из заголовка активного
                                    отношения*/
{
  year=1900+DBF_DATE [1];
  month=     DBF_DATE [2];
  day=       DBF_DATE [3];
}
else
                                 /* если отношение не является
                                    активным, т. е. дата из заго-
                                    ловка равна 0, то возвраща-
                                    ется пустая дата*/
  year=month=day=0;
  /*построение строки в формате даты*/
  date_str [0] = '1'
  date_str [1] = '9'
  date_str [2] = '0' + (year % 100) / 10;
  date_str [3] = '0' + (year % 10);
  date_str [4] = '0' + (month / 10);
  date_str [5] = '0' + (month % 10);
  date_str [6] = '0' + (day / 10);
  date_str [7] = '0' + (day % 10);
  date_str [8] = '0' + '\0';
  _retds (date_str);             /*возврат даты*/
}
```

См. также DBF ( ), FIELD ( ), NDX ( ), RECCOUNT ( ), REC SIZE ( ).

3.4.50. MAX — определение большего из двух чисел (dBASE III Plus и Clipper)

Класс: M

Синтаксис. MAX (⟨expN1⟩, ⟨expN2⟩)

Входные параметры : два числовых выражения

Функция возвращает : число

Эта функция сравнивает два заданных числовых выражения и возвращает большее из них. В Clipper функция MAX реализована на языке dBASE и находится в файле Extenddb.prg:

```
FUNCTION MAX
*Синтаксис: MAX ( ⟨expN1⟩, ⟨expN2⟩ )
*Функция возвращает : наибольшее из двух чисел
*
PARAMETERS cl_n1, cl_n2
RETURN IF (cl_n1 < cl_n2, cl_n1, cl_n2)
```

Пример.

```
? MAX (59, 45)
59
STORE 7 TO t
STORE 12 TO t1
? MAX (t, t1)
12
```

См. также MIN ( ).

3.4.51. MEMOEDIT — редактирование переменных или атрибутов типа текст (только Clipper)

Класс: C

Синтаксис. MEMOEDIT (⟨expC⟩, ⟨expN1⟩, ⟨expN2⟩, ⟨expN3⟩, ⟨expN4⟩, ⟨expL⟩])

Входные параметры : ⟨expC⟩ — начальная строчка меню

⟨expN⟩ — четыре координаты, определяющие верхний левый и нижний правый углы окна редактирования

⟨expL⟩ — флаг модификации: если текст разрешено редактировать — .T., если только просматривать — .F.

Функция возвращает : символьное выражение

Функция MEMOEDIT не включена в состав стандартных функций dBASE III Plus. В Clipper MEMOEDIT позволяет просматривать и/или редактировать переменные или атрибуты типа текст (memo). В Clipper переменные памяти могут содержать в себе текст произвольной длины, тип символьный там включает и текст. В dBASE III Plus переменные памяти не могут содержать текст произвольной длины.

Эта функция активизирует интерактивный режим экранного редактирования. Управляющие клавиши для редактирования:

Управляющие клавиши	Передвижение курсора
Вверх или Ctrl—E	На строку вверх
Вниз или Ctrl— C	На строку вниз
Влево или Ctrl — S	На символ влево
Вправо или Ctrl—D	На символ вправо
Ctrl—влево или Ctrl—A	На слово влево
Ctrl—вправо или Ctrl—F	На слово вправо
HOME	В начало текущей строки
END	В конец текущей строки
Ctrl—HOME	В начало меню
Ctrl—END	В конец меню
PgUp	В предыдущее окно редактирования
PgDn	В следующее окно редактирования
Ctrl—PgUp	В начало текущего окна
Ctrl—PgDn	В конец текущего окна

Ключи редактирования	
Ctrl—W	Конец редактирования
Esc	Отказ от редактирования
Ctrl—Y	Уничтожение текущей строки
Ctrl—T	Уничтожение слова справа от курсора
Ctrl—B	Переформирование меню к окну редактирования

Примеры.

```
* Для редактирования текущего атрибута типа memo:
REPLACE memo_field WITH MEMOEDIT (memo_field, 5, 10,
20, 69, .T.)
* Для просмотра текста без редактирования:
IF ' ' = MEMOEDIT (memo_field, 5, 10, 20, 69, .F.)
```

3.4.52. MESSAGE — текст ошибочного сообщения dBASE III Plus (только dBASE III Plus)

Класс: И
Синтаксис. MESSAGE ( )
Входные параметры : нет
Функция возвращает : символьную строку
Эта функция предназначена для вывода сообщения об ошибке на экран либо для запоминания этого сообщения в переменной памяти в зависимости от требований программы или оператора.
См. также ERROR ( ).

3.4.53. MIN — определение меньшего из двух чисел (dBASE III Plus и Clipper)

Класс: М
Синтаксис. MIN (⟨expN1⟩, ⟨expN2⟩)
Входные параметры : два числовых выражения
Функция возвращает : число
Эта функция сравнивает два заданных числовых выражения и возвращает меньшее из них. В Clipper функция MIN реализована на языке dBASE и находится в файле Extenddb.prg:

```
FUNCTION MIN
* Синтаксис. MIN (⟨expN1⟩, ⟨expN2⟩ )
* Функция возвращает : наименьшее из двух чисел
*
PARAMETERS cl_n1, cl_n2
RETURN IF (cl_n1 < cl _n2, cl_n1, cl_ n2)
```

Примеры.

```
STORE 7 TO t
STORE 12 TO t1
? MIN (t, t1)
7
```

См. также MAX ( ).

3.4.54. MOD — вычисление остатка от деления по модулю (dBASE III Plus и Clipper)

Класс: М
Синтаксис. MOD (⟨expN1⟩, ⟨expN2⟩)
Входные параметры : два числовых выражения
Функция возвращает : число
Эта функция производит деление по модулю.
Функция возвращает остаток от деления выражения ⟨expN1⟩ на выражение ⟨expN2⟩.
Формула вычисления остатка от деления по модулю следующая: ⟨expN1⟩ — floor (⟨expN1⟩,⟨expN2⟩) * ⟨expN2⟩, где floor( ) возвращает ближайшее целое, меньшее или равное своему аргументу. Функция floor не является функцией dBASE III Plus.
Функция dBASE III Plus INT ( ) отличается от floor ( ) тем, что просто отсекает любые знаки после десятичной запятой.

Если выражение ⟨expN2⟩ — положительное, то и результат функции MOD — положительная величина. Если выражение ⟨expN2⟩ — отрицательное, то результат MOD — отрицательная величина.
В Clipper функция MOD реализована на языке dBASE и находится в файле Extenddb. prg:

```
FUNCTION MOD
* Синтаксис. MOD (⟨expN1⟩, ⟨expN2⟩)
* Функция возвращает: остаток от деления ⟨expN1⟩ на ⟨expN2⟩
* Замечания по реализации. Различия между функцией MOD
* dBASE III Plus и оператором деления по модулю в Clipper приведены ниже и помечены ⟨— —⟩
```

	Clipper оператор		Функция dBASE III Plus
*	3 % 3 : : = 0.00		MOD (3, 3) : : = 0
*	3 % 2 : : = 1.00		MOD (3, 2) : : = 1
*	3 % 1 : : = 0.00		MOD (3, 1) : : = 0
*	3 % 0 : : = 0.00	⟨——⟩	MOD (3, 0) : : = 3
*	3 % —1 : : = 0.00		MOD (3, —1) : : = 0
*	3 % —2 : : = 1.00	⟨——⟩	MOD (3, —2) : : = —1
*	3 % —3 : : = 0.00		MOD (3, —3, : : = 0
*			
*	—3 % 3 : : = 0.00		MOD (—3, 3) : : = 0
*	—3 % 2 : : = —1.00	⟨——⟩	MOD (—3, 2) : : = 1
*	—3 % 1 : : = 0.00		MOD (—3, 1) : : = 0
*	—3 % 0 : : = 0.00	⟨——⟩	MOD (—3, 0) : : = —3
*	—3 % —1 : : = 0.00		MOD (—3, —1) : : = 0
*	—3 % —2 : : = 1.00		MOD (—3, —2) : : = —1
*	—3 % —3 : : = 0.00		MOD (—3, —3) : : = 0
*			
*	3 % 3 : : = 0.00		MOD (3, 3) : : = 0
*	2 % 3 : : = 2.00		MOD (2, 3) : : = 2
*	1 % 3 : : = 1.00		MOD (1, 3) : : = 1
*	0 % 3 : : = 0.00		MOD (0, 3) : : = 0
*	—1 % 3 : : = —1.00	⟨——⟩	MOD (—1, 3) : : = 2
*			
*	—2 % 3 : : = —2.00	⟨——⟩	MOD (—2, 3) : : = 1
*	—3 % 3 : : = 0.00		MOD (—3, 3) : : = 0
*			
*	3 % —3 : : = 0.00		MOD (3, —3) : : = 0
*	2 % —3 : : = 2.00	⟨——⟩	MOD (2, —3) : : = —1
*	1 % —3 : : = 1.00	⟨——⟩	MOD (1, —3) : : = —2
*	0 % —3 : : = 0.00		MOD (0, —3) : : = 0
*	—1 % —3 : : = —1.00		MOD (—1, —3) : : = —1
*	—2 % —3 : : = —2.00		MOD (—2, —3) : : = —2
*	—3 % —3 : : = 0.00		MOD (—3, —3) : : = 0
*			

```
PARAMETERS cl_num, cl_base
PRIVATE cl_result
cl_result = cl_num % cl_base
RETURN IF (cl_base = 0, cl_num,;
          IF (cl_result * cl_base <0, cl_result + cl_base, cl_result)
```

Примеры.

```
? MOD (14, 12)
2
? MOD (0,32)
0
? MOD (1, —3)
—2
? MOD (—2,3)
1
? MOD (—4, —3)
—1
```

3.4.55. MONTH — определение месяца года (dBASE III Plus и Clipper)

Класс: Д
Синтаксис. MONTH (⟨expD⟩)
Входные параметры : выражение типа дата
Функция возвращает : число

Эта функция определяет порядковый номер месяца года из заданного выражения типа дата.

Выражение типа дата может быть сконструировано из переменных памяти этого типа, значений атрибутов типа дата, результатов функций, возвращающих дату (например, функция DATE ( ) — возврат системной даты).

Примеры.

```
Пусть системная дата — 05/15/84
? MONTH (DATE ( ) )
5
STORE MONTH (DATE( ) ) TO mm
? mm
5
? TYPE («mm»)
N
```

См. также CMOUNTH ( ), DAY ( ), YEAR( ).

3.4.56. NDX — определение имени индексного файла (dBASE III Plus и Clipper)

Класс: И
Синтаксис. NDX (⟨expN⟩)
Входные параметры : число
Функция возвращает : символьную строку — имя индексного файла для dBASE III Plus
символьную строку «NTX ⟨expN⟩» для Clipper

В среде dBASE III Plus функция NDX возвращает имена активных индексных файлов в текущей рабочей зоне. Это позволяет программе манипулировать индексными файлами без явного указания их имен. Числовое выражение, задаваемое в функции NDX, указывает номер открытого индексного файла в списке индексных файлов текущей выбранной рабочей зоны. Это числовое выражение должно быть в диапазоне от 1 до 7.

Если в момент обращения функции NDX в указанной позиции нет открытого индексного файла, то функция NDX возвратит пустую строку.

В Clipper эта функция реализована на языке dBASE и находится в файле Extenddb.prg:

```
FUNCTION NDX
* Синтаксис. NDX ( ⟨expN⟩)
* Функция возвращает : строку 'NTX ⟨expN⟩'
*
PARAMETERS cl_1
RETURN 'NTX' + LTRIM (STR (cl_1))
```

Примеры.

Для того чтобы просмотреть список всех открытых индексных файлов, можно использовать следующую группу команд:

```
i = 1
Null = " "
DO WHILE Null < NDX (i) .AND. i < = 7
   ? NDX (i)
   i = i + 1
ENDDO
```

См. также функции DBF ( ), FIELD ( ), LUPDATE ( ), RECCOUNT ( ), RECSIZE ( );
команды SET INDEX, SET ORDER TO.

3.4.57. OS — определение имени операционной системы (dBASE III Plus и Clipper)

Класс: И
Синтаксис. OS ( )
Входные параметры : нет
Функция возвращает : символьную строку

Эта функция возвращает имя операционной среды для dBASE III Plus и Clipper. Информируя программу, под управлением какой операционной системы работает dBASE III Plus или Clipper, функция OS дает возможность создать программу, переместимую между DOS и UNIX.

В Clipper функция OS реализована на языке dBASE и находится в файле Extenddb.prg.

См. также DISKSPACE ( ), GETENV ( ), VERSION ( ).

3.4.58. PCOL — определение текущего столбца на устройстве печати (dBASE III Plus и Clipper)

Класс: И
Синтаксис. PCOL ( )
Входные параметры : нет
Функция возвращает : число

Эта функция определяет номер текущего столбца на печатающем устройстве. Из тела программы функция PCOL позволяет указывать местоположение будущей выводимой на печать строки символов.

В интерактивном режиме с установленным SET PRINT ON результат функции PCOL всегда 0.

Примеры.

Пусть требуется начать печать с заданной позиции по отношению к текущей позиции печати:

```
SET DEVICE TO PRINT
@ 1, PCOL ( ) + 5 SAY "Это пример относительной адресации печати"
SET DEVICE TO SCREEN
```

См. также PROW ( ), COL ( ), ROW ( ).

3.4.59. PROCLINE — определение номера строки исполняемой программы (только Clipper)

Класс: И
Синтаксис. PROCLINE( )
Входные параметры : нет
Функция возвращает : число

Функция PROCLINE в состав стандартных функций dBASE III Plus не включена. В Clipper эта функция возвращает текущий номер строки исходного текста исполняемой программы или процедуры.

3.4.60. PROCNAME — определение имени исполняемой программы или процедуры (только Clipper)

Класс: И
Синтаксис. PROCNAME ( )
Входные параметры : нет
Функция возвращает : символьную строку

В состав стандартных функций dBASE III Plus функция PROCNAME не включена. В Clipper функция PROCNAME возвращает имя исполняемой в данный момент программы или процедуры.

Таким образом, программа сама может информировать пользователя о ходе своего выполнения.

Пример.

```
? 'THE PROCEDURE BEING EXECUTED IS —'+ PROCNAME( )
```

3.4.61. PROW — определение текущей строки на устройстве печати (dBASE III Plus и Clipper)

Класс: И
Синтаксис. PROW( )
Входные параметры : нет
Функция возвращает : число

Эта функция позволяет определить текущую строку печати.

Примеры

Чтобы определить текущую строку печати:

```
? PROW ( )
10
x = PROW ( )
? x
10.
```


Пусть требуется напечатать "это пример относительной адресации" на пять строк ниже текущей строки печати (номер текущей строки должен быть меньше 249):

```
SET DEVICE TO PRINT
@ PROW ( ) + 5,10 SAY "Это пример относительной адресации"
SET DEVICE TO SCREEN
```

Отрицательная адресация в подобных случаях недопустима.
См. также @, COL ( ), PCOL ( ), ROW ( ).

3.4.62. READKEY (LASTKEY) — код последней нажатой клавиши (dBASE III Plus и Clipper)

Класс: В
Синтаксис. READKEY ( ) — для dBASE III Plus
LASTKEY ( ) — для Clipper
Входные параметры : нет
Функция возвращает : число

См. описание функции LASTKEY настоящего документа.

3.4.63. READVAR — определение имени текущей GET/MENU переменной (только Clipper)

Класс: И
Синтаксис. READVAR( )
Входные параметры : нет
Функция возвращает : символьную строку

В состав стандартных функций dBASE III Plus функция READVAR не включена. В Clipper функция READVAR возвращает имя текущей GET/MENU переменной либо пустую строку, если нет «подвешенных» полей ввода (список полей ввода пуст).

3.4.64. RECCOUNT (LASTREC) — количество кортежей в активном отношении (dBASE III Plus и Clipper)

Класс: И
Синтаксис. RECCOUNT ( ) — для dBASE III Plus
LASTREC ( ) — для Clipper
Входные параметры : нет
Функция возвращает : число

См. описание функции LASTREC.

3.4.65. RECNO — определение номера текущего кортежа активного отношения (dBASE III Plus и Clipper)

Класс: И
Синтаксис. RECNO( )
Входные параметры : нет
Функция возвращает : число

Эта функция определяет номер текущего кортежа в активном отношении. По умолчанию если в файле нет записей, то RECNO

возвращает 1 и EOF — .T. Если были попытки переместить указатель кортежей после последнего кортежа отношения и EOF —.T., то RECNO возвращает величину на единицу больше числа кортежей в отношении (т. е. RECCOUNT + 1). Если указатель кортежей установлен перед первым кортежем отношения и BOF — .T., то RECNO возвращает 1.

Примеры.

```
USE Clients
? RECNO ( )
1
SKIP — 1
? BOF ( )
.T.
? RECNO ( )
1
GO BOTTOM
? RECNO( )
49
SKIP
? EOF ( )
.T.
STORE RECNO ( ) TO t
? t
50
```

3.4.66. RECSIZE — определение длины кортежа (dBASE III Plus и Clipper)

Класс: И

Синтаксис. RECSIZE ( )

Входные параметры : нет

Функция возвращает : число

Эта функция определяет размер в байтах кортежа активного отношения. Если активного отношения в данный момент нет, то возвращается величина 0.

Использование функции RECSIZE позволяет программе манипулировать отношениями, бо́льшими по размеру, чем свободное дисковое пространство.

Замечания по реализации.

Рекомендуется использовать функцию RECSIZE совместно с функциями RECCOUNT и DISKSPACE для создания автоматических копий отношений на диске, чтобы убедиться в наличии свободного дискового пространства, достаточного для этих копий.

В Clipper функция RECSIZE реализована на языке СИ и находится в файле Extendc. c:

```
/**************************
* RECSIZE ( )
* Синтаксис. RECSIZE( )
*Функция возвращает : длину кортежа в байтах активного отношения
*/
RECSIZE ( )
{
    _retnl (DBF_OPEN ? REC_SIZE : 0L);
}
```


Примеры.

Пусть необходимо с жесткого диска перекопировать отношение File на дискету, установленную на дисководе B, и размер отношения превышает размер свободного дискового пространства на дискете. Следовательно, копирование будет производиться на несколько дискет. Следует учесть размер заголовка файла, который вычисляется по формуле

32 * количество атрибутов +35

Программа такого копирования выглядит следующим образом:

```
USE File
SET DEFAULT TO B
DO WHILE .NOT. EOF ( )
    WAIT "Insert new disk In drive B, and press a key. . ."
    COPY NEXT  (DISKSPACE ( ) — ⟨размер  заголовка⟩ ) / RE-
        CSIZE ( ); TO Backup
    SKIP
ENDDO
USE
```

См. также DBF ( ), DISKSPACE ( ), FIELD ( ), NDX ( ), RECCOUNT ( ).

3.4.67. REPLICATE — повторить символьное выражение (dBASE III Plus и Clipper)

Класс: С

Синтаксис. REPLICATE (⟨expC⟩, ⟨expN⟩)

Входные параметры : символьное выражение, числовое выражение

Функция возвращает : символьную строку

Эта функция повторяет заданное символьное выражение заданное количество раз. Результирующее символьное выражение не должно превышать 254 символа. В то же время числовое выражение ⟨expN⟩, задающее число повторов, разделенное на количество символов в ⟨expC⟩, должно быть числом, меньшим чем 254.

Примеры.

```
? REPLICATE ("AbC", 3)
AbCAbCAbC
```

Пусть необходимо построить графический эквивалент значений атрибутов отношения Tours:

```
USE Tours
LIST NEXT 5 Percent, REPLICATE («*», Percent)
10.000      **********
7.500       *******
5.000       *****
10.000      **********
7.50        *******
```

3.4.68. RIGHT — выбор подстроки от правого конца строки (dBASE III Plus и Clipper)

Класс: С
Синтаксис.: RIGHT (⟨expC⟩, ⟨expN⟩)
Входные параметры : символьное выражение, числовое выражение
Функция возвращает : символьную строку.

Эта функция выделяет заданное количество символов из заданного символьного выражения начиная справа. Применяется при необходимости просмотра хвостовой части строки либо совместно с функцией LTRIM для заполнения какими-либо символами ведущих пробелов, которые получаются вследствие функции STR.

Если числовое выражение ⟨expN⟩ равно 0 или отрицательно, то функция RIGHT вернет пустую строку. Если числовое выражение ⟨expN⟩ больше длины исходной строки, то функция RIGHT вернет исходную строку.

В Clipper функция RIGHT реализована на языке dBASE и находится в файле Extenddb. prg:

```
FUNCTION RIGHT
* Синтаксис. RIGHT (⟨expC⟩,⟨expN⟩)
* Функция возвращает : ⟨expN⟩ символов выражения ⟨expC⟩ начиная справа
*
PARAMETERS cl_str, cl_len
RETURN SUBSTR (cl_str, LEN (cl_str) — cl_len + 1)
```

Примеры.

```
? RIGHT («abcdel», 3)
del
Padding = «**********»
? RIGHT (Padding + LTRIM (STR (2334,10)), 10)
*****2334
```

См. также LEFT ( ), LTRIM ( ), STUFF ( ), SUBSTR ( ), RTRIM ( ), TRIM ( ).

3.4.69. ROUND — округление числа (dBASE III Plus и Clipper)

Класс: М
Синтаксис. ROUND (⟨expN1⟩, ⟨expN2⟩)
Входные параметры : два числовых выражения
Функция возвращает : число

Эта функция округляет заданное числовое выражение ⟨expN1⟩ до заданного количества знаков после запятой ⟨expN2⟩. Если выражение ⟨expN2⟩ отрицательно, то функция ROUND вернет округленное целое число.

Примеры.

```
? ROUND (14.746321,2)
14.750000
? ROUND (17.321111,3)
17.321
```



```
STORE 10.7654321 TO x
? ROUND (x, 0)
11.0000000
? ROUND (x, 1)
10.8
? ROUND (14911, —3)
15000
```

3.4.70. ROW — определение номера строки текущего положения экранного курсора (dBASE III Plus и Clipper)

Класс: И
Синтаксис. ROW ( )
Входные параметры : нет
Функция возвращает : число

Эта функция определяет местоположение курсора на экране, возвращая номер строки экрана, на которой в данный момент стоит курсор. Наиболее часто эта функция используется для относительной адресации курсора.

Примеры.

```
? ROW ( )
1
@ ROW ( ) + 5,1 SAY «это пример относительной адресации»
```

См. также @, COL ( ), PROW ( ), PCOL ( ).

3.4.71. RTRIM — удаление хвостовых пробелов (dBASE III Plus и Clipper)

Класс: С
Синтаксис. RTRIM (⟨exp C⟩)
Входные параметры : символьное выражение
Функция возвращает : символьную строку

Эта функция удаляет хвостовые пробелы в заданном символьном выражении. По своим действиям аналогична функции TRIM. В Clipper функция RTRIM реализована на языке dBASE и находится в файле Extenddb.prg:

```
FUNCTION RTRIM
* Синтаксис. RTRIM (⟨expC⟩)
* Функция возвращает: ⟨expC⟩ без хвостовых пробелов
*
PARAMETERS cl_str
RETURN TRIM (cl_str)
```

См. также LEFT ( ), LTRIM ( ), RIGHT ( ), TRIM ( ).

3.4.72. SECONDS — определение системного времени в секундах в отсчете от полночи (только Clipper)

Класс: Д
Синтаксис. SECONDS ( )
Входные параметры : нет

Функция возвращает : число

В состав стандартных функций dBASE III Plus эта функция не включена. В Clipper функция SECONDS возвращает системное время в формате ⟨секунды⟩. ⟨сотые доли секунды⟩. Отсчет секунд производится начиная от полночи.

Диапазон возвращаемого значения — от 0 до 86399.

3.4.73. SECS — преобразование строки времени в секунды (только Clipper)

Класс: П

Синтаксис. SECS (⟨expT⟩)

Входные параметры : символьная строка времени в формате ⟨17 : 24 : 17⟩

Функция возвращает : число

Эта функция возвращает число секунд, эквивалентное заданной строке времени. В состав стандартных функций dBASE III Plus функция SECS не включена. В Clipper эта функция реализована на языке dBASE и находится в файле Extenddb.prg:

```
FUNCTION SECS
* Синтаксис. SECS ( ⟨expT⟩ )
* Функция возвращает : количество секунд, эквивалентное ⟨expT⟩
*                          60 с   = 1 мин
*                        3600 с   = 1 ч
*                       86400 с   = 1 сут
*
PARAMETERS cl_time
RETURN VAL(                cl_time    ) * 3600 + ;
       VAL (SUBSTR (cl_time, 4)) *   60 + ;
       VAL (SUBSTR (cl_time, 7))
```

3.4.74. SELECT — определение номера активной рабочей области (только Clipper)

Класс: И

Синтаксис. SELECT ( )

Входные параметры : нет

Функция возвращает : число

В состав стандартных функций dBASE III Plus функция SELECT не включена. В Clipper эта функция возвращает номер активной рабочей области.

Примеры.

```
IF SELECT ( ) = 1
  @ 23,0 SAY 'PLEASE ENTER MAIN DATA'
ENDIF
IF SELECT ( ) = 2
  @ 23,0 SAY 'PLEASE ENTER SECONDARY DATA'
ENDIF
```


3.4.75. SOUNDEX — преобразование слова (имени) в SOUND код (только Clipper)

Класс: П

Синтаксис. SOUNDEX (⟨expC⟩)

Входные параметры : символьная строка

Функция возвращает : «SOUND» код символьной строки ⟨expC⟩ в формате A9999

В состав стандартных функций dBASE III Plus не включена. В Clipper функция SOUNDEX возвращает 5-символьный код слова (обычно имени) и используется для поиска в некоторых приложениях имени, подобного закодированному, но которое могло быть введено в базу с орфографическими ошибками.

Реализована на языке dBASE и находится в файле Extenddb.prg:

```
FUNCTION SOUNDEX
* Синтаксис : SOUNDEX ( ⟨expC⟩ )
* Функция возвращает : код имени в форме A9999
*
PARAMETERS cl_name
PRIVATE cl_name, cl_code, cl_pointer
cl_name = UPPER (cl_name)
cl_code = SUBSTR (cl_name, 1,1)
cl_pointer = 2
DO WHILE cl_pointer <= LEN (cl_name) .AND. LEN (cl_code) < 5
   DO CASE
     CASE SUBSTR (cl_name, cl_pointer, 1) $ 'BEPV'
       cl_code = cl_code + IF (SUBSTR (cl_code, LEN
       (cl_code), 1) #' 1', '1', [ ])
       CASE SUBSTR (cl_name, cl_pointer, 1) $ 'CGJKQSXZ'
       cl_code = cl_code + IF (SUBSTR (cl_code, LEN
       (cl_code), 1) #'2', '2', [ ])
     CASE SUBSTR (cl_name, cl_pointer, 1) $ 'DT'
       cl_code = cl_code + IF (SUBSTR (cl_code, LEN
       (cl_code), 1) #' 3', '3', [ ])
     CASE SUBSTR (cl_name, cl_pointer, 1) $ 'L'
       cl_code = cl_code + IF (SUBSTR (cl_code, LEN (cl_
       code), 1) #'4','4', [ ])
     CASE SUBSTR (cl_name, cl_pointer, 1) $ 'MN'
       cl_code = cl_code + IF (SUBSTR (cl_code, LEN
       (cl_code), 1) #'5', '5', [ ])
     CASE SUBSTR (cl_name, cl_pointer, 1) $ 'R'
       cl_code = cl_code + IF (SUBSTR (cl_code, LEN (cl_code),
       1) #' 6' , '6', [ ])
     ENDCASE
       cl_pointer = cl_pointer + 1
ENDDO
RETURN cl_code + TRIM (SUBSTR ( '0000' , LEN (cl_code) ))
```

Данная функция может рассматриваться в качестве примера для написания собственной подобной функции.

3.4.76. SPACE — построение строки пробелов заданной длины (dBASE III Plus и Clipper)

Класс: С
Синтаксис. SPACE (⟨expN⟩)
Входные параметры : число
Функция возвращает : символьную строку
Эта функция служит для формирования символьной строки заданной длины и заполненной пробелами.
Максимальный размер строки — 254 пробела.
Примеры.
Пусть необходимо построить пустую строку длиной 20, ограниченную символами "*".

```
STORE SPACE (20) TO blanks
? "*" + blanks + "*"
*                    *
```

3.4.77. SQRT — вычисление квадратного корня (dBASE III Plus и Clipper)

Класс: М
Синтаксис. SQPT (⟨expN⟩)
Входные параметры : числовое выражение
Функция возвращает: число
Эта функция вычисляет квадратный корень из заданного числового выражения. Квадратный корень вычисляется функцией SQRT только для положительных чисел.
Функция SQRT по умолчанию возвращает результат в форме числа с десятичной запятой с фиксированным числом десятичных позиций.

Примеры.

```
? SQRT (4)
2.00
? SQRT (2 * 2)
2.00
n = 4.000
? SQRT (n)
2.000
```

См. также команды SET DECIMAL, SET FIXED.

3.4.78. STR — преобразование числа в символьную строку (dBASE III Plus и Clipper)

Класс: П
Синтаксис. STR(⟨expN⟩, [⟨length⟩], [⟨decimal⟩])
Входные параметры : три числовых выражения
Функция возвращает: символьную строку
Функция STR преобразует числовое выражение в символьную строку.
Если длина строки явно не указана, то по умолчанию длина результирующей строки равна 10. Если не указано количество позиций после десятичной точки, заданное числовое выражение округляется до целого. Длина результирующей символьной строки включает в себя десятичную точку и не включает знак. Если задается длина меньше, чем количество десятичных знаков в заданном числовом выражении, то функция STR возвращает символ *. Если задается меньшее количество десятичных знаков после запятой, чем в исходном числовом выражении, функция STR округляет результат до заданного числа десятичных знаков.

Примеры.

```
x= 11.14
? x
11.14
y = STR (x * 10,5)
? y
111
y = STR (x * 10, 5, 2)
? y
111,4
STORE STR (x * 10, 6, 2) TO y
? y
111.40
```

См. также VAL ( ).

3.4.79. STRZERO — преобразование числа в символьную строку с ведущими нулями вместо пробелов (только Clipper)

Класс: П
Синтаксис. STR (⟨expN⟩, [⟨length⟩],
[⟨decimal⟩])
Входные параметры : от одного до трех числовых выражений
Функция возвращает : символьную строку
В состав стандартных функций dBASE III Plus функция STRZERO не включена. В Clipper функция STRZERO осуществляет преобразования, подобные осуществляемым функцией STR, но ведущие пробелы заменяет символом «0». Реализована на языке dBASE и находится в файле Extenddb .prg:

```
FUNCTION STRZERO
* Синтаксис. STR( ⟨expN⟩, [ ⟨length⟩ ], [ ⟨decimal⟩ ])
* Функция возвращает : STR ( ) от ⟨expN⟩ с ведущими нулями
*
PARAMETERS cl_num, cl_len, cl_dec
PRIVATE cl_str
DO CASE
   CASE TYPE ('cl_des') # 'U'
        cl_str = STR (cl_num, cl_len, cl_des)
   CASE TYPE ('cl_len') # 'U'
        cl_str = STR (cl_num, cl_len)
   OTHERWISE
        cl_str = STR (cl_num)
ENDCASE
IF '—' $ cl_str                  && отрицательное число
   RETURN '—' + REPLICATE('0', LEN (cl_str) — LEN
              (TRIM (cl_str))) +;
              SUBSTR (cl_str, AT ('—' ,cl_str) + 1)
```

```
ELSE
    RETURN REPLICATE('—' ,LEN (cl_str) — LEN (LTRIM
           (cl_str))) +;
           LTRIM (cl_str)
ENDIF
```

3.4.80. STUFF — замена части строки (dBASE III Plus и Clipper)

Класс: С

Синтаксис. STUFF (⟨expC1⟩ , ⟨expN1⟩, ⟨expN2⟩, ⟨expC2⟩)

Входные параметры : два символьных и два числовых выражения

Функция возвращает : символьную строку

Эта функция замещает в заданной символьной строке ⟨expC1⟩ заданное количество символов ⟨expN2⟩ с указанной позиции ⟨expN1⟩ на другую заданную последовательность символов ⟨expC2⟩.

Если заданное количество замещаемых символов равно нулю, то символьная строка ⟨expC2⟩ просто вставляется в исходную строку с указанной позиции ⟨expN1⟩. Если в качестве ⟨expC2⟩ задана пустая строка, то из исходной строки вычищается заданное количество символов начиная с указанной позиции. Длина результирующей строки равна длине исходной строки только в том случае, когда количество замещаемых символов равно длине выражения ⟨expC2⟩.

В Clipper функция STUFF реализована на языке dBASE и находится в файле Extenddb .prg:

```
FUNCTION STUFF
* Синтаксис. STUFF ( ⟨expC1⟩, ⟨expN1⟩, ⟨expN2⟩, ⟨expC2⟩ )
* Функция возвращает : символьную строку ⟨expC1⟩ с фрагментом начиная с ⟨expN1⟩ длиной ⟨expN2⟩, замененным на ⟨expC2⟩
PARAMETERS cl_string, cl_start, cl_len, cl_replace
RETURN SUBSTR (cl_string, 1, cl_start — 1) + cl_replace+;
       SUBSTR (cl_string, cl_start + cl_len)
```

Примеры.

```
? STUFF ("abc", 2, 1, "xyz")
axyzc
? STUFF ("abc", 2, 1, "")          && удаление второго символа
ac
? STUFF ("abc", 2, 0, "xyz")
axyzbc
```

См. также LEFT ( ), RIGHT ( ), SUBSTR ( ).

3.4.81. SUBSTR — выделение подстроки в строке (dBASE III Plus и Clipper)

Класс: С

Синтаксис. SUBSTR (⟨expC⟩, ⟨expN1⟩, ⟨expN2⟩)

Входные параметры : символьное выражение, два числовых выражения

Функция возвращает : символьную строку

Эта функция выделяет в заданной символьной строке подстроку заданной длины начиная с заданной позиции. Если количество символов, задающее длину подстроки, не задано, то выделяется подстрока начиная с указанной позиции и заканчивая концом исходной строки.

В dBASE III Plus в том случае, если заданное количество символов больше, чем количество символов от заданной позиции до конца исходной строки, выделяется подстрока от указанной позиции до конца исходной строки. В Clipper в том случае, если начальная позиция или длина подстроки указывают за конец исходной строки, всегда возвращается пустая строка.

Номер начальной позиции обязательно должен быть положительным числом.

Примеры.

Пусть необходимо выделить подстроку "59" из строки "1958 1959 1960"

```
? SUBSTR ("1958 1959 1960", 8,2)
59
STORE "abcd123456789" TO x
? SUBSTR (x, 5, 20)
123456789
```

См. также AT ( ), LEFT ( ), STR ( ), STUFF ( ).

3.4.82. TIME — определение системного времени (dBASE III Plus и Clipper)

Класс: Д

Синтаксис. TIME ( )

Входные параметры : нет

Функция возвращает : символьную строку

Эта функция возвращает системное время в виде символьной строки формата чч : мм : сс (например, 17 : 24 : 23).

Примеры.

```
? TIME ( )
21 : 22 : 23
t = TIME ( )
? t
21 : 22 : 23
```

3.4.83. TRANSFORM — вывод строки в задаваемом PICTURE формате (dBASE III Plus и Clipper)

Класс: С

Синтаксис. TRANSFORM (⟨expI⟩, ⟨expC⟩)

Входные параметры : символьная строка, шаблон вывода

Функция возвращает : символьную строку

Эта функция позволяет навязывать для выводимой информации определенный заданный PICTURE формат без использования @... SAY команды.

Символьное выражение ⟨expC⟩ задает формат вывода для выражения ⟨expI⟩. Функция TRANSFORM устанавливает формат для таких команд, как ?, ??, DISPLAY, LABEL, LIST и REPORT. Для более подробной информации о PICTURE формате следует обратиться к описанию команды @.

Примеры

Пусть необходимо вывести содержимое атрибута, содержащего символьное имя "Москва", вразбивку:

```
USE Filename
LIST NEXT TRANSFORM (Name, "@R X X X X X X X X X X
X X X X")
М О С К В А
```

См. также @. . .PICTURE.

3.4.84. TRIM — удаление хвостовых пробелов в строке (dBASE III Plus и Clipper)

Класс: С

Синтаксис. TRIM (⟨expC⟩)

Входные параметры : символьное выражение

Функция возвращает : символьную строку

Эта функция убирает хвостовые пробелы в заданном символьном выражении.

Команды LIST и DISPLAY TRIM (⟨атрибут1⟩), TRIM (⟨атрибут2⟩) не подавляют пробелы между выводимыми атрибутами. Чтобы эти пробелы удалялись при командах LIST и DISPLAY, необходимо использовать функцию TRIM следующим образом: TRIM (⟨ атрибут1 ⟩) + (⟨атрибут2⟩). Однако длина выводимого выражения будет комбинацией длин атрибутов, составляющих это выражение.

Примеры.

Пусть отношение Filename имеет атрибуты Place и Address:

```
USE Filename
? Place, Address
МОСКВА      КИЕВ
? TRIM (Place), Address
МОСКВА      КИЕВ
  ? TRIM (Place) + "," +Address
МОСКВА, КИЕВ
```

См. также LTRIM ( ), RTRIM ( ).

3.4.85. TSTRING — формирование строки времени из числа секунд (только Clipper)

Класс: П

Синтаксис. TSTRING (⟨expN⟩)

Входные параметры : числовое выражение

Функция возвращает : символьную строку в формате «чч:мм:сс»

В состав стандартных функций dBASE III Plus не включена. В Clipper функция TSTRING формирует символьную строку формата "чч : мм : сс" из задаваемого числового выражения.

Реализована на языке dBASE и находится в файле Extenddbprg:

```
FUNCTION TSTRING
* Синтаксис. TSTRING ( ⟨expN⟩ )
* Функция возвращает : строку времени, эквивалентную числу
секунд
*
PARAMETERS cl_secs
RETURN STRZERO ( INT ( MOD (cl_cecs/3600,24) ), 2,0 )+':';
       STRZERO ( INT ( MOD (cl_secs/ 60,60) ), 2,0 ) + ':';
       STRZERO ( INT ( MOD (cl_secs      ,60) ), 2,0 )
```


3.4.86. TYPE — определение типа выражения (dBASE III Plus и Clipper)

Класс: И

Синтаксис. TYPE (⟨expC⟩)

Входные параметры : выражение

Функция возвращает : символ

Эта функция возвращает тип задаваемого выражения.

Возможные типы: С — символьное выражение; N — числовое выражение; L — логическое выражение; M — выражение типа memo (текст); U — необъявленная переменная; A — массив (для Clipper).

Часто функция TYPE применяется для проверки существования переменной (т. е. проверки, объявлена ли она).

Примеры.

```
? TYPE ("test")
U
STORE "testing" TO test
? TYPE ("test")
C
```

3.4.87. UPDATE — проверка изменения данных в процессе экранного редактирования (только Clipper)

Класс: Т

Синтаксис. UPDATE ( )

Входные параметры : нет

Функция возвращает : логическую величину

В состав стандартных функций dBASE III Plus не включена. В Clipper функция UPDATE ( ) осуществляет тестирование модификации данных, если оно имело место в процессе экранного редактирования.

Функция UPDATE возвращает .T., если были проведены какие-либо изменения данных при экранном редактировании, в противном случае возвращается .F.

3.4.88. UPPER — преобразование строчных латинских букв в прописные (dBASE III Plus и Clipper)

Класс: П

Синтаксис. UPPER (⟨expC⟩)

Входные параметры : символьное выражение

Функция возвращает : символьную строку

Эта функция преобразует строчные буквы латинского алфавита в прописные.

Примеры.

```
? UPPER ("This is a cat")
THIS IS A CAT
STORE "This is a cat" TO x
? UPPER (x)
THIS IS A CAT
```

См. также ISLOWER ( ), ISUPPER ( ), LOWER ( ).

3.4.89. VAL — преобразование символьной строки в число (dBASE III Plus и Clipper)

Класс: П
Синтаксис. VAL (⟨expC⟩)
Входные параметры : символьное выражение
Функция возвращает : число

Эта функция преобразует символьное выражение в числовое. Если первые символы символьного выражения — не цифры, то функция VAL возвращает величину 0.

Функция VAL осуществляет преобразование с точностью, заданной командой SET DECIMALS.

Примеры.

```
? VAL ("ABC")
0
? VAL («123.45»)
123.45
SET DECIMALS TO 1
STORE «123.45» TO x
? x
123.45
? VAL (x)
123.5
```

См. также SET DECIMALS, STR ( ).

3.4.90. VERSION — определение версии dBASE III Plus/Clipper (dBASE III Plus и Clipper)

Класс: И
Синтаксис. VERSION ( )
Входные параметры : нет
Функция возвращает : символьную строку

Эта функция возвращает символьную строку с названием версии активизированной в данный момент системы dBASE III Plus или Clipper.

В Clipper функция VERSION реализована на языке dBASE и находится в файле Extenddb .prg:

```
FUNCTION VERSION
* Синтаксис. VERSION ( )
* Функция возвращает : название версии dBASE III Plus или Clipper
RETURN "Clipper, Fall '85"
```

См. также DISKSPACE ( ), GETENV ( ), OS ( ).

3.4.91. WORD — преобразование чисел к короткой форме при передаче параметров (только Clipper)

Класс: П
Синтаксис. CALL ⟨process⟩ WITH WORD (⟨expN⟩)
Входные параметры : числовое выражение
Функция возвращает : число

В состав стандартных функций dBASE III Plus не включена. В Clipper функция WORD осуществляет преобразование передаваемых командой CALL числовых параметров из чисел типа DOUBLE в числа типа INT (имеются в виду стандартные типы языка СИ — здесь соответственно 4 и 2 байта).

Если значение ⟨expN⟩ не превышает + —32K, то нет необходимости передавать длинный параметр, и здесь уместно пресбразование WORD. Нельзя использовать функцию WORD для передачи чисел, превышающих этот диапазон.

3.4.92. YEAR — определение года (dBASE III Plus и Clipper)

Класс: Д
Синтаксис. YEAR (⟨expD⟩)
Входные параметры : выражение типа «дата»
Функция возвращает : число

Эта функция определяет год из заданного выражения типа дата.

Примеры.

Пусть системная дата 05/15/84

```
? YEAR (DATE ( ))
1984
STORE YEAR (DATE ( )) TO d
? d
1984
? TYPE (d)
N
```

Можно индексировать отношения по дате и символьным ключам одновременно. Пусть необходимо проиндексировать отношение по атрибуту datel в хронологическом порядке и иметь атрибут namel в алфавитном порядке для каждой даты:

```
INDEX ON STR (YEAR (date), 4) + STR (MOUNTH (date),2) +;
         STR (DAY (date), 2) ) + name
```

Часть II

ИНТЕГРИРОВАННЫЙ ПАКЕТ KNOWLEDGEMAN

Глава 1

ВВЕДЕНИЕ в КМап

1.1. Общая характеристика

Интегрированный пакет KnowledgeMan (далее сокращенно КМап) является программной системой, позволяющей объединить в рамках единого универсального интерфейса пользователя такие традиционные приложения ПЭВМ, как текстовая обработка, работа с расчетной (табличной) информацией, реляционные базы данных и др., обеспечиваемые ранее независимыми программными пакетами типа редакторов текстов, процессоров табличной информации, реляционных СУБД и т. п.

Одной из основных причин, требовавших интеграции этих возможностей, явилось массовое применение ПЭВМ в области автоматизации учрежденческих работ, где в первую очередь возникает проблема подготовки разнообразных документов на машинных носителях в сочетании с возможностью вывода их на бумажные носители. Документы, циркулирующие в сфере учрежденческой деятельности, в основном включают текстовую и табличную (с расчетными числовыми данными) информацию. Весьма полезно для различных числовых рядов (тенденции рынка, динамика итоговых показателей по временным интервалам и пр.) иметь некоторое графическое представление в виде диаграмм, графиков и т. п. (так называемая деловая графика). Необходима также структурированная информация, поддерживаемая СУБД (например, о множестве самих документов). Организация хранения таких документов в виде отдельных файлов (текстовых, табличных, отношений реляционных баз данных, графиков и пр.), обрабатываемых отдельными пакетами программ, неудобна, поскольку требует существенных знаний по отдельным пакетам и затрудняет обмен информацией, содержащейся в разных файлах.

Пакет КМап включает следующие функциональные возможности:

управление базами данных (БД);
обработку крупноформатных электронных таблиц (ЭТ);
деловую графику;
обработку текстов;
выполнение функций настольного калькулятора;
проведение краткого статистического анализа информации;
создание прикладных систем с использованием внутреннего языка программирования.

Интеграция программного обеспечения позволяет затрачивать минимум времени на связь различных компонентов, используя данные различных форматов без выхода из среды программного продукта. Это обеспечивает обработку данных, представленных в различной форме, в едином технологическом потоке без загрузки пользователя действиями по изменению формата данных и упрощает переход из одной функциональной части пакета в другую.

Пакет обеспечивает следующие уровни взаимодействия (интерфейса) пользователя, ориентированные на различную профессиональную подготовку:

меню-ориентированный интерфейс, рассчитанный на начинающего пользователя. Он предлагает достаточное подмножество функций КМап для выбора способов представления информации и ее обработки;

командный язык, используемый прикладным программистом и представляющий пользователю весь спектр возможностей КМап для разработки конкретных приложений путем объединения команд в процедуры;

язык информационных запросов, близкий к естественному английскому, используемый конечным пользователем при непосредственной работе с пакетом;

интерфейс законченных приложений, ориентированный на конечного пользователя (при наличии этих приложений на языке команд КМап);

программный интерфейс путем вызова функций пакета из программ на языке СИ.

1.2. Структура КМап

На рис. 4 приведена структура интегрированного пакета, отражающая взаимодействие основных компонентов. Ядром интегрированного пакета является процессор команд КМап, объединяющий взаимодействие программных (активных) и информационных (пассивных) компонентов.

Работа пользователя в любом из рассмотренных интерфейсов КМап интерпретируется ядром как набор выполняемых команд, активизирующих выбираемые компоненты. Процессор команд управляет работой следующих элементов пакета:

СУБД;
процессора электронных таблиц (ЭТ);
процессора деловой графики;
процессора экранных форм;
текстового процессора.

В функциональном смысле текстовый процессор может рассматриваться как самостоятельный компонент для создания текстовых документов, а также как инструмент создания программ на командном языке. Отдельным элементом исполнительного уровня является генератор отчетов, который обеспечивает известные для СУБД возможности вывода форматированной информации в виде отчета, имеющего итоговые строки.

СУБД обеспечивает возможность работы пользователя с информацией, хранимой в отношениях реляционных БД.

Процессор ЭТ хранит данные и результаты их обработки в виде ЭТ.

Процессор деловой графики позволяет пользователю получить наглядное (диаграммное) представление числовой информации, хранимой в ЭТ и в отношениях БД.

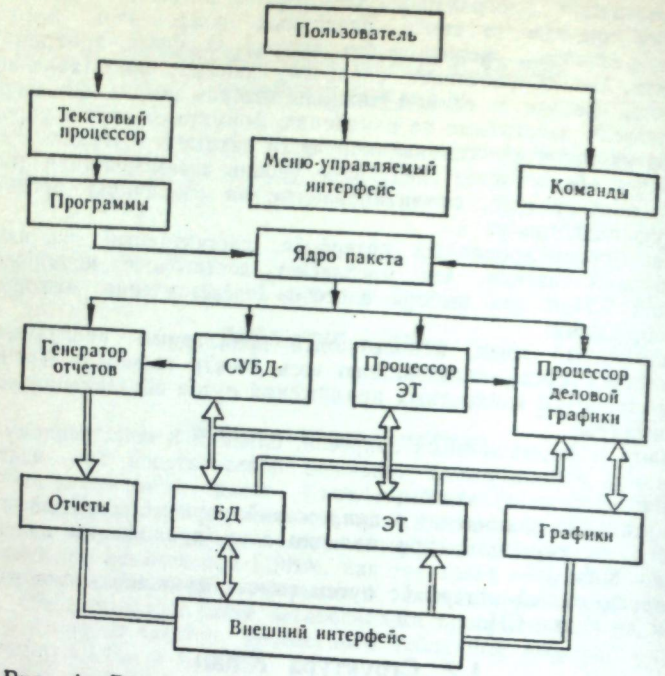


Рис. 4. Взаимодействие основных компонентов пакета.

Пакет КМап обеспечивает также некоторый уровень внешнего информационного интерфейса, т. е. взаимодействия на уровне обмена файлами, подготовленными и/или передаваемыми для дальнейшей обработки другими пакетами.

Интеграция различных программных и информационных компонентов позволяет некоторым активным элементам, например СУБД или процессору ЭТ, производить обмен информацией между собой и операционной средой, в которой работает КМап. Так, данные, хранимые в БД, иногда удобно обрабатывать в виде ЭТ, а полученные результаты опять записывать в БД. И наоборот, для заполнения ЭТ можно использовать информацию, хранимую в БД или подготовленную другими пакетами в стандартном системном формате.

Совокупность команд КМап — это универсальный язык программирования, включающий следующие группы команд управления и связи рассмотренных компонентов:

команды определения отношений БД;
команды ввода и модификации данных в отношениях;
команды выборки данных;
команды действий над отношениями;
команды управления средой (состоянием процессоров);
команды статистической обработки;
команды управления ЭТ;
команды создания и управления выходными формами (отчетами);
команды — операторы управления для создания процедур;
команды, определяемые пользователем для расширения языка КМап.

1.3. Основные элементы КМап-машины

1.3.1. Структуры памяти

Пакет КМап как система обработки данных может рассматриваться в виде некоторой виртуальной КМап-машины, имеющей некоторые логические представления основных узлов любой ЭВМ, т. е. оперативной и внешней памяти, процессора и систем ввода — вывода.

Как указывалось, в качестве основных информационных объектов, т. е. пассивных компонентов пакета, которые сохраняются во внешней памяти ПЭВМ как самостоятельные поименованные единицы, используются текстовые файлы, таблицы, отношения БД и графики.

Текстовые файлы как структуры хранения и обмена удовлетворяют требованиям стандартного системного формата операционной системы ПЭВМ. С логической точки зрения текстовые файлы, имеющие в пакете стандартные расширения, могут быть трех типов: .txt, .dat и .ipf.

Первый тип (.txt) представляет собой структуру для создания и хранения текстовых документов — таких, как отчет, статья, записка и т. п. Подобный документ имеет некоторую логическую и физическую структуры. Он представляет собой последовательность строк, имеющих заданную длину (в символах), может иметь абзацные отступы, позиции табуляции, а также быть разбит на страницы с заданным количеством строк. Эти понятия фиксируются в виде некоторого состояния текстового файла, в результате чего КМап автоматически выполняет функции переноса, равномерного расположения текста в строке, разбиения на страницы и др. Пользователь видит текстовый документ через «окно», предоставляемое на экране дисплея, при этом документ как некоторая поверхность может превышать размеры окна и по вертикали, и по горизонтали. Соответствующими командами можно осуществлять вертикальное и горизонтальное перемещение (скролинг) окна по поверхности документа. При выводе на печать документ также характеризуется его расположением на бумажном носителе, т. е. отступом слева, межстрочным расстоянием и т. д.

Следующий тип текстового файла (.dat) используется в основном для служебных целей обмена с другими пакетами и соответствует стандартному представлению текста в коде ASCII.

Третий тип текстового файла (.ipf) используется для создания текстов программ на языке КМап.

Электронные таблицы предназначены для представления и обработки информации, имеющей привычную для человека табличную форму, что позволяет проводить бухгалтерские и другие экономические расчеты, а также решать задачи планирования, прогнозирования и оптимизации.

Табличный файл имеет стандартное расширение .iss и как структура данных может быть представлен в виде прямоугольной матрицы. Строки таблицы нумеруются последовательными натуральными числами, а столбцы — буквами: A, B, C... Элемент матрицы называется ячейкой и может принимать числовые и символьные значения, а также содержать формулу (выражение). Ячейка обозначается своими координатами, например B5 — ячейка на пересечении 5-й строки и столбца B. Формула представляет собой выражение, составленное из констант, обозначений ячеек, ряда

общеупотребительных математических функций, связанных знаками арифметических операций. Например,

SUM (C3, C6) + SUM (A2, E2) — 1

означает вычисление суммы смежных ячеек, находящихся в столбце С с 3-й по 6-ю строку, и ячеек, расположенных во 2-й строке в столбцах от А до Е, минус единица. Более строгое определение выражения см. ниже.

Таким образом, таблица является особой структурой, сохраняющей признаки структуры данных и управляющей структуры. В зависимости от применяемых команд она может представлять собой данные или своеобразную программу вычислений. В этом случае ячейки, содержащие формулы, представляют собой значения — результаты вычислений по заданным формулам. Поскольку в формулах могут быть ссылки (имена ячеек) на другие формулы, то порядок их вычисления определяется сверху вниз, слева направо, что дает однозначный эффект.

Следующий тип файла представляет собой отношение реляционной СУБД и имеет стандартное расширение .itb. В целом представление отношения аналогично рассмотренному в первой части отношению СУБД dBASE III Plus, однако типы значений полей записи ограничены используемыми в KMan символьными, числовыми и логическими значениями. Аналогично dBASE III Plus отношения могут быть проиндексированы, что обеспечивает упорядоченность записей файла отношения по заданному индексному выражению. В результате индексации создается файл со стандартным расширением .ind. Для одного файла отношения может быть построено несколько индексов.

Графические образы диаграмм, полученных по числовым рядам, хранятся в файлах, имеющих стандартное расширение .plt.

Обработка содержимого рассмотренных компонентов (информационных структур во внешней памяти) предполагает организацию доступа к ним в оперативной памяти ПЭВМ. Для KMan-машины модель ее оперативной памяти представлена следующими структурами данных:

скалярные переменные;
прямоугольные массивы;
ячейки доступной (открытой) электронной таблицы;
поля записей доступных (открытых) отношений БД;
плоскость доступного текстового файла.

Скалярные переменные и элементы массивов имеют буферное назначение (они используются для хранения промежуточных, временных результатов) и могут принимать символьные, числовые и логические значения основных элементов хранения данных, т. е. ячеек ЭТ, полей отношений БД и др. Так, например, для текстовых документов предусмотрена возможность вставки в текст некоторых условных обозначений — переменных подстановки (макроподстановки). Макропеременные могут принимать значения, содержащиеся в таблицах или отношениях БД. Имеется режим вывода документа, при котором происходит подстановка текущего значения переменной в месте ее вхождения в текст. Таким образом удобно организовать деловую переписку, при которой некоторая заготовка письма, рассылаемого по многим адресам, сохраняется в виде текстового файла с макропеременной адреса. Адреса могут храниться, например, в отношении БД. Цикл вывода этого текстового файла с предварительным присваиванием макропеременной соответствующего адреса из

БД обеспечивает эффективную организацию подобной процедуры делопроизводства.

Скалярные переменные обозначаются именами (идентификаторами) и вводятся при выполнении команды присваивания вида

LET ⟨имя переменной⟩ = ⟨выражение⟩

Массивы вводятся командой определения массива

DIM ⟨имя массива⟩ (⟨размерность⟩)

Более строго понятия имен, размерности и выражения будут рассмотрены ниже.

Необходимо указать еще так называемые служебные переменные, которые используются KMan-машиной для хранения результатов ряда операций. Доступ к этим переменным односторонний, т. е. можно только прочитать значения этих переменных.

Отметим следующие различия между таблицами и отношениями реляционных баз данных при внешнем сходстве матричного представления информации для пользователя. По способу хранения информации и доступа к ней таблица представляет собой некоторое единое целое (возможно, с ограничениями по объему оперативной памяти), в то время как текущий доступ к отношению БД осуществляется только в рамках записи (т. е. одной из строк таблицы). Отношение БД должно иметь по столбцу однородную информацию (числовую, символьную и т. д.), т. е. быть одного типа, что необязательно для таблиц, для которых тип данных определен на уровне ячейки. Отношение БД в традиционных реализациях также является только структурой данных и не предполагает такого типа атрибута, как формула.

Таким образом, в отношении удобнее хранить упорядоченную, однородную по структуре записи информацию, для которой объем не ограничивается размерами оперативной памяти и естественны поисковые операции типа: найти все записи, удовлетворяющие заданным условиям. Таблицы более удобны для отражения таких форматных документов, как бухгалтерская ведомость, форма статистической отчетности и т. п.

В ряде случаев, где это не будет вызывать неоднозначности, отношение БД будет также называться таблицей.

1.3.2. Структуры управления и обмена

Модель основного процессора KMan-машины представляется совокупностью исполняемых команд, иерархия которых была рассмотрена выше. Отметим наличие специальной структуры, сохраняемой в виде файла со стандартным расширением .lmd, который предназначен для хранения специального словаря. Последний позволяет создавать словарь терминов (под термином будем подразумевать команду или часть команды языка KMan). Хранимые в словаре термины далее можно использовать как макроподстановки для команд, используя имя соответствующего термина.

Для KMan-процессора можно ввести понятие состояния, которое определяется совокупным текущим значением памяти, включая открытые таблицы и отношения, а также набором так называемых переменных среды. Имена этих переменных имеют префикс E., после чего следует мнемоническое обозначение, например E. DECI — количество цифр дробной части числовых данных (т. е. регулируемая

точность арифметических операций процессора); E.ECOL — столбец, в котором начинается вывод сообщения об ошибках.

Всего KMan-процессор использует 60 переменных среды, приведенных ниже.

Для удобной и эффективной организации обмена данными KMan-машина использует некоторые дополнительные структуры, хранимые в виде подготовленных пользователем и ориентированных на конкретные приложения файлов (со стандартными расширениями .iff и .tmp). Файл .iff определяет так называемую экранную форму, при определении которой пользователь может специфицировать размещение подсказок и элементов формы (поля записей отношений, ячейки таблиц) на экране. Для каждого элемента указывается способ его заполнения, т. е. будет ли значение этого элемента заполняться пользователем или будет выводиться на экран средствами KMan.

Возможны автоматическое редактирование и посимвольная проверка допустимости для любого экранного элемента. При генерировании форм предусматривается выбор цветности, реверса, мигания, различной степени яркости и др.

Стандартные формы могут использоваться командами управления данными в процессе создания записей или просмотра отношений СУБД. С помощью сгенерированных форм упрощается построчный экранный ввод — вывод данных или вывод их на печать.

Для вывода законченных документов в виде отчетов, содержащих заголовки (шапки), поколонное расположение информации с равномерным размещением внутри колонок, вычислением итоговых строк и т. д., используется форматный файл (генератора отчетов) со стандартным расширением .tmp.

В интегрированном пакете KMan предусмотрена разветвленная система защиты по доступу к данным на различных иерархических уровнях пакета начиная от паролированного входа в систему KMan до защиты отдельных полей, записей, ячеек по чтению и/или записи. Предусмотрена также регистрация пользователей.

1.4. Выражения

Наиболее распространенным понятием в системах обработки данных, отражающим вычисление значений, является выражение. Выражение в языке KMan используется во многих командах и представляет собой конструкцию, состоящую из последовательности констант, переменных памяти, ячеек таблиц, полей отношений БД и функций, связанных знаками операций. Выражение вычисляет значение определенного типа (символьное, числовое, логическое). Компоненты выражения должны иметь одинаковый тип значений. Ниже рассматриваются основные элементы выражений, за исключением функций, которые строго описываются в гл. 4.

Вычисления в выражении могут использовать круглые скобки с общематематическим смыслом изменения порядка вычисления для операций различного приоритета.

1.4.1. Константы

Символьные (строковые) константы содержат от 1 до 254 символов, заключенных в двойные кавычки. Если символ двойной кавычки необходимо использовать внутри строковой константы, ему должна предшествовать обратная косая (\). Например,

"*** ОШИБКА: ДАННЫЕ НЕКОРРЕКТНЫ ***"
"ЭТО ОБРАТНАЯ КОСАЯ:\\"

Числовые константы могут быть целыми и дробными, положительными и отрицательными. Они могут содержать до 40 цифровых позиций. Десятичная точка считается как позиция. Например, +.678 —123456789

Логические константы обозначаются: TRUE (ИСТИНА) и FALSE (ЛОЖЬ).

1.4.2. Переменные

Понятие переменной в пакете KMan включает рассмотренные в предыдущем разделе основные элементы представления данных: поля доступных отношений (таблиц) БД, ячейки открытых электронных таблиц, скалярные переменные и массивы переменных, а также группу так называемых предопределенных переменных, среди которых рассмотренные выше переменные среды и служебные переменные.

Переменные определяются своими именами, конструируемыми по определенным правилам, и текущим значением заданного типа.

Полное имя переменной — поля записи отношения БД — имеет следующий вид:

[⟨имя файла отношения⟩.] ⟨имя поля⟩

Имена элементов конструируются из букв и цифр, а также символа _. Если файл-отношение известно, то достаточно имени поля, например RATE1, RATE2, STAFF.NAME.

Имя переменной — ячейки электронной таблицы — имеет следующий вид:

#⟨столбец⟩⟨строка⟩ | # (⟨номер строки⟩, ⟨номер столбца⟩)

где столбец обозначается буквами от A до Z и затем AA, AB, AC, . . . , BA, . . . , а строка — числовым номером.

В определенных случаях фрагмент таблицы может рассматриваться как двухмерный массив с именем # (второй вариант). Тогда на переменную #A2 можно ссылаться как на # (2,1). Обращение к переменным ячеек в терминах массива часто удобно при использовании ячеек в процедуре (например, в итерационном цикле, при использовании переменных в качестве индексов массивов).

Имена скалярных переменных конструируются из букв и цифр и начинаются с буквы. Имена переменных — элементов массива (при условии его предварительного определения) — имеют следующий вид:

⟨имя массива⟩ (⟨индексы⟩)

Допускаются одно- и двухмерные массивы. Например, пусть определены следующие одномерный и двухмерный массивы:

DIM MAS (36)
DIM ARR (5,5)

Тогда первый элемент массива MAS будет обозначаться как MAS (1), второй — MAS (2) и т. д. ARR (1,3) будет элементом массива ARR, расположенным в первой строке и третьем столбце. Элементы массива могут иметь разные типы значений. Действующие (активные) в данный момент переменные в общем случае не должны иметь одинаковых имен.

Таблица 1.1

Имя переменной	Назначение	Тип	Значение по умолчанию
E. AUTO	Автоматический переход к следующему меню или ячейке ЭТ	L	FALSE
E. BACG	Цвет фона для вывода электронных бланков, текстов, графиков	S	U
E. BELL	Звонок при неправильном вводе данных	L	TRUE
E. CCCS	Символ закрытия класса символов	S	]
E. CCNS	Символ отрицания класса символов	S	^
E. CCOS	Символ открытия класса символов	S	[
E. CF	Преобразование формата в ASCII (0), DIF (1), BASIC-подобный (2), бескавычный ASCII (3)	N	0
E. COMP	Вычисление значений ячеек электронных бланков по строкам, а не по колонкам	L	TRUE
E. DAVE	Вывод статистики о средних значениях для команд SELECT, STAT	L	TRUE
E. DCNT	Вывод статистики о счетчиках	L	TRUE
E. DECI	Количество цифр справа от десятичной точки	N	5
E. DELI	Символ разделителя для присоединения символьной строки (допустимо @ & : \ + — * / # ( ) . ; %) (= ")	S	⟨нулевое значение⟩
E. DMAX	Вывод статистики о максимальных значениях	L	TRUE
E. DMIN	Вывод статистики о минимальных значениях	L	TRUE
E. DSDV	Вывод статистики о стандартных отклонениях	L	TRUE
E. DTYP	Формат задания даты для функций обработки данных TOJUL, TODATE (M — месяц/день/год, D — день/месяц/год, Y — год/месяц/день)	S	M
E. DSUM	Вывод статистики о сумме	L	TRUE
E. DVAR	Вывод статистики об отклонениях	L	TRUE
E. ECHO	Эхо выводится на печать, если E. OPRN = TRUE, и в файл #DSKOUT, если E. ODSK = TRUE	L	FALSE
E. ECOL	Столбец, в котором начинается вывод сообщения об ошибке во время работы с экранной формой или электронным бланком	N	
E. EROW	Строка, в которой начинается вывод сообщения об ошибке при работе с экранной формой или электронным бланком	N	⟨номер последней строки⟩
E. FORG	Цвет шрифта при выводе электронных таблиц, текстов и графиков	S	W

Имя переменной	Назначение	Тип	Значение по умолчанию
E. GUID	Выход в меню-управляемый интерфейс сразу после загрузки KMan	L	TRUE
E. HELP	Автоматическая контекстная помощь при допущении ошибки в команде	L	FALSE
E. ICAS	Игнорирование различия между прописными и строчными буквами	L	TRUE
E. ICOM	Немедленный перерасчет таблицы при изменении или переоценка форм в случае ввода данных для GET	L	FALSE
E. IMAC	Игнорирование всех макросов	L	FALSE
E. IMRK	Игнорирование существующих маркированных записей в СУБД (кроме команд UNMARK, COMPRESS, INDEX, SORT)	L	FALSE
E. INUP	Автоматическая корректировка индексов при изменении БД	L	TRUE
E. LEGN	Использование значения #LEGEND в качестве заголовка колонок для вывода SELECT, STAT	L	FALSE
E. LLOG	Максимальная длина логического шаблона (по умолчанию)	N	5
E. LMOD	Перенос данных из последней записи в новую	L	TRUE
E. LNUM	Максимальная длина числового шаблона (по умолчанию)	N	14
E. LSTR	То же строкового шаблона	N	15
E. M1	Шаблон замены, соответствующий одному произвольному символу	S	$
E. MS	Шаблон замены, соответствующий набору произвольных символов	S	*
E. OCON	Вывод осуществляется на консоль	L	TRUE
E. ODSK	Вывод осуществляется на диск	L	FALSE
E. OPRN	Вывод осуществляется на принтер	L	FALSE
E. PAUS	Пауза после заполнения каждого экрана (или листа, если вывод идет на принтер) до нажатия любой клавиши	L	FALSE
E. PCCS	Символьная строка, отключающая печать при нормальном окончании работы с KMan	S	0, 0. 0, 0
E. PDEP	Длина печатного листа	N	60
E. PEJE	Символ перевода листа	S	\L (^L)
E. PMAR	Левая граница печати	N	0
E. PNT	Символ десятичной точки	S	.
E. POCS	Символьная строка, посылаемая на печать при первой попытке печати	S	0, 0, 0, 0
E. PROM	Символ подсказки ввода	S	—
E. PWID	Ширина печатного листа	N	120
E. SCMD	Символ, заменяющий \ в командах процессора электронных бланков	S	,
E. SECB	Подавление перевода страницы при встрече кода перевода страницы	L	FALSE

Продолжение табл. 1.1

Имя переменной	Назначение	Тип	Значение по умолчанию
E. SERR	Подавление вывода сообщения об ошибках	L	FALSE
E. SKIP	Пропуск строки после каждой строки данных, найденной SELECT	L	FALSE
E. SPAC	Количество пробелов между колонками при печати таблицы	N	2
E. SPGN	Подавление автоматической нумерации страниц при выводе SELECT и STAT	L	FALSE
E. STAT	Автоматическое вычисление статистики по команде SELECT	L	TRUE
E. STEP	Пошаговое эхо команд при выполнении процедуры	L	FALSE
E. SUPD	Подавление вывода на печать найденных записей	L	FALSE
E. SUPH	Подавление заголовков колонок при STAT и SELECT	L	FALSE
E. SWIN	Синхронизация перемещения ячеек внутри окон электронного бланка	L	FALSE

Переменные среды исполнения имеют фиксированные имена, приведенные в следующей таблице. Значение каждой из этих переменных можно изменить с помощью команды LET.

Для получения списка всех переменных определения среды и их текущих значений можно использовать команду SHOW ENVIRONMENT.

Имена служебных переменных начинаются с символа #.

Описание служебных переменных приводится в следующей таблице.

Таблица 1.2

Имя служебной переменной	Тип	Назначение
# AVER	Массив N	Одномерный массив, содержащий статистику о средних значениях, полученный по последней из команд — SELECT или STAT
# BUSY	S	Сообщение о занятости, которое будет выведено во время обработки долговыполнимых команд (по умолчанию — "ЖДИТЕ!")
# CNT	N	Статистический счетчик (количество обработанных записей), вычисленный по последней команде — SELECT, STAT, CHANCE, MARK или UNMARK
# COL	N	Количество столбцов в результирующем массиве, заполненном командой CONVERT
# DATE	S	Дата, заданная при вызове KMan (с помощью аргумента —d). Если не указана при вызове, устанавливается по системным часам

Продолжение табл. 1.2

Имя служебной переменной	Тип	Назначение
#DEFAULT	S	Имя текущей таблицы БД, используемой по умолчанию. Пустая строка, если нет такой таблицы
# DSKOUT	S	Полностью квалифицированное имя дискового файла, в который будет осуществляться вывод, если E. ODSK = TRUE. В начале сеанса используется имя "DSKOUT. TXT"
# ERRNO	N	Номер последнего из выданных KMan диагностических сообщений (начальное значение — нуль)
# FALSE	Массив S	Одномерный массив, элементам которого присваиваются синонимы ключевого слова FALSE. Для использования массива необходимо задать его размерность и установить значения элементов
# FOUND	L	TRUE, если команда поиска или функция LOOKUP нашли точное соответствие, иначе — FALSE
# LEGEND	Массив S	Одномерный массив, элементы которого станут заголовками для выходной таблицы команд STAT или SELECT, если E. SUPH = = FALSE и E. LEGN = TRUE
# MAX	Массив N/S	Одномерный массив, содержащий статистику о максимальных значениях, полученный по последней из команд — SELECT или STAT
# MIN	Массив N/S	Одномерный массив, содержащий статистику о минимальных значениях, полученный по последней из команд — SELECT или STAT
# PREFIX	S	Однобуквенный префикс, указывающий дисковое устройство, используемое KMan для временных рабочих файлов при сортировке и индексации
# ROW	N	Количество строк в результирующем массиве, заполненном командой CONVERT
# STDV	Массив N	Одномерный массив, содержащий статистику о стандартных отклонениях, полученный по последней из команд — SELECT или STAT
# SUM	Массив N	Одномерный массив, содержащий статистику о суммарных значениях, полученный по последней из команд — SELECT или STAT
# TITLE	S	Заголовок для листов выходных таблиц, сформированных командами SELECT или STAT (используется при E. OPRN = TRUE)

Продолжение табл. 1.2

Имя служебной переменной	Тип	Назначение
# TRUE	Массив S	Одномерный массив, элементам которого присваиваются синонимы ключевого слова TRUE. Для использования массива необходимо задать его размерность и установить значения элементов
# USER	S	Имя текущего пользователя КМап
# VAR	Массив N	Одномерный массив, содержащий статистику о разбросе (вариации), полученный по последней из команд — SELECT или STAT

Значения некоторых служебных переменных могут быть изменены с помощью команды LET. Например,

```
LET #TITLE = "Это заголовок"
```

определит заголовок, который будет помещаться при выводе таблиц БД, пока переменная #TITLE не получит другого значения.

1.4.3. Числовое выражение

Числовое выражение состоит из одной или более числовых констант, переменных и арифметических функций. Допустимыми знаками операций являются: + (сложение), — (вычитание), * (умножение), / (деление), ** (возведение в степень), MOD (остаток от деления). Перечень функций приводится в гл. 4. При вычислении числовых выражений действует обычный приоритет арифметических операций. Примеры:

```
A1 + 0.003
129
NUMBER — 206 * (143 — (67.3 * —2.8) ) + 178 MOD 3
10 * EXP (X + Y) + ARR (1,1)
```

В последнем примере используются стандартная функция EXP ( ) (экспонента) и элемент массива ARR (допустимый при его предварительном определении командой DIM).

1.4.4. Символьное выражение

Символьное выражение состоит из одной или более символьных констант и (или) переменных. Допустимой операцией является конкатенация строк, обозначаемая знаком «+». Соответствующие функции рассмотрены в гл. 4.

Примеры.

```
"Это — строка"
HEADER1 + HEADER 2
SUBSTR ("ABCDEFGH", I, 1)
```

В последнем примере используется функция выделения одного символа из строки "ABCDEFGH" в позиции, задаваемой числовой переменной I.

1.4.5. Логическое выражение

Логическое выражение состоит из логических констант, логических переменных, выражений отношения и логических функций, связанных знаками логических операций NOT (НЕ), AND (И), OR (ИЛИ), XOR (исключающее ИЛИ) и IN (вхождение). Символ «&» является синонимом AND.

Выражение отношения представляет пару однотипных выражений, связанных операциями: =, < >, >, >= , <, <=, для которых могут использоваться также буквенные эквиваленты: EQ, NE, GT, GE, LT, LE. Например,

```
ТАБНОМЕР = 1021
NAME1 + " " + NAME2, EQ "СЕРГЕЙ БОРИСОВ"
ЗАРПЛАТА > 180.00
```

При вычислении выражений отношения над строчными данными хвостовые пробелы в строках игнорируются.

Справа от оператора IN может быть задана группа величин. Значением операции будет ИСТИНА (TRUE), если величина, указанная слева от оператора IN, совпадает с одной из величин, специфицированных справа. Группа величин может содержать строковые, числовые и логические константы и должна быть заключена в квадратные скобки, например

```
NAME1 IN [ "ИВАНОВ", "ПЕТРОВ", "СИДОРОВ" ]
FLAG IN [ TRUE ]
CODE IN [ 2, 5, 14, I + J ]
```

При задании строковых констант в качестве значений правой части оператора IN могут использоваться спецсимволы, выполняющие функции, описанные в следующей таблице.

Таблица 1.3

Спецсимвол	Назначение	Пример	Соответствующие строки
*	Соответствует последовательности из нуля, одного или более любых символов	СЛОВО IN ["АВТОМАТ*"]	АВТОМАТ АВТОМАТИКА АВТОМАТИЗАЦИЯ
$	Соответствует одному произвольному символу	ТЕРМИН IN ["АВТОМАТ$"]	АВТОМАТА АВТОМАТУ АВТОМАТЕ
[...]	В данном месте строковой константы допустим один из заключенных в скобки символов	СЛОВО IN ["М[АО]Й"]	МАЙ МОЙ
[^...]	В данном месте строковой константы допустим любой символ, кроме заключенных в скобки	СЛОВО IN "КРА[^Н]"	КРАЙ КРАХ

В заключение приведем таблицу старшинства операций при вычислении выражений.

Таблица 1.4

Операция	Приоритет
( )	Наивысший
**	
Унарный —, NOT	
*, /, MOD	
+, —	
IN	
=, <>, <, >, <=, >=	
AND	
OR, XOR	Наинизший

Глава 2

МЕНЮ-ОРИЕНТИРОВАННЫЙ РЕЖИМ

Основным исходным режимом работы большинства пользователей пакета является меню-ориентированный режим, который представлен некоторой иерархией меню. Каждое меню представляет собой набор действий или режимов, доступных пользователю для выбора на данном шаге его работы. Меню выглядит как небольшое окно, обрамленное рамкой, в котором построчно записаны команды меню (т. е. действия или режимы работы). Одна из строчек отмечается определенным способом, например негативной подсветкой. Выделенная подсветкой строка меню называется курсором меню. Выбор требуемой команды производится нажатием клавиш управления курсором (вертикальные стрелки), которое сопровождается видимой реакцией на экране — перемещением подсветки. Выполнение команды производится нажатием клавиши ввода (в дальнейшем может обозначаться ENTER).

В процессе работы пользователь может также вводить дополнительную информацию в различных полях экрана в виде строк символов. В этом случае курсор представляет светящуюся отметку позиции ввода символа.

Иерархия команд отражается в иерархии меню. Таким образом, если команда требует дополнительной спецификации ряда действий, последние образуют меню следующего уровня. Визуально на экране это сопровождается наложением окна следующего меню на предыдущее с таким сдвигом, чтобы можно было представить весь путь выбора по дереву иерархии команд.

Таким образом, выбор действия приводит к переходу на следующий уровень иерархии меню — подменю (для конкретизации выбранного режима или действия либо задания его параметров) либо к выполнению этого действия в случае, если вся информация для его выполнения уже определена.

2.1. Вход в режим

Пакет KMan обеспечивает работу некоторыми именованными квантами времени — сеансами, запоминая состояние KMan-машины («операционную обстановку») при окончании сеанса и обеспечивая возможность через некоторое время вернуться к продолжению работы в этом сеансе.

Сеанс определяется как период непрерывной работы пакета, начинающийся со старта программы KMan и заканчивающийся выходом из нее. На системном диске при этом сохраняется текущее состояние сеанса до выхода, так что можно возобновлять его через некоторое время.

Для начала сеанса пакет вызывается из операционной системы вводом имени

d : > KMAN

Если не выдается подсказка для задания имени, пользователь попадает в сеанс с меню-управляемым режимом.

Если предыдущий сеанс откладывался, пакет запрашивает необходимость возобновления одного из отложенных сеансов с возможностью ответа (д/н).

Если пользователь хочет возобновить один из отложенных сеансов (д), пакет выдает на экран список файлов предыдущих отложенных сеансов. Для выбора возобновляемого сеанса необходимо подвести курсор к выбираемому имени из списка и нажать клавишу ENTER.

Если необходимо начать работу с нового сеанса (н), нужно подтвердить ввод нажатием клавиши ENTER и в ответ на подсказку ввести имя нового сеанса. Если вводимое имя уже существует в списке файлов сеансов, на экране появится сообщение с предложением удалить предыдущий одноименный файл сеанса. При подтверждении (д) новый файл фиксируется вместо предыдущего одноименного файла сеанса.

2.2. Структура экрана в меню-управляемом режиме

Экран меню KMan состоит из пяти специальных областей: области меню, области сообщений, служебной области, строки ввода и области команд.

Область меню находится в левой верхней части экрана. Каждое меню заключено в рамку. При выборе режима или действия на текущем уровне иерархии меню новое меню наложится сверху на предыдущее. Выбранный режим или действие из предыдущего меню остаются видимыми, располагаясь над верхней строкой рамы нового меню.

Всякий раз при появлении нового меню выделяется ярким цветом его верхняя строка и курсор располагается в начале этой строки. Выделенная строка называется текущей. Перемещая курсор по строкам, можно сделать текущей любую строку меню.

Для текущей строки можно получить страницу помощи, которая описывает действие или режим, зафиксированный в этой строке, или определить свой выбор нажатием клавиши ENTER.

Область сообщений состоит из двух строк под областью меню, которые отведены для подсказок или вывода последних результатов вычислений.

Строка ввода предназначена для ввода необходимых параметров по запросу пакета.

Область команд составляет последние три строки в конце экрана, зарезервированные для визуализации последовательности команд, сформированной пакетом как программа на языке KMan, описывающая определенное пользователем действие. Такое представление служит для обучения начинающего пользователя командному языку KMan в процессе его работы в меню-ориентированном режиме.

Служебная область находится в правой части экрана. Используется для различных целей. Может содержать информацию об управляющих клавишах, используемых в текущем режиме, либо список объектов или параметров для выбора. В служебной области могут также появляться страницы помощи.

2.3. Выбор режимов меню

Выбор требуемого режима или действия производится с помощью клавиш «стрелка вверх» и «стрелка вниз» с последующим нажатием клавиши ENTER. Если выбранное действие не является конечным (выполняемым), на экране появится меню следующего уровня иерархии — подменю для уточнения выбранного на предыдущем уровне решения.

На любом уровне меню имеется возможность возврата в предыдущее меню путем нажатия клавиши ESC либо выбора соответствующего действия в текущем меню (в каждом меню имеется действие возврата на предыдущий уровень). Для прямого возврата из любого уровня иерархии в меню верхнего уровня — главное меню — нужно одновременно нажать клавиши CTRL и C (обозначается ^C).

2.4. Получение помощи

В любой момент на экране может быть получена помощь — вспомогательная информация, описывающая текущий режим меню и возможные действия пользователя в этом режиме. Для получения помощи необходимо выбрать соответствующий режим текущего меню и нажать клавиши ^L. После завершения чтения страницы помощи для возврата в меню-управляемый режим достаточно нажать клавишу ENTER. Подобный вид пояснений называют контекстно-чувствительной помощью.

2.5. Управление курсором и исправление ошибок

Имеется два метода исправления данных, напечатанных во вводной строке: перемещение курсора с помощью управляющих клавиш к месту ошибки и ее исправление; удаление всей строки с помощью клавиш ^Z и ввод данных заново.

Всякий раз, когда в процессе формирования действия программе необходимо выяснить у пользователя некоторый параметр, курсор на экране устанавливается в начале строки ввода, а в области подсказки в верхней правой части экрана появляется перечень текущих возможностей по редактированию вводимой информации.

В таблице представлены основные клавиши управления и редактирования в меню-управляемом режиме.

Таблица 2.1. Клавиши функций управления

Клавиша	Описание
ESC	Возврат к предыдущему меню системы
^R	Уничтожение символа под курсором
^T	Удаление текущей строки
^X (стрелка вниз)	Перемещение курсора на строчку вниз
^B	Перемещение курсора на конец строки
ENTER	Фиксация окончания ввода данных
^C	Возврат к главному меню
^L	Вызов помощи
^W	Переключение режимов вставки — замены символа над курсором
^S	Перемещение курсора на одну колонку влево
^F (стрелка вправо)	Перемещение курсора на следующее слово
^A (стрелка влево)	Перемещение курсора к предыдущему слову
^Z	Очистка всей строки
^D	Перемещение курсора на один символ вправо
^V	Перемещение курсора на начало текущей строки
^I	Перемещение курсора вправо к следующей табличной позиции
^E (стрелка вверх)	Перемещение курсора на строку вверх
^H	Возврат курсора с разрушением текста

2.5. Режим выполнения

После того как с помощью навигации по иерархии меню действие для пакета KMan полностью определено и заданы необходимые параметры для его выполнения, следует выполнить это действие. При этом у пользователя есть несколько возможностей, доступ к которым осуществляется посредством выбора действия ВЫПОЛНИТЬ, имеющегося в составе тех меню, из которых можно переходить к выполнению действия. Ниже приводится описание различных вариантов действия ВЫПОЛНИТЬ.

Установить среду исполнения. Это действие позволяет выбрать определенные установки среды исполнения (см. гл. 1), которые оказывают влияние на появление на экране или печать результатов, сформированных KMan. Меняя значения этих установок, можно получать результаты в требуемом формате.

Выполнить команду. Это действие меню позволяет выполнить определенную во время процесса выбора меню задачу (команда для которой показана в области команд).

Вывод на печать. Это действие меню позволяет не только выполнить определенную в процессе выбора меню задачу, но также

выдать информацию на принтер. Чтобы выполнить это действие, принтер должен быть включен.

Вывод в файл. Можно не только выполнить определенную задачу, но и переслать полученные результаты в выбранный пользователем файл. Для этого нужно задать имя файла, в который будет пересылаться информация.

Изменить команду. После выполнения сформированного действия его можно отредактировать для последующего выполнения похожего, но другого действия. Последовательность команд, интерпретирующих выполняемое действие, появляется в строке ввода, после чего в ней можно сделать любые изменения.

Примечание. Этот режим рекомендуется только для опытных пользователей, знакомых с синтаксисом команд KMan.

Сохранить команду. Это действие меню позволяет сохранить команду, которая была образована во время выбора меню, назначив ей имя (слово, выбранное пользователем). В дальнейшем для выполнения этой команды достаточно только задать соответствующее ей имя.

2.6. Выход из меню-управляемого режима

После завершения обработки, которую нужно было выполнить в меню-ориентированном режиме, нужно нажать клавиши ^C для возврата в главное меню. Выбрав режим ВЫХОД в этом меню, можно выйти в операционную систему или в командный режим. При этом имеется набор различных вариантов выхода из режима, описанных далее.

Сохранить сеанс и выйти в ОС. Этот вариант выхода автоматически сохраняет текущий сеанс в файле. (Имя файла состоит из имени сеанса, определенного пользователем при старте KMan. Файл будет иметь расширение .isf). После этого осуществляется автоматический выход из KMan в операционную систему.

Выйти в ОС. Этот вариант осуществляет выход в ОС без сохранения текущего сеанса. Данный сеанс не может быть возобновлен в будущем.

Выйти в командный режим. Если выбран выход в командный интерфейс, будет выдано сообщение

Хотите вначале сохранить этот сеанс? . .

При подтверждении предложения (д) текущее состояние сеанса автоматически сохранится в файле (с расширением .isf). В противном случае (н) состояние сеанса не сохраняется. Независимо от варианта ответа появляется возможность вводить очередную команду для продолжения работы в KMan. Для окончания работы в командном режиме необходимо объявить команду BYE в ответ на командную подсказку, что сопровождается выходом в меню ВЫХОД.

Предыдущее меню. Этот вариант выхода возвращает пользователя в главное меню.

2.7. Пример работы в меню-ориентированном режиме

Меню-ориентированный режим предназначен для работы начинающего пользователя в случаях, когда необходимо решить разовую частную задачу, для которой не имеет смысла составлять и отлаживать отдельную программу. В связи с ориентацией на начинающего пользователя режим снабжен достаточно подробными комментариями, «помощью», подсказками, которыми сопровождается

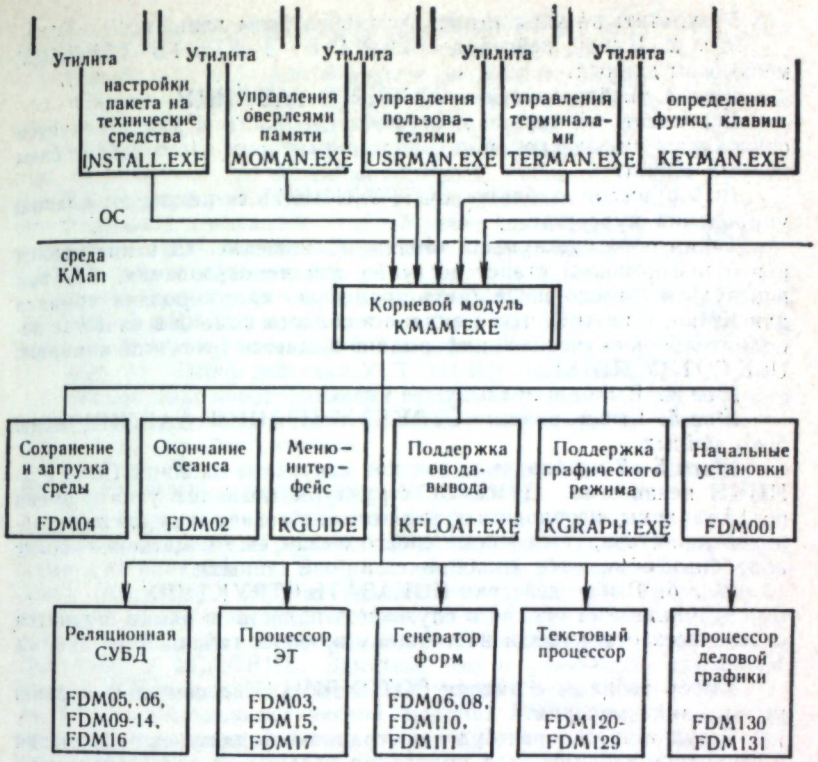


Рис. 5. Модульная структура пакета.

каждый шаг пользователя в этом режиме. Поэтому в настоящем пособии не рассматриваются все возможности обработки данных, предоставляемые меню-ориентированным режимом, и соответственно все ветви иерархии меню.

В настоящем разделе в качестве иллюстрации рассматривается процесс решения задачи запроса об информации в базе данных в меню-ориентированном режиме.

В примере используются русские обозначения структур данных и наименований режимов меню, хотя оригинальный пакет ориентирован на использование латинских букв для меню и конструирования имен. Ключевые слова для формируемых команд сохраняются в оригинальном виде.

Рассмотрим последовательность действий, необходимую для решения следующей задачи: просмотреть данные (Ф. И. О., год рождения, должность и оклад) о сотрудниках предприятия, имеющих общий стаж работы более 20 лет. Известно, что в базе данных есть таблица (отношение реляционной БД) СОТРУДНИКИ, поля которой содержат различную информацию о работниках.

Ниже приводится описание необходимой последовательности шагов, каждый из которых заключается в выборе одного режима или действия из соответствующего меню.

В примере подразумевается, что KMan уже был загружен.

Шаг 1. Выбор с помощью курсора режима БАЗА ДАННЫХ в главном меню KMan.

Результат: на экране появится меню базы данных.

Шаг 2. Выбор действия ОТКРЫТЬ / ЗАКРЫТЬ ТАБЛИЦУ меню базы данных.

Шаг 3. Выбор действия ОТКРЫТЬ ТАБЛИЦУ.

Результат: на экране в служебной области в рамке появится список всех имеющихся в активном директории диска таблиц базы данных KMan.

Выбор имени таблицы СОТРУДНИКИ с помощью клавиш управления курсором.

Таким образом, нужная таблица (отношение БД и имеющаяся в ней информация) станет доступна для использования. Так как в результате этого шага была полностью сформирована команда для KMan, в нижней части экрана в области команд в качестве дополнительной обучающей информации выдается текст этой команды: USE СОТРУДНИКИ.

Шаг 4. Выход в предыдущее меню.

Шаг 5. Выбор режима СТРУКТУРИЗАЦИЯ ТАБЛИЦ меню базы данных.

Шаги 5—7 необходимы, так как структура таблицы СОТРУДНИКИ неизвестна. Просмотр структуры позволит узнать имена полей таблицы, необходимых для решения задачи (каждое поле таблицы характеризуется кроме своего имени еще и меткой — более подробным описанием хранящихся в поле данных).

Шаг 6. Выбор действия ПОКАЗАТЬ СТРУКТУРУ.

Результат: на экране в служебной области в рамке появится список всех открытых к настоящему времени таблиц базы данных KMan.

Выбор таблицы с именем СОТРУДНИКИ с помощью клавиш управления курсором.

Результат: на экран будет постранично выдаваться информация о структуре таблицы (для получения следующей страницы нужно нажать любой символ).

Допустим, путем анализа структуры были определены имена полей, участвующих в данной задаче: поле ФИО содержит фамилию, имя и отчество сотрудника; поле ГОДРОЖД — его год рождения, ДОЛЖН — должность, ОКЛАД — размер месячного оклада, ОБЩСТАЖ — общий стаж работы. Так как в результате этого шага была полностью сформирована команда для KMan, в области команд появится ее текст: SHOW СОТРУДНИКИ.

Шаг 7. Выход в предыдущее меню.

Шаг 8. Выбор действия ПРОСМОТР ДАННЫХ меню базы данных.

Результат: на экране появится меню просмотра данных.

Шаг 9. Выбор режима МНОЖЕСТВЕННЫЙ ДОСТУП (так как, очевидно, на предприятии работает достаточно много сотрудников, чей общий стаж работы превышает 20 лет).

В результате шага 9 на экране появилось меню нового типа, внутри которого необходимо осуществить последовательный выбор всех содержащихся в нем действий. При попытке пропустить одно из входящих в меню действий KMan будет выдавать сообщение об ошибке.

Шаг 10. Выбор действия ВЫБОР ТАБЛИЦЫ. На экране в служебной области появится список всех открытых таблиц базы данных.

Выбор таблицы СОТРУДНИКИ.

Шаг 11. Выбор действия ВЫБОР ПОЛЕЙ.

Результат: на экране в служебной области появится список всех

полей таблицы, в конце которого будут помещены три дополнительные строки:

"(Все)" — для быстрого выбора всех полей таблицы,

"(Выражение)" — для возможности выбора необходимых полей путем печатания их имен,

"(ВВОД)" — для указания окончания процесса выбора необходимых полей.

Выбор курсором первого необходимого поля — ФИО.

Результат: выбранное таким образом поле будет отмечено звездочкой (в случае ошибочного выбора повторное нажатие клавиши ENTER снимет эту отметку).

Выбор (аналогично предыдущему выбору) полей ГОДРОЖД, ДОЛЖН и ОКЛАД, после чего необходимы установка курсора на последнюю строку списка (ВВОД) и нажатие клавиши ENTER.

Шаг 12. Выбор действия УСТАНОВИТЬ УСЛОВИЕ.

Это действие необходимо для установки условия выбора записей из таблицы, другими словами — выбора сотрудников, данные о которых будут просматриваться.

Шаг 13. Меню установки условия содержит два режима — НОВОЕ УСЛОВИЕ и ДОБАВИТЬ УСЛОВИЕ.

Так как условие выбора записей еще не задавалось, необходимо выбрать первый из режимов. В данной задаче выбор второго режима и не понадобится. В случае же более сложного заданного условия (например, получить данные о сотрудниках предприятия, имеющих общий стаж работы более 20 лет и зарплату менее 200 рублей) после задания нового условия потребовалось бы выбрать режим ДОБАВИТЬ УСЛОВИЕ. Заметим, что в меню-ориентированном режиме KMan каждое добавляемое условие присоединяется к предыдущим с помощью логической операции AND. Командный режим позволяет при формировании условий задавать более разнообразные операции.

Шаг 14. Выбор действия ПОСТРОИТЬ УСЛОВИЕ (действие ВВЕСТИ УСЛОВИЕ подразумевает знакомство с синтаксисом задания условий в командном языке KMan).

Результат: на экране в области сообщений появится подсказка "Укажите поле:", а в служебной области — список всех полей таблицы СОТРУДНИКИ.

Выбор поля ОБЩСТАЖ (принимающего участие в условии выбора записей таблицы: общий стаж работы превышает 20 лет).

Результат: в нижней части экрана в области команд появится начало текста формируемого условия — имя выбранного поля ОБЩСТАЖ, в области сообщений появится подсказка для следующего действия — "Укажите оператор:", а в служебной области — список возможных операторов: равно, не равно, больше, больше или равно, меньше, меньше или равно.

Выбор оператора "больше".

Результат: в нижней части экрана в области команд появится продолжение текста формируемого условия — строка "ОБЩСТАЖ)", в области сообщений появится подсказка для следующего действия — "Укажите второй операнд:", а в служебной области — список возможных типов операндов: числовая константа, строковая константа, поле, выражение.

Выбор операнда "числовая константа".

Результат: в нижней части экрана в области ввода появится подсказка "Введите выражение", а под ним — пустая строка ввода. В правом верхнем углу экрана будет выведена помощь — описание клавиш редактирования ответа.

Необходимо ввести значение числовой константы — 20 и нажать клавишу ENTER.

Шаг 15. Выход в предыдущее меню.

Шаг 16. Выход в предыдущее меню.

Шаг 17. Выбор действия ВЫПОЛНИТЬ.

Результат: на экране в области меню появится меню ВЫПОЛНИТЬ, а в области команды — полный текст сформированной (шаги 9—16) команды: BROWSE СОТРУДНИКИ FOR ОБЩСТАЖ > 20 WITH ФИО, ДОЛЖН, ОКЛАД.

Шаг 18. Выбор действия ВЫПОЛНИТЬ КОМАНДУ.

Результат: экран очистится и на нем появится список имен и значений выбранных для решения задачи полей (ФИО, ДОЛЖН и ОКЛАД) первой удовлетворяющей условию записи таблицы СОТРУДНИКИ. В верхней строке экрана будет выведена служебная информация — номер найденной записи и подсказка возможных действий, необходимых при просмотре всех удовлетворяющих заданному условию записей о сотрудниках предприятия:

^F = вперед ^A = назад ENTER = изменить ESC = уйти

Действия «вперед» и «назад» позволяют получать на экране следующую или предыдущую запись таблицы, удовлетворяющую поставленному условию. Действие "изменить" позволяет войти в режим модификации значений выбранных полей записи (т. е. режим МНОЖЕСТВЕННЫЙ ДОСТУП меню ПРОСМОТРА ДАННЫХ поддерживает не только вывод, но и изменение записей). Действие "уйти" обеспечивает возможность досрочного выхода из режима просмотра записей (иначе выход из просмотра будет произведен автоматически после вывода на экран последней из удовлетворяющих условию записей).

После просмотра всех интересующих записей и получения необходимой информации на экране вновь появляется меню ВЫПОЛНИТЬ.

Шаг 19. Нажатие ^C осуществляет быстрый выход в главное меню.

Шаг 20. Выбор режима БАЗА ДАННЫХ главного меню KMan.

Результат: на экране появится меню базы данных.

Шаг 21. Выбор действия ОТКРЫТЬ / ЗАКРЫТЬ ТАБЛИЦУ.

Шаг 22. Выбор действия ЗАКРЫТЬ ТАБЛИЦУ.

Результат: на экране в служебной области в рамке появится список всех открытых в настоящий момент таблиц базы данных KMan.

Выбор имени таблицы СОТРУДНИКИ с помощью клавиш управления курсором.

Таким образом, таблица (и имеющаяся в ней информация) перестанет быть доступной для использования. Так как в результате этого шага полностью сформирована команда для KMan, в области команд будет выдан текст этой команды: FINISH СОТРУДНИКИ.

2.8. Схемы иерархии основных меню режима

Ниже приводится иерархия меню KMan.

— Главное меню —
- Установки
- Вычисления
- База данных
- Электронные таблицы
- Тексты
- Графики
- Внешняя среда
- Выход

— Установки —
- Открыть/закрыть таблицу
- Среда исполнения
- Словарь
- Сохраняемые образы
- Освобождение ресурсов
- Предыдущее меню

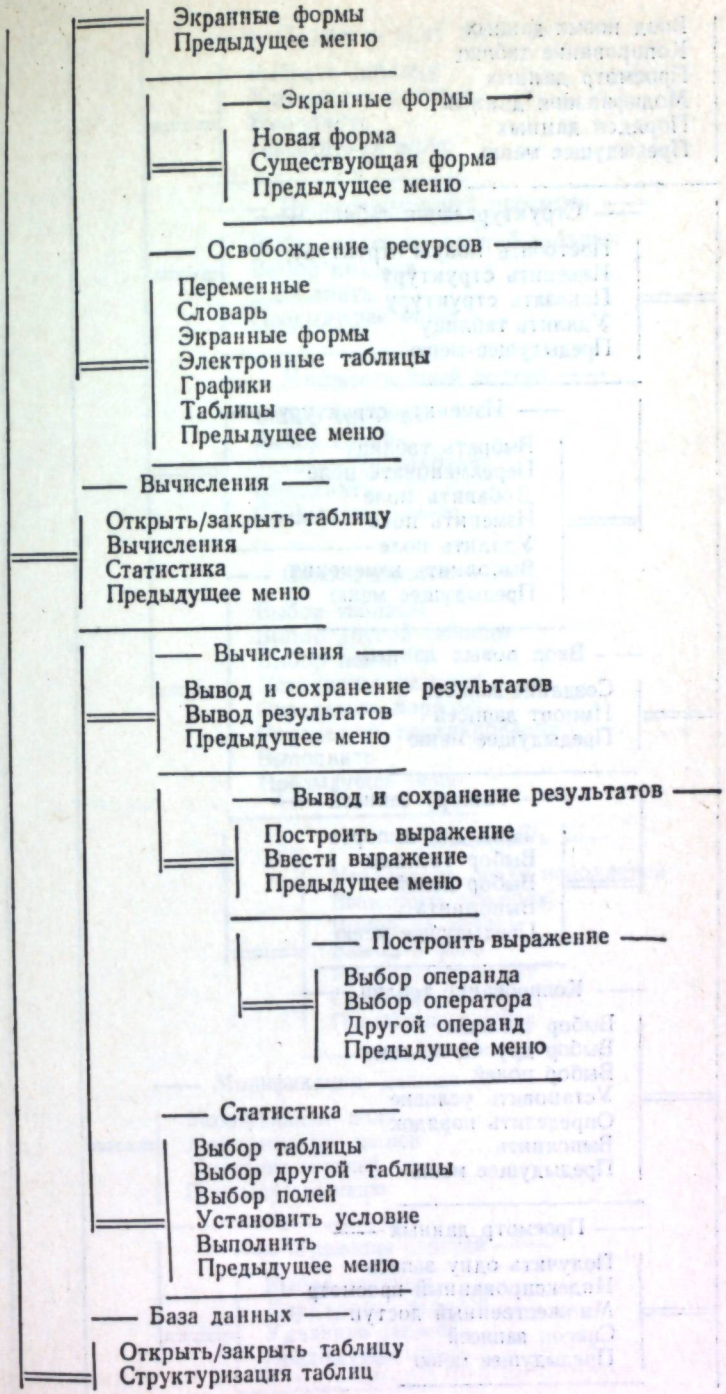
— Среда исполнения —
- Установки данных
- Установки дискового вывода
- Установки статистики
- Установки электронных таблиц
- Установки печати
- Установки цвета
- Установки листингов
- Установки графиков
- Предыдущее меню

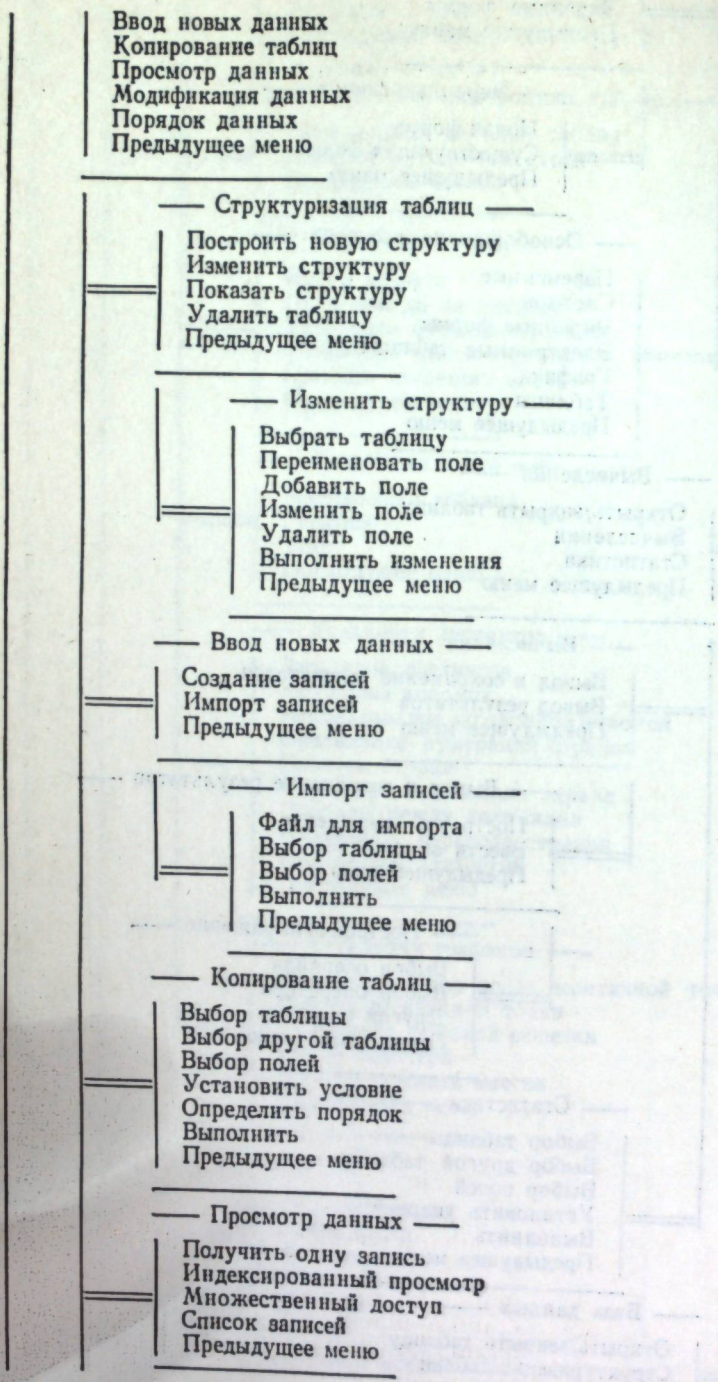
— Установки данных —
- Начало с последней записи
- Игнорировать прописные буквы
- Количество десятичных цифр
- Длина чисел
- Символ десятичной точки
- Предыдущее меню

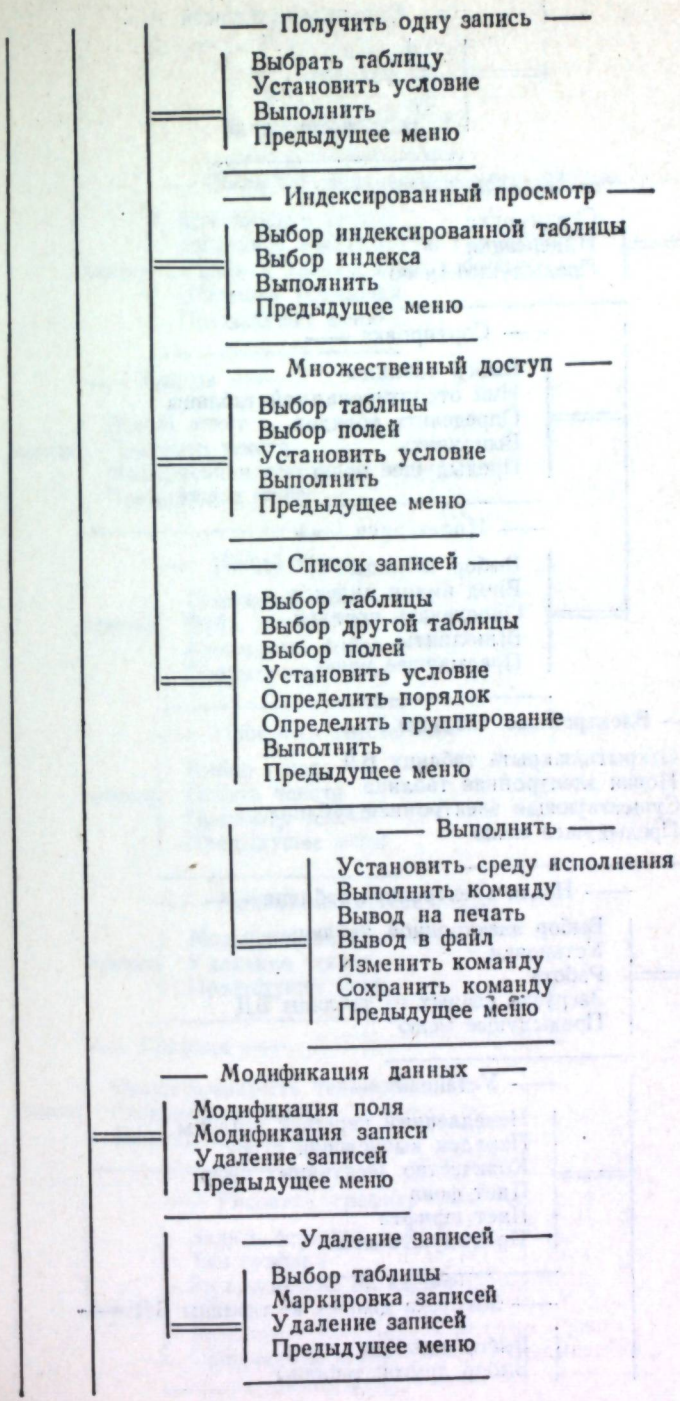
— Установки дискового вывода —
- Путь вывода для файла
- Имя выводного файла
- Предыдущее меню

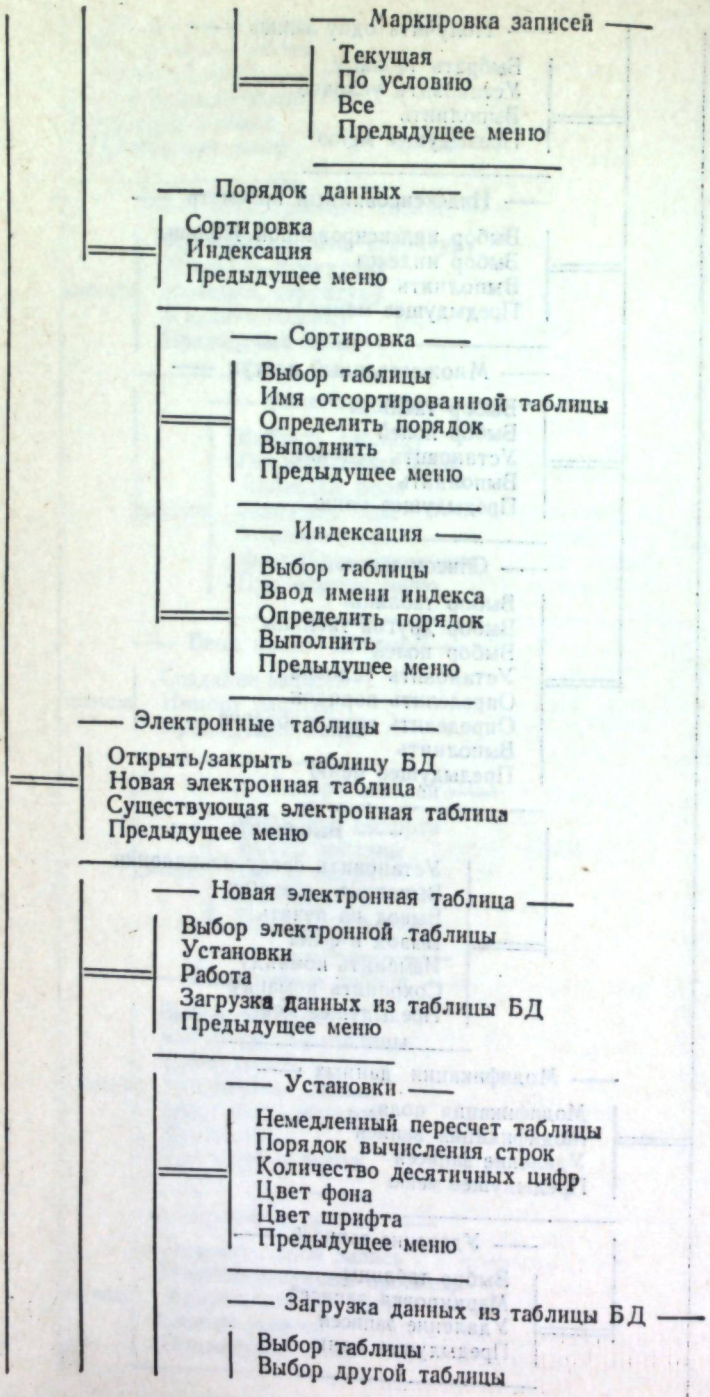
— Установки статистики —
- Среднее
- Счетчик
- Максимум
- Минимум
- Стандартное отклонение
- Сумма

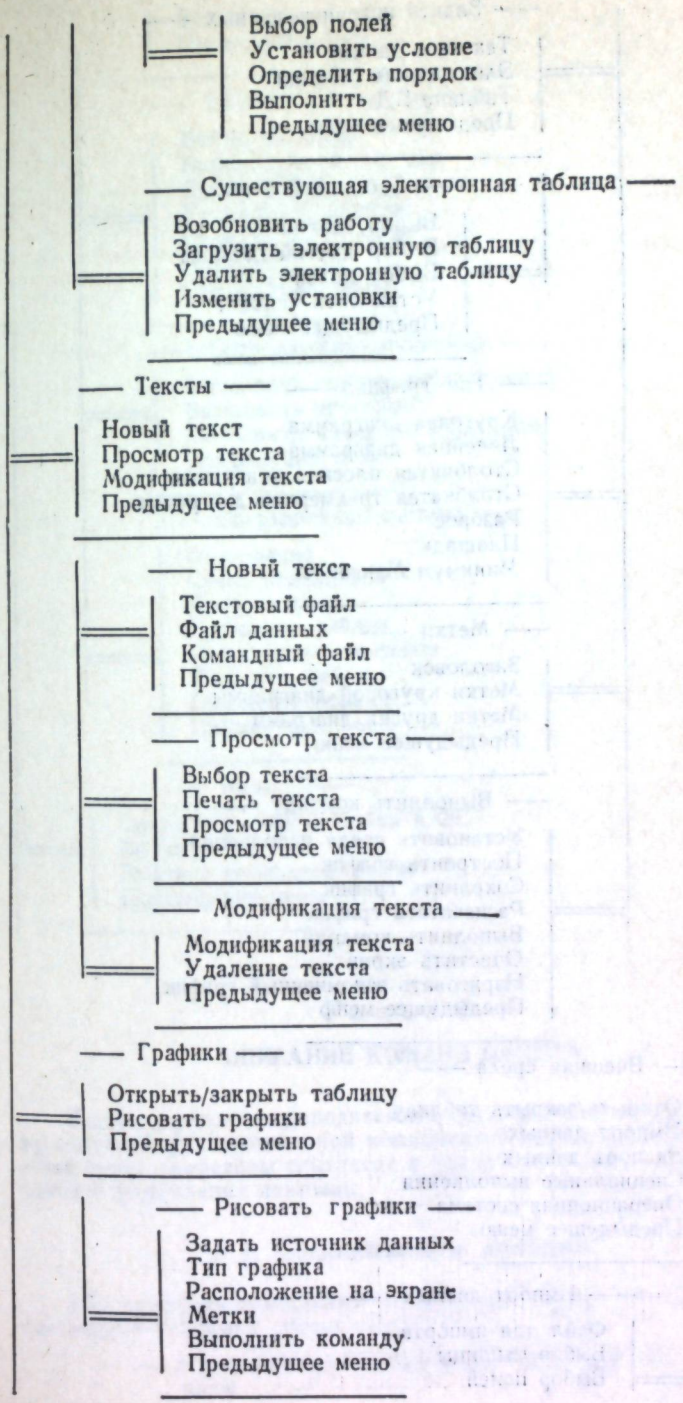

Вариация
Предыдущее меню
Установки электронных таблиц
Немедленный пересчет таблиц
Порядок вычисления строк
Предыдущее меню
Установки печати
Выбор шрифта
Путь вывода на принтер
Установка ширины страницы
Установка длины страницы
Границы печатного листа
Предыдущее меню
Установки цвета
Электронная таблица
Графики
Текст
Предыдущее меню
Установки листингов
Заголовок листингов
Заголовки колонок
Использование заголовков колонок
Подавление нумерации страниц
Пропуск строки
Пауза при заполнении экрана
Пробелы между колонками
Подавление перевода страниц
Подавление статистик
Предыдущее меню
Установки графиков
Количество цифр после десятичной точки
Символ десятичной точки
Включение фоновой решетки
Выбор плоттера
Транспонировать массив
Предыдущее меню
Словарь
Сохранить / загрузить
Ввод нового слова
Просмотр словаря
Модификация словаря
Предыдущее меню
Сохраняемые образы

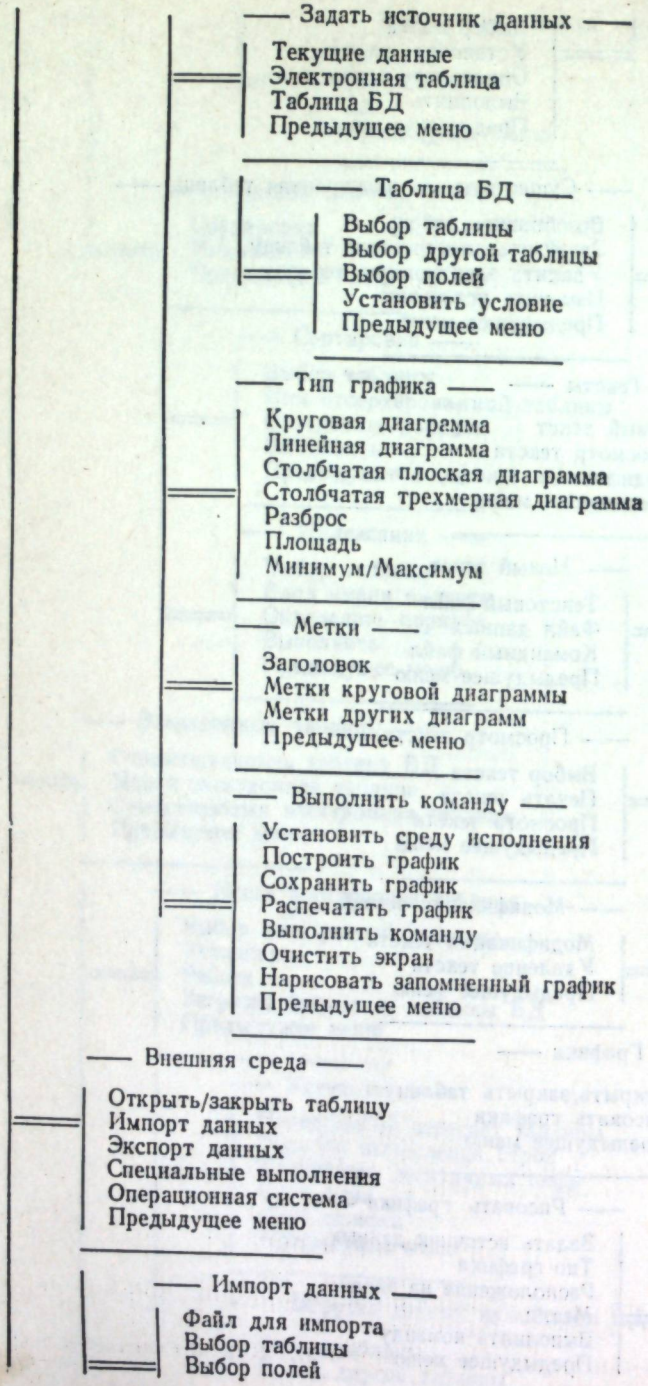

Экранные формы
Предыдущее меню
Экранные формы
Новая форма
Существующая форма
Предыдущее меню
Освобождение ресурсов
Переменные
Словарь
Экранные формы
Электронные таблицы
Графики
Таблицы
Предыдущее меню
Вычисления
Открыть/закрыть таблицу
Вычисления
Статистика
Предыдущее меню
Вычисления
Вывод и сохранение результатов
Вывод результатов
Предыдущее меню
Вывод и сохранение результатов
Построить выражение
Ввести выражение
Предыдущее меню
Построить выражение
Выбор операнда
Выбор оператора
Другой операнд
Предыдущее меню
Статистика
Выбор таблицы
Выбор другой таблицы
Выбор полей
Установить условие
Выполнить
Предыдущее меню
База данных
Открыть/закрыть таблицу
Структуризация таблиц

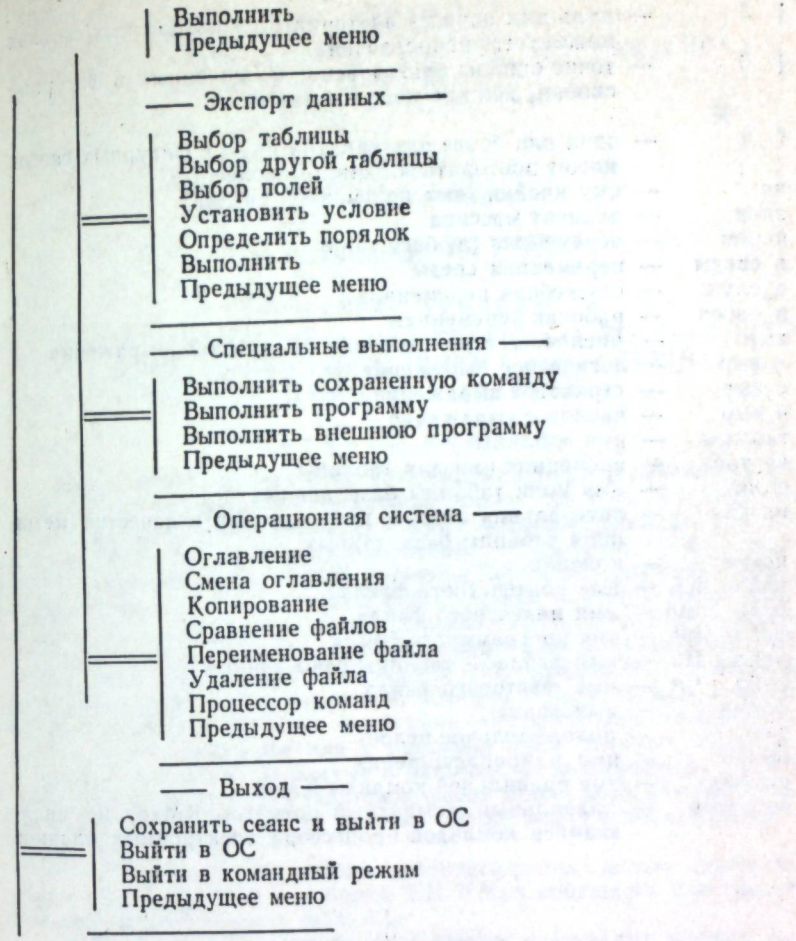

Ввод новых данных
Копирование таблиц
Просмотр данных
Модификация данных
Порядок данных
Предыдущее меню
Структуризация таблиц
Построить новую структуру
Изменить структуру
Показать структуру
Удалить таблицу
Предыдущее меню
Изменить структуру
Выбрать таблицу
Переименовать поле
Добавить поле
Изменить поле
Удалить поле
Выполнить изменения
Предыдущее меню
Ввод новых данных
Создание записей
Импорт записей
Предыдущее меню
Импорт записей
Файл для импорта
Выбор таблицы
Выбор полей
Выполнить
Предыдущее меню
Копирование таблиц
Выбор таблицы
Выбор другой таблицы
Выбор полей
Установить условие
Определить порядок
Выполнить
Предыдущее меню
Просмотр данных
Получить одну запись
Индексированный просмотр
Множественный доступ
Список записей
Предыдущее меню


Получить одну запись
Выбрать таблицу
Установить условие
Выполнить
Предыдущее меню
Индексированный просмотр
Выбор индексированной таблицы
Выбор индекса
Выполнить
Предыдущее меню
Множественный доступ
Выбор таблицы
Выбор полей
Установить условие
Выполнить
Предыдущее меню
Список записей
Выбор таблицы
Выбор другой таблицы
Выбор полей
Установить условие
Определить порядок
Определить группирование
Выполнить
Предыдущее меню
Выполнить
Установить среду исполнения
Выполнить команду
Вывод на печать
Вывод в файл
Изменить команду
Сохранить команду
Предыдущее меню
Модификация данных
Модификация поля
Модификация записи
Удаление записей
Предыдущее меню
Удаление записей
Выбор таблицы
Маркировка записей
Удаление записей
Предыдущее меню


Маркировка записей
Текущая
По условию
Все
Предыдущее меню
Порядок данных
Сортировка
Индексация
Предыдущее меню
Сортировка
Выбор таблицы
Имя отсортированной таблицы
Определить порядок
Выполнить
Предыдущее меню
Индексация
Выбор таблицы
Ввод имени индекса
Определить порядок
Выполнить
Предыдущее меню
Электронные таблицы
Открыть/закрыть таблицу БД
Новая электронная таблица
Существующая электронная таблица
Предыдущее меню
Новая электронная таблица
Выбор электронной таблицы
Установки
Работа
Загрузка данных из таблицы БД
Предыдущее меню
Установки
Немедленный пересчет таблицы
Порядок вычисления строк
Количество десятичных цифр
Цвет фона
Цвет шрифта
Предыдущее меню
Загрузка данных из таблицы БД
Выбор таблицы
Выбор другой таблицы


Выбор полей
Установить условие
Определить порядок
Выполнить
Предыдущее меню
Существующая электронная таблица
Возобновить работу
Загрузить электронную таблицу
Удалить электронную таблицу
Изменить установки
Предыдущее меню
Тексты
Новый текст
Просмотр текста
Модификация текста
Предыдущее меню
Новый текст
Текстовый файл
Файл данных
Командный файл
Предыдущее меню
Просмотр текста
Выбор текста
Печать текста
Просмотр текста
Предыдущее меню
Модификация текста
Модификация текста
Удаление текста
Предыдущее меню
Графики
Открыть/закрыть таблицу
Рисовать графики
Предыдущее меню
Рисовать графики
Задать источник данных
Тип графика
Расположение на экране
Метки
Выполнить команду
Предыдущее меню





Глава 3

ОПИСАНИЕ КОМАНД ПАКЕТА

В данном разделе приводится описание команд пакета. Поскольку основной функциональной компонентой пакета является СУБД, ниже будут приведены синтаксис и семантика команд манипулирования и управления данными.

3.1. Соглашения о нотации

При описании синтаксиса команд пакета будут использоваться следующие условные обозначения:

〈 〉 — в угловые скобки заключаются нетерминальные символы

[ ] — нуль или одна из альтернатив в квадратных скобках может быть использована

{ } — точно одна из альтернатив, заключенных в фигурные скобки, должна использоваться

$\{\ \}^{*}$ — одна или более альтернатив внутри фигурных скобок может повторяться один или более раз

яч — имя ячейки (имя переменной-ячейки)

элем — элемент массива

перем — переменная (любого типа)

п_среды — переменная среды

п_служ — служебная переменная

п_рабоч — рабочая переменная

выр — числовое, строковое или логическое выражение

л_выр — логическое выражение

с_выр — строковое выражение

ч_выр — числовое выражение

таблица — имя таблицы

вр_табл — временное имя для таблицы

поле — имя поля таблицы базы данных

метка — литеральная строка, используемая в качестве метки поля таблицы базы данных

кол — колонки

файл . icf — имя контекстного файла

файл . ind — имя индексного файла

файл . ipf — имя программного файла

файл . itb — имя файла — таблицы базы данных

файл . txt — имя текстового файла

форма — имя формы

цел — положительное целое

макро — имя макрорасширения

команда — имя правильной команды KMan

оператор — правильный командный оператор KMan, не являющийся командой процессора электронных бланков

3.2. Определения

⟨коды доступа⟩ : : = { [READ [ACCESS] "⟨коды чтения⟩"] [WRITE [ACCESS] "⟨коды записи⟩"}

⟨разрыв⟩ : : = ⟨выражение⟩

⟨условие⟩ : : = л_выр⟩

⟨коорд⟩ : : = ⟨ч_выр-1⟩, ⟨ч_выр-2⟩

⟨эффекты⟩ : : = { [B] [R] [L] [S] }

⟨режимы загрузки⟩ : := { [W] [A] [M] [F] [E] [C] [U] }

⟨режимы сохранения⟩::= { [W] [A] [M] [F] [E] [C] [U] }

⟨порядок⟩ : : = {ASCENDING | AZ | DESCENDING | ZA} {⟨выр⟩}*

⟨позиция⟩ ::= {FIRST | LAST | ⟨ч_выр-1⟩ | PRIOR [⟨ч_выр-2⟩] | NEXT [⟨ч_выр-3⟩]}

⟨пределы⟩:: = {CURRENT | ALL | NEXT [⟨ч_выр-1⟩] | PRIOR [⟨ч_выр-2⟩]| RANGE [⟨ч_выр-3⟩ [, ⟨ч_выр-4⟩]]}

⟨тип⟩ : : = {STR | NUM | INT | LOGIC}

⟨блок⟩ : : = {⟨элем-1⟩ TO ⟨элем-2⟩ | ⟨яч-1⟩ TO ⟨яч-2⟩} *

⟨коды⟩ : : = ⟨ч_выр-1⟩ [{, ч_выр-2⟩ }]

⟨кол⟩ : : = ⟨ч_выр-1⟩ [{, ⟨ч_выр-2⟩}*]

⟨положение⟩ : : = {TOP [LEFT | RIGHT] | BOTTOM [LEFT | RIGHT] | LEFT | RIGHT}

⟨сектора⟩ : : = ⟨ч_выр-1⟩ [{, ⟨ч_выр-2⟩ }*]

3.3. Список команд

3.3.1. Команда ATTACH

Синтаксис команды

ATTACH {[FROM] }" файл .txt⟩" | CELL[⟨яч-1⟩ TO ⟨яч-2⟩]}[TO {⟨таблица⟩ | вр_табл⟩}] [WITH {⟨поле⟩ | STR}]} | {⟨ч_выр⟩ [TO {⟨таблица⟩ | вр_табл⟩}]}

Описание команды

Эта команда используется для присоединения к таблице существующих в стандартном текстовом файле данных. Необходимо иметь доступ записи к таблице. Такие текстовые файлы могут быть созданы многими различными способами: с помощью BASIC программы, текстового редактора, команды CONVERT системы KMan. Предполагается, что текстовый файл, записи которого будут присоединены к таблице, содержит данные, соответствующие полям в определении записи таблицы. Числовые и логические значения в текстовом файле могут быть ограничены (разделены) символами табуляции, пробелами, запятыми, точками с запятой или концом строки. Взятые в кавычки строковые значения ограничиваются также этими символами или с помощью пары двойных кавычек. Строковые значения, не взятые в кавычки, ограничиваются только концом строки. Запись в текстовом файле может быть расположена на многих строках, или же на одной строке могут помещаться многие записи.

При использовании данных текстового файла с целью присоединения новых записей к таблице БД KMan соблюдает следующие правила, приведенные в табл. 3.1.

После заполнения полей новой записи значениями данных эта новая запись присоединяется в конец таблицы. После этого система KMan продолжает чтение следующего значения данных в текстовом файле и т. д., до тех пор пока не будет присоединена другая новая запись. Этот процесс продолжается, до тех пор пока система не обнаружит условие конца файла в текстовом файле, что приводит к завершению обработки команды ATTACH.

Последняя присоединенная к таблице запись становится текущей записью таблицы. В этом случае, если ни одна запись не была присоединена, текущая запись остается без изменений. Отметим, что можно присоединить файл к пустой таблице.

Параметр ⟨ч_выр⟩ является положительным целым числом, которое указывает, сколько записей присоединять к таблице с именем, определенным параметром ⟨таблица⟩ (выражение должно начинаться с числовой константы). Значения этих записей определяются переменной E. LMOD.

При присоединении записей текстового файла к таблице иногда оказывается полезным использовать значения данных текстового файла только для некоторых вновь присоединенных полей записи.

Таблица 3.1

Поле	Допустимые значения текстового файла	Результирующее значение поля
Числовое	Число	То же самое число
	Число,заключенное в двойные кавычки	Число
Логическое	.ИСТИНА. или ИСТИНА или «ИСТИНА»	ИСТИНА
	.ЛОЖЬ. или ЛОЖЬ или «ЛОЖЬ»	ЛОЖЬ
	Ненулевое число	ИСТИНА
	0 или «0»	ЛОЖЬ
	Ненулевая строка, которая начинается цифрой	ИСТИНА
	F, f, «F», «f», N, n, «N», «n»	ЛОЖЬ
	T, t, «T», «t», Y, y, «Y», «y»	ИСТИНА
Строковое	Символьная строка, заключенная в двойные кавычки	Та же строка
	Символьная строка, которая заканчивается кодом конца строки	Та же строка

Такую операцию можно осуществить с помощью явной спецификации тех действительных полей, значения которых должны быть получены из текстового файла. Последовательность этих полей определяет тот порядок, в котором ожидается появление значений данных в текстовом файле. Поля, не специфицированные с помощью оператора WITH, получают стандартные значения по умолчанию от KMan. С другой стороны, при присоединении записей к таблице можно проигнорировать некоторые входные величины данных. Если, например, каждая запись текстового файла содержит пять значений данных, а таблица БД содержит только три поля, то два значения входных данных могут быть проигнорированы при присоединении каждой новой записи к таблице. Это легко сделать, добавляя ключевые слова STR, NUM и LOGIC во фразу WITH всюду, где соответствующие строковые, числовые или логические значения должны быть проигнорированы.

Если таблица содержит виртуальные поля, KMan выполняет обработку команды ATTACH так, как будто эти поля не существуют. Нельзя присоединять файл к таблице БД, не имея доступа по записи ко всем полям этой таблицы.

Команда ATTACH может обрабатывать несколько символьных строк в одной и той же строке текстового файла, даже если каждая из них не заключена в двойные кавычки. Для этого величину переменной E. DELI необходимо установить равной тому символу, который используется для разделения этих символьных строк. Значение E. DELI и символ конца строки будут восприниматься KMan как ограничители символьных строк. Для возвращения к нормальному режиму работы ATTACH необходимо вернуть переменной E. DELI пустое значение (LET E.DELI = " ").

3.3.2. Команда BROWSE

Синтаксис команды

BROWSE {⟨таблица⟩ | ⟨вр_табл⟩} [{FOR | WHERE | WHILE} {⟨условие⟩}] [⟨пределы⟩] [WITH⟨форма⟩ | WITH {⟨поле⟩}]

Описание команды

Команда BROWSE используется для очистки экрана и отображения значений полей определенной записи заданной таблицы. Номер записи указывается в первой строке экрана, а значения полей начинаются с третьей строки. С помощью использования одного из пределов можно управлять выбором отображаемой записи из таблицы.

После появления значений записи на экране курсор устанавливается в верхней части экрана. Если не нужно изменять выведенные значения, следует нажать клавишу СНЯТИЕ. Нажатие клавиши ВВОД вызывает перемещение курсора в позицию первого значения. В этом случае можно отредактировать это значение, используя различные управляющие клавиши.

Чтобы просматривать записи таблицы, необходимо иметь доступ по чтению для этой таблицы. Редактировать записи во время их просмотра можно лишь в том случае, если есть доступ по записи к таблице.

Отредактированное значение поля заменяет старое значение поля в таблице при нажатии клавиши ВВОД. При нажатии клавиши ВВОД после завершения обработки последнего отображенного значения автоматически отображаются значения следующей записи. Если следующей записи в указанных пределах нет, команда BROWSE завершается.

Поскольку таблица может иметь записи очень большого размера, стандартный формат просмотра может не обеспечить одновременное появление всех данных на одном экране. В этом случае KMan автоматически форматирует данные найденной записи в несколько экранов. Если данные не помещаются в экран, существование экранов продолжения указывается сообщением: «Еще данные».

Необязательная фраза «FOR ⟨условие⟩» указывает, что необходимо выводить лишь записи, которые удовлетворяют заданным условиям. Если вместо FOR используется фраза WHILE, просмотр прекращается, как только появится запись, для которой условие не выполняется. Если с фразой WHILE не заданы пределы, просмотр начинается с текущей записи. Фраза "WITH ⟨поле⟩" указывает поля записи, которые нужно выводить на экран для просмотра и обработки. Если список полей опущен, обрабатываются все доступные поля.

Если использование экранной формы не задано, в ответ выводятся номер записи, имена всех полей и значения каждого поля первой записи в заданных пределах, которая удовлетворяет заданным условиям. Если список полей записи задан явно, появляются значения только этих полей. Затем курсор помещается в верхнюю часть экрана.

При просмотре отображаются лишь те поля, к которым имеется доступ по чтению. Кроме того, не разрешается редактирование значения любого поля, к которому нет доступа по записи (включая виртуальные поля).

После просмотра и редактирования первой отображенной записи на экран может быть выведена следующая запись, удовлетворяющая условиям, и т. д., пока не будет достигнут конец заданных пределов.

3.3.3. Команда CHANGE

Синтаксис команды

```
CHANGE ⟨поле⟩ [ IN {⟨таблица⟩ | ⟨вр_табл⟩} ]  TO ⟨выр⟩
       [ {FOR | WHERE | WHILE } {⟨условие⟩} ] [⟨пределы⟩]
```

Описание команды

Команда предназначена для изменения значений полей в записях таблиц БД.

Параметр ⟨пределы⟩ определяет запись в указанной таблице. Значение выражения становится новым значением заданного поля этой записи. Если имя таблицы опущено, меняется запись в текущей таблице.

Измененная запись становится текущей записью в таблице. Если не заданы пределы, предполагается, что изменения вносятся во все записи таблицы.

С помощью одной команды CHANGE можно изменить группу записей. Значение заданного выражения становится новым значением указанного поля каждой записи в группе. При этом заданное выражение вычисляется заново соответствующим образом для каждой записи. Если значение переменной E.STAT равно ИСТИНА, автоматически изменяется значение служебной переменной #CNT, отражая количество измененных записей.

Фраза "FOR ⟨условие⟩" указывает, что меняются лишь те записи в указанных пределах, которые удовлетворяют заданным условиям. Значение выражения становится новым значением заданного поля для каждой из этих записей. Если вместо фразы FOR задана WHILE, изменения прекращаются, как только встретится запись, для которой условие не выполняется. Если с фразой WHILE не заданы пределы, поиск начинается с текущей записи.

Нельзя изменять поле, не имея к нему доступа по записи. Указанное поле должно быть действительным (т. е. не виртуальным).

Таблица может содержать много записей, удовлетворяющих заданным условиям. KMan рассматривает лишь те, которые находятся в указанных пределах.

3.3.4. Команда COMPRESS

Синтаксис команды

```
COMPRESS { ⟨таблица⟩ | ⟨вр_табл⟩}
```

Описание команды

Команда предназначена для удаления из таблицы всех записей, отмеченных с помощью TRUE.

Таблицу можно сжимать только в случае, если имеется доступ по записи ко всем полям таблицы. Команда COMPRESS не зависит

от значения переменной E. IMPK. Имя таблицы должно быть задано явно.

Удаление помеченных записей приводит к физическому сжатию таблицы, выполняемому путем уничтожения всех TRUE записей. Оставшиеся записи автоматически перенумеровываются. Пустых мест в таблице нет. После сжатия нет текущей записи. Каждый индексный файл, существующий для таблицы, становится неактуальным.

3.3.5. Команда CONVERT

Синтаксис команды

```
CONVERT [ALL UNIQUE] [ { ⟨выр-1⟩ [USING " ⟨шаблон⟩"} ]*
        [FROM { ⟨таблица⟩ | ⟨вр_табл⟩} [{FOR | WHERE |
        WHILE} {⟨условие⟩}]]  |  [PLUCK ⟨выр-2⟩ [WITH
        " ⟨файл .ind⟩"]] TO { "⟨файл  .txt⟩" | CELL [⟨яч-1⟩
        [[TO] ⟨яч-2⟩]] |
        ARRAY ⟨элем-1⟩ [TO ⟨элем-2⟩] | ⟨таблица⟩}*  [⟨пределы⟩] [ ORDER BY {⟨порядок⟩} ]
```

Описание команды

Команда предназначена для преобразования таблиц БД в файлы специальных форматов.

В ответ на эту команду KMan начинает рассматривать все записи внутри обозначенных пределов в указанной таблице. Для каждой из этих записей KMan оценивает заданные выражения (максимум 255) и записывает результирующие значения как записи или строки в обозначенный приемник. Выражением могут быть: поле; арифметическое выражение, включающее числовые целые и дробные поля; строковые выражения, включающие строковые поля; логические выражения, включающие логические поля.

Если выражения не заданы, преобразовываются все поля. Заданная в команде CONVERT таблица становится новой таблицей по умолчанию. Если необязательная фраза "FROM ⟨имя таблицы⟩" опущена, используется текущая по умолчанию таблица.

В качестве приемника данных может быть задано:

"⟨файл. txt⟩" — данные, сгенерированные командой CONVERT, будут записаны в этот внешний файл. Значение переменной E. CF управляет форматом записи данных в файл

⟨таблица⟩ — данные, сгенерированные командой CONVERT, будут добавлены как новые записи в эту таблицу. Значение переменной E. CF должно быть равно 4. Если указанная таблица не существует, команда CONVERT автоматически определяет ее структуру в соответствии с образованными записями перед тем, как помещать туда записи

CELL ⟨блок ячеек⟩ — данные, сгенерированные командой CONVERT, становятся новым определением ячеек заданного блока. Блок ячеек задается в терминах двух ячеек, разделенных словом TO

CELL ⟨ячейка⟩ — данные, образованные командой CONVERT, становятся новыми определениями ячеек внутри части

электронного бланка, начинающейся с заданной ячейки и потенциально продолжающейся до конца электронного бланка

ARRAY ⟨блок массива⟩ — данные, образованные командой CONVERT, становятся новыми значениями элементов массива внутри обозначенного блока. Блок массива определен в терминах двух элементов массива, разделенных словом TO

ARRAY ⟨элемент⟩ — данные, сгенерированные командой CONVERT, становятся новыми значениями элементов массива внутри части массива, начинающейся с обозначенного элемента и потенциально продолжающейся до конца массива. Если данные преобразуются в массив, служебные переменные #ROW и #COL покажут, сколько строк и сколько столбцов массива получили данные по команде CONVERT

Исходная таблица БД (источник информации) этой операцией не изменяется. Если таблица-источник не определена, по умолчанию подразумевается текущая таблица. Значения данных преобразуются только для заданных выражений.

Команда CONVERT позволяет получать и преобразовывать связанные данные из различных таблиц. В этом случае пределы (если они установлены) относятся только к первой таблице. Имя каждой таблицы, поля которой упоминаются в каком-либо из выражений команды, должно быть описано с помощью фразы FROM. При задании условия для каждой таблицы могут быть использованы как значения полей из этой таблицы, так и поля любой из таблиц БД, упомянутой в предыдущих фразах FROM этой команды. При этом ссылки на поля из любой таблицы, кроме заданной первой (как при задании условий, так и пределов), должны содержать имя соответствующей таблицы.

Условия WHILE допустимы только для первой таблицы. Условия для второй (и последующих) таблицы должны включать спецификацию, какие записи в этой таблице связаны с записями предыдущих таблиц. Это достигается заданием равенства значений соответствующих друг другу полей различных таблиц. KMan сливает две таблицы на основании соответствующих значений полей, соединяя вместе записи с одинаковыми значениями. Для каждого соединения записей KMan преобразует значения данных для каждого выражения, установленного в команде CONVERT. Таблицы-источники данных при этом не изменяются.

Можно потребовать, чтобы созданные выходные строки для передачи приемнику динамически сортировались в любой комбинации направлений ASCENDING (или AZ) и DESCENDING (или ZA). Следует определить направление и последовательность из одного или более выражений, противоположное направление и последовательность выражений, снова первое направление и последовательность выражений и т. д.

Если значение переменной E. ICAC равно ИСТИНА, KMan игнорирует различия между верхним и нижним регистрами при сортировке. Если выражение — это просто поле, то его шаблон (или размер) определяет число символов, используемых в целях сортировки. Для более сложных выражений значение переменной E.LSTR определяет, сколько символов в значении выражения используется в целях сортировки. Так как сортировка — дополнительная работа для KMan, то она увеличивает время выполнения команды CONVERT.

В случае использования в команде CONVERT только одной таблицы — источника данных — выходные записи могут быть упорядочены по значению любого из полей (или выражений, включающих такие поля) этой таблицы, даже если эти поля не используются при передаче. Если указана передача данных из многих таблиц, KMan делает промежуточную таблицу, содержащую значения всех полей, существующих во всех таблицах. Любые из этих полей (или выражения, включающие эти поля) также могут быть определены во фразе ORDER.

Если слова ALL UNIQUE заданы непосредственно после фразы CONVERT, то сформированная выходная строка передается в приемник, только если она отличается от последней переданной в приемник строки.

Если вторая или последующая заданная в команде таблица используется с индексом, то для этой таблицы фраза «FOR ⟨условие⟩» может быть заменена фразой PLUCK для этого индекса, что заменяет традиционное соответствие полей. Синтаксис фразы PLUCK аналогичен одноименной команде, однако имя таблицы не задается. Таблица уже должна быть определена в команде CONVERT. Индекс, который должен быть использован для доступа к этой таблице, должен быть открыт.

3.3.6. Команда CREATE

Синтаксис команды

CREATE [⟨позиция⟩] [RECORD] [{FOR | WHERE} {⟨таблица⟩ | ⟨вр_табл⟩}] [WITH ⟨форма⟩ | WITH {⟨поле⟩}]

Описание команды

После определения таблицы (с помощью команды DEFINE) или после ее открытия (с помощью команды USE) можно в интерактивном режиме осуществлять создание новых записей в таблице. Этот процесс осуществляется с помощью команды CREATE.

При вызове команды CREATE указывается имя таблицы, в которой создаются записи. Если имя таблицы не специфицировано, система предполагает, что новая запись должна быть создана в текущей таблице по умолчанию. KMan очищает экран и выводит на дисплей имена полей таблицы. Эти имена полей играют роль подсказок. После появления всех подсказок на экране курсор останавливается в точке с координатами (1,1). Нажатие клавиши ВВОД перемещает курсор к первой подсказке. Теперь можно набирать значение первого поля новой записи. После завершения набора необходимо нажать клавишу ВВОД, и курсор переместится к имени второго поля. Теперь набирается значение этого поля и т. д., до тех пор пока не будет введено значение для каждого поля новой записи. Эта новая запись становится последним входом в таблицу, и система рассматривает ее в качестве текущей записи таблицы. После этого система выдает подсказки на ввод следующей записи (для выхода из режима CREATE нужно нажать клавишу СНЯТИЕ в тот момент, когда курсор находится в левом верхнем углу экрана).

Служебное поле #MARK при создании записей автоматически получает значение ЛОЖЬ.

При попытке создать записи в таблице, к которой у пользователя нет доступа по записи, KMan защищает таблицу от создания записей, не позволяя выполнить команду CREATE.

Необязательный оператор «WITH ⟨форма⟩» приводит к тому, что KMan выдает на дисплей указанную экранную форму (созданную пользователем заранее) и принимает значения данных с помощью этой формы, а не по стандартным подсказкам команды CREATE.

В ответ на ввод команды пакет отвечает выводом на экран номера создаваемой записи, списка имен всех действительных (т. е. невиртуальных) полей записи, к которым имеется доступ по записи, и всех определенных в шаблоне каждого поля констант. В том случае, если явно специфицированы поля с помощью параметра WITH, подсказки возникают только для указанных полей. После этого курсор будет позиционирован в левом верхнем углу экрана терминала. Нажатие клавиши ВВОД приводит к перемещению курсора к подсказке первого поля.

Создание действительных полей обусловливается защитой доступа по записи поля, специфицированной в процессе определения таблицы. Если пользователь не обладает доступом к записи некоторых полей таблицы, KMan не выдает подсказки для ввода значений таких полей в процессе создания записи. Защищенные поля в новой записи имеют неопределенные значения.

Во время создания записи система выдает подсказки только для имен действительных полей. Подсказки для значений виртуальных полей не выдаются, так как эти поля не занимают память в таблице. В том случае, если переменная E.ICOM имеет значение ИСТИНА, значения виртуальных полей немедленно вычисляются и выдаются на дисплей, как только создана запись. (Значения виртуальных полей автоматически вычисляются всякий раз, когда они в последующем «выбираются» из таблицы.) Далее можно явно специфицировать список тех полей, для которых выводятся подсказки, если подсказки для всех полей таблицы не нужны.

Может оказаться, что все поля (т. е. подсказки для всех полей), не помещаются на одном экране в процессе создания записи. В этой ситуации KMan предоставляет несколько экранов продолжения, которые содержат подсказки для полей этой таблицы. Наличие экранов продолжения приводит к выдаче сообщения «Еще данные». Предоставляются две клавиши в целях обеспечения быстрого перемещения по экранам продолжения. Нажатие клавиши ВНИЗ (^X) в тот момент, когда курсор позиционирован в точке с координатами (1,1), приводит к выдаче на экран дисплея следующего экрана продолжения. В том случае, если уже «исследован» последний из экранов продолжения записи, команда ВНИЗ ведет себя аналогично клавише ВПЕРЕД (^F), выдавая на дисплей значения для следующей записи в указанной области. Нажатие клавиши ВВЕРХ (^E) в тот момент, когда курсор позиционирован в точке с координатами (1,1), приводит к выдаче на дисплей предыдущего экрана продолжения. В том случае, если продолжения предыдущего экрана нет, на дисплей выводится последний экран продолжения записи. Клавиши ^E и ^X не оказывают никакого воздействия в тот момент, когда курсор позиционирован в точке, не обладающей координатами (1,1), или когда для записи не требуется экран продолжения.

3.3.7. Команда DEFINE

Синтаксис команды

DEFINE [PROTECT | UNPROTECTED] {⟨таблица⟩ [WITH "⟨файл .itb⟩"] | [⟨таблица⟩] WITH "⟨файл .itb⟩"} [⟨коды доступа⟩] {[FIELD ⟨поле-1⟩ {STR ⟨ цел ⟩ [= ⟨с_выр⟩] [USING "⟨шаблон⟩"] | NUM [= ⟨ч_выр⟩] [USING "⟨шаблон⟩"] | INT [= ⟨ч_выр⟩] [USING "⟨шаблон⟩"] | LOGIC[= ⟨л_выр⟩] [USING "⟨шаблон⟩"] } [⟨коды доступа⟩] [LABELED"⟨метка⟩"];] | [⟨поле-2⟩, ⟨поле-3⟩;] | [\⟨поле-4⟩[, ⟨поле-5⟩];]} ENDDEF

Описание команды

Команда DEFINE используется для интерактивного определения таблицы. Система предоставляет пользователю возможность специфицировать имя и дисковод файла, в котором физически будет размещена таблица. После этого будет выдана подсказка для определения всех полей, которые должны быть включены в таблицу. KMan позволяет определять и использовать неограниченное количество таблиц.

KMan запрашивает информацию о каждом поле таблицы: его имени, типе (STR, NUM, INT, LOGIC) и метке. Для строкового поля можно задать размер (до 65534 символов). Для логических или числовых полей специфицировать размер не надо. После завершения определения поля (посредством нажатия клавиши ВВОД) KMan выдает подсказку на ввод следующего поля. В том случае, если вводится имя поля, которое уже существует в данной таблице, происходит переопределение предыдущего определения. Для каждой таблицы автоматически определяется логическое служебное поле с именем #MARK. Таблица может содержать не более 255 полей.

Можно указать ключевое слово UNPROTECTED непосредственно за словом DEFINE, если нет необходимости ограничивать доступ к таблице существующими значениями кодов доступа (файл KPASS .IGU).

После определения таблицы можно дополнительно специфицировать ее коды доступа по чтению и/или записи. KMan использует эти коды для определения всех пользователей, которые могут получать (читать) данные из таблицы и изменять (записывать) содержимое данных таблицы. Кроме этих характеристик безопасности можно определять с помощью шаблонов условия целостности и редактирования для каждого поля. Система автоматически осуществляет проверку для всех значений по мере их заполнения в указанной таблице.

Код доступа представляет собой любую букву в диапазоне A — P. Ограничение кода доступа для таблицы специфицируется в виде группы кодов доступа. Коды доступа по чтению таблицы должны представлять собой подмножество определяющих кодов доступа по чтению, принадлежащих пользователю. Если для таблицы не специфицирована группа кодов по чтению, то система предполагает, что коды чтения таблицы имеют те же значения, что и определяющие коды доступа по чтению пользователя. Ограничения доступа по записи для таблицы специфицируются в виде группы кодов доступа аналогичным образом. Эта группа может отличаться от группы кодов доступа по чтению той же таблицы.

После определения таблицы система предоставляет доступ к чтению (записи) любому пользователю, который обладает такими кодами доступа к чтению (записи), как и соответствующая группа кодов доступа таблицы.

Для любого поля по желанию может быть объявлен шаблон. При вводе нового значения в поле таблицы KMan осуществляет проверки целостности и редактирования, указанные с помощью шаблона. Шаблон, кроме того, служит в качестве формата для значений

поля при их выборке из таблицы и отображении на дисплее. В тех случаях, когда шаблон не специфицирован, предполагается шаблон, заданный по умолчанию.

3.3.8. Команда DESTROY

Синтаксис команды

DESTROY { ⟨таблица⟩ | ⟨вр_табл⟩ } WITH "⟨файл .itb⟩"

Описание команды

Команда DESTROY предназначена для уничтожения таблицы. Файл, содержащий уничтожаемую таблицу, физически удаляется. В момент выдачи команды удаляемая таблица должна быть открыта, в противном случае команда DESTROY не сработает. Нельзя уничтожить таблицу, не имея доступа по записи для всех ее полей.

3.3.9. Команда FINISH

Синтаксис команды

FINISH [ALL | ⟨таблица⟩ | ⟨вр_табл⟩ | LIBRARY "⟨файл .ilf"⟩

Описание команды

Команда предназначена для завершения работы с таблицей. Параметр ⟨таблица⟩ определяет имя открытой таблицы, которая не требуется для дальнейшей работы.

При закрытии таблицы, заданной по умолчанию, новой таблицей, заданной по умолчанию, становится та таблица, которая была открыта раньше других. Если в команде FINISH не специфицированы ни имя таблицы, ни параметр ALL, закрывается текущая таблица, заданная по умолчанию.

3.3.10. Команда IMPRESS

Синтаксис команды

IMPRESS ⟨таблица-1⟩ TO ⟨таблица-2⟩ [WITH "⟨файл .itb⟩"]

Описание команды

Эта команда позволяет применить определение существующей таблицы для таблицы, которой до настоящего времени не было.

Параметр ⟨таблица-1⟩ задает имя ранее определенной таблицы, а параметр ⟨таблица-2⟩ — имя новой таблицы, определение которой генерируется с помощью команды IMPRESS. Определение новой таблицы идентично определению исходной таблицы. Новой таблице присваивается указанное имя. В том случае, если необязательная фраза WITH "⟨файл .itb⟩" опущена, таблица размещается в файле с таким же именем и расширением .itb.

3.3.11. Команда INDEX

Синтаксис команды

INDEX "⟨файл .ind⟩" [FOR {⟨таблица⟩ | ⟨вр_табл⟩}] [BY] {порядок⟩}*

Описание команды

Команда предназначена для индексирования таблицы.

Параметр ⟨файл .ind⟩ определяет файл, который содержит индекс, построенный системой KMan для таблицы с именем, заданным параметром ⟨таблица⟩. Если для построения индекса специфицировано имя файла, который уже существует, то система откажется строить индекс. Если фраза "FOR ⟨таблица⟩" опущена, то индекс генерируется для текущей таблицы по умолчанию.

Индекс сортируется по указанному параметру ⟨порядок⟩ в направлении ASCENDING или DESCENDING. Каждое выражение включает в себя одно или более полей указанной таблицы. Слово BY является необязательным.

Индекс может быть построен по комбинации направлений ASCENDING и DESCENDING. Для этого необходимо специфицировать направление и последовательность из одного или более выражений (каждое выражение может включать в себя одно или более полей), противоположное направление и последовательность из одного или более выражений, снова первое направление и последовательность выражений и т. д. Вся последовательность выражений составляет индексный ключ. Если переменная E. ICAS имеет значение ИСТИНА, то в процессе формирования индекса различия между верхним и нижним регистрами в значении индексных ключей игнорируются.

3.3.12. Команда LIST

Эта команда является синонимом команды SELECT.

3.3.13. Команда MARK

Синтаксис команды

MARK [RECORD | RECORDS] [IN {⟨таблица⟩ | ⟨вр_табл⟩}
[WITH⟨выр⟩] [{FOR | WHERE | WHILE}
{⟨условие⟩}] [⟨пределы⟩]

Описание команды

Команда MARK позволяет отметить запись в таблице с помощью логического значения TRUE или FALSE, помещая ее в одну из двух этих категорий. При создании запись автоматически помечается с помощью FALSE. Запись, отмеченную как TRUE, можно затем переотметить (т. е. отметить с помощью FALSE), используя команду UNMARK. Все записи, отмеченные с помощью TRUE, могут быть удалены из таблицы с помощью команды COMPRESS.

При определении структуры таблицы в нее автоматически включается специальное поле #MARK. Это поле не может быть удалено с помощью команды REDEFINE. Отметка записи с помощью FAL-

SE (TRUE) приводит к тому, что значением поля #MARK этой записи становится FALSE (TRUE). Отмеченная запись становится текущей записью таблицы.

Если переменная E. IMRK равна FALSE, отмеченные записи могут искаться и модифицироваться. Однако если E. IMRK равна TRUE, KMan игнорирует существование отмеченных записей почти для всех команд обработки. Исключениями являются команды UNMARK, COMPRESS, INDEX и SORT (если таблица сортируется в исходном файле).

Фраза "FOR ⟨условие⟩" указывает, что нужно отмечать лишь те записи в заданных пределах, которые удовлетворяют заданным условиям. Эти записи отмечаются с помощью значения заданного логического выражения. Если вместо фразы FOR используется WHILE, отметка заканчивается на записи, не удовлетворяющей условиям. Если с фразой WHILE не заданы пределы, отметка начинается с текущей записи.

Нельзя отмечать записи таблицы, не имея доступа по записи ко всем ее полям.

3.3.14. Команда OBTAIN

Синтаксис команды

OBTAIN {[⟨позиция⟩] | 0} {RECORD | RECORDS} FROM {⟨таблица⟩ | ⟨вр_табл⟩} [{FOR | WHERE} {⟨условие⟩}] [AT | @] ⟨коорд⟩{? | OUTPUT} { ⟨выр⟩[USING "⟨шаблон⟩"] }

Описание команды

Команда OBTAIN находит заданную запись и выводит ее на экран. Параметр ⟨позиция⟩ указывает, какую запись KMan должен извлечь из заданной таблицы. Если параметр ⟨таблица⟩ опущен, подразумевается текущая по умолчанию таблица БД. Слово RECORD можно не указывать.

Система отвечает выводом на экран номера найденной записи, списком имен всех полей и значений данных, хранимых в каждом поле этой записи.

Если ни одна из позиций в команде не задана, KMan выдает на экран текущую запись. В любом случае выданная запись становится текущей записью таблицы.

Команда OBTAIN может содержать одно или более условий для получения записи, удовлетворяющей установленному критерию. Условие — это обычное логическое выражение. Таблица может содержать несколько записей, удовлетворяющих условию. Заданный параметр ⟨позиция⟩ определит, какую именно запись выдаст KMan.

Если параметр ⟨позиция⟩ в команде OBTAIN не задан вообще, то первая запись, удовлетворяющая условию, выдается на экран и становится текущей записью таблицы. Если же таблица не содержит ни одной записи, удовлетворяющей заданным условию и позиции, KMan выдает соответствующее сообщение. Положение текущей записи при этом не изменяется.

Если при определении таблицы ее полям были назначены шаблоны, то KMan осуществляет соответствующее редактирование значений этих полей при обработке команды OBTAIN.

Если таблица открыта совместно с ее индексом, то при выдаче записи учитывается последовательность записей, основанная на порядке их сортировки в индексе, а не физическая последовательность записей в таблице. Если при этом необходимо получить запись, основываясь на ее физическом расположении в таблице, то для отключения индексного поиска нужно ввести

OBTAIN FIRST
OBTAIN NEXT FOR CURREC (⟨имя таблицы⟩) = ⟨номер записи⟩.

Так как KMan в своей работе использует технику виртуальной памяти, в оперативной памяти (ОП) ПЭВМ он помещает свои буферы. Задавая команду OBTAIN с нулевым аргументом, можно заставить KMan очистить все буферы в ОП. Это особенно полезно после выполнения больших и сложных изменений в таблице, так как гарантирует запись всех изменений в таблицу. Этот способ обеспечивает минимизацию потерь данных при сбоях системы и оборудования.

После выполнения команды OBTAIN 0 таблица не имеет текущей записи. Это значит, что функция CURREC (⟨имя таблицы⟩) равна 0, а функция EOT (⟨имя таблицы⟩) принимает значение ЛОЖЬ.

3.3.15. Команда PLUCK

Синтаксис команды

PLUCK {⟨выр⟩}* [FROM {⟨таблица⟩ | ⟨вр_табл⟩}] [USING "⟨файл .ind⟩"]

Описание команды

Команда PLUCK может быть использована как альтернатива команды OBTAIN для поиска в таблице необходимой записи. Найденная запись становится текущей записью таблицы. KMan очень быстро находит требуемую запись таблицы, используя индекс. Этот индекс должен быть предварительно создан командой INDEX, а затем открыт совместно с обрабатываемой таблицей с помощью команды USE. Если для поиска записей используется не первичный индекс, то имя файла, содержащего этот индекс, должно быть явно указано в команде PLUCK.

Последовательность выражений для команды PLUCK должна соответствовать последовательности выражений (например, имен полей), определенных для индекса. В команде PLUCK нужно определить выражение для каждого из индексных выражений. KMan вычислит значение каждого заданного выражения, чтобы затем быстро найти запись в таблице, имеющую такие же значения в своих индексных выражениях.

Если PLUCK выражение — это просто поле, то его шаблон (или размер) определяет число символов, используемых при индексном поиске. Для более сложных выражений значение переменной E. LSTR определяет, сколько символов в значении выражения используется при поиске записей.

Если значения выражений команды PLUCK соответствуют индексному ключу, то требуемая запись считается найденной, а значение служебной переменной #FOUND становится равным ИСТИНЕ.

Если значения выражений команды PLUCK не соответствуют никакому индексному ключу, переменная #FOUND устанавливается в ЛОЖЬ, а KMan извлекает запись, индексный ключ которой непосредственно следует за заданным индексным ключом, если таковой существует.

Если более чем одна запись имеет значение индексных полей, равное значению PLUCK выражения, то первая из этих записей будет выдана, а для получения остальных надо использовать команду OBTAIN NEXT.

3.3.16. Команда REDEFINE

Синтаксис команды

```
REDEFINE [PROTECT | UNPROTECTED] [⟨таблица⟩ | ⟨вр_таблица⟩] [WITH "⟨файл .itb⟩"] [⟨коды доступа⟩] {[+]
⟨поле-1⟩ {STR ⟨цел⟩ [= ⟨с_выр⟩]
[USING "⟨шаблон⟩"] | NUM [= ⟨ч_выр⟩] [USING "⟨шаблон⟩"] | INT [= ⟨ч_выр⟩] [USING "⟨шаблон⟩"] | LOGIC
[= ⟨л_выр⟩] [USING "⟨шаблон⟩"]} [⟨коды доступа⟩] [LABELED "метка"] ; ] | [⟨поле-2⟩,⟨поле-3⟩;] | [\⟨поле-4⟩ [,
⟨поле-5⟩]; ) } ENDDEF
```

Описание команды

С помощью команды REDEFINE можно переопределить характеристики существующей таблицы или любого ее поля.

3.3.17. Команда RENAME

Синтаксис команды

```
RENAME { ⟨таблица-1⟩ | ⟨вр_табл⟩} TO ⟨таблица-2⟩
```

Описание команды

Команда RENAME используется для изменения имени существующей таблицы. Перед переименованием таблица должна быть открыта. После переименования таблица остается открытой.

3.3.18. Команда SELECT

Синтаксис команды

```
SELECT [ ALL UNIQUE ] [{[UNIQUE] ⟨выр⟩ [USING "⟨шаблон⟩" ] }*] [{FROM{⟨таблица⟩ | ⟨вр_табл⟩} [ [ {FOR |
WHERE | WHILE} {⟨условие⟩ } ] | [PLUCK ⟨выр-2⟩
[WITH "⟨файл .ind⟩" ] ] [⟨пределы⟩] [ {GROUP BY|
BY}{⟨разрыв⟩}* [ORDER BY {⟨порядок⟩}* ]
```

Описание команды

Команда SELECT генерирует выходную таблицу с заданными характеристиками. KMan автоматически создает выходную таблицу, используя значения тех полей таблицы (или нескольких таблиц, если необходимо), которые удовлетворяют заданному условию. Команда SELECT может обрабатывать несколько таблиц БД одновременно. Все операции реляционной алгебры могут выполняться над определенными пользователем данными. Можно задавать различные способы вывода полученных результатов.

В ответ на эту команду KMan начинает рассматривать все записи в указанных пределах таблицы, определяемой параметром ⟨таблица⟩. Для каждой из этих записей KMan вычисляет и выводит на экран значения каждого из заданных выражений. Поле; арифметическое выражение, включающее числовые поля; строковое выражение, включающее строковые поля; логическое выражение, включающее логические поля,— вот наиболее типичные примеры выражений, используемых в команде SELECT. Если никаких выражений в команде не задано (или задана ✳), то выводятся значения всех полей каждой записи, за исключением поля #MARK.

Заданная в команде таблица становится новой стандартной таблицей по умолчанию. Если фраза "FROM ⟨таблица⟩" пропущена, используется текущая таблица по умолчанию. Термины LIST и SELECT равнозначны.

Каждая строка в выходной таблице составлена из значений одной записи таблицы БД из указанных параметром "пределов". Общий вид и направление вывода выходной таблицы управляются текущим значением следующих переменных: E.STAT, E.PNT, E.SPAC, E.PAUS, E.OCON, E.ODSK, E.OPRN, E.SPGN, E.LEGH, #TITLE. Одна команда SELECT может содержать до 255 выражений. Если заданные значения не помещаются на одной строке экрана, они продолжают выводиться на следующих строках. Колонки выходной таблицы выстраиваются в том же порядке, что и выражения в команде. Если выражение состоит только из одного поля, то соответствующая колонка выходной таблицы имеет заголовок, состоящий из имени этого поля. Для других типов выражений KMan проставляет стандартные заголовки (T01, T02, . . .).

Если переменная E.STAT равна ИСТИНА, для каждой колонки выходной таблицы подсчитывается и выводится на экран статистика. Каждая статистическая величина запоминается в соответствующей служебной переменной (например, #AVER). Значения этих переменных не вычисляются, если значение переменной E.STAT равно ЛОЖЬ.

KMan рассматривает все записи, удовлетворяющие заданным условиям и расположенные в заданных пределах таблицы с указанным именем. Выходная таблица содержит столбец для каждого выражения, заданного параметром ⟨выр⟩. Если указан шаблон, то соответствующее выражение будет отредактировано.

Фраза "FOR" в команде SELECT может быть заменена "WHERE". Если задано "WHILE", обработка команды прерывается, как только встретится запись, не удовлетворяющая условию.

Команда SELECT позволяет задавать условие выбора записей из таблицы. Условие — это обычное логическое выражение. Если условие задано с фразой WHILE, то первая же запись, для которой условие не будет выполнено, вызовет конец обработки команды. Если после фразы WHILE никакой предел не задан, поиск начнется с текущей записи.

Если для выражения задан шаблон, то он управляет редактированием значения выражения при выводе на экран и шириной соответствующего столбца выходной таблицы. Числовые значения и статистика заменяются звездочками, если эти значения слишком велики для заданной шаблоном ширины колонки.

При выборе информации из многих таблиц с помощью фраз FROM должно быть явно задано имя каждой таблицы, поля которой упоминаются в любом из выражений команды SELECT. Условия выбора записей могут быть заданы для каждой такой таблицы; условие может содержать ссылки на поля этой таблицы плюс ссылки на поля любой таблицы, объявленной в данной команде в предыдущей фразе FROM. Если в такой ссылке упоминается имя поля не из первой таблицы (т. е. теперь — таблицы по умолчанию), то перед ним через точку нужно указывать имя соответствующей таблицы.

Условие WHILE может быть задано только для первой таблицы. Условие для второй (и последующих) таблицы должно включать указание, как определить запись в этой таблице, связанную с найденной записью в предыдущей таблице. Это достигается путем задания в условии поля одной таблицы, соответствующего полю в другой (например, СЛУЖАЩИЕ.НОМОТД=ОТДЕЛЫ. НОМОТД). KMan эффективно осуществляет слияние двух таблиц, основываясь на соответствующих друг другу значениях полей, соединяя вместе записи с одинаковыми значениями.

Для каждого набора соединенных записей KMan выдает список значений для каждого из заданных в команде SELECT выражений, выполняя при этом обусловленное шаблоном редактирование.

В случае получения списка записей из одной таблицы последняя из записей внутри заданных пределов, удовлетворяющая заданным условиям, становится текущей записью. При мультитабличной обработке в каждой из таблиц последняя запись, удовлетворяющая условиям, становится текущей.

Любое из выражений может предваряться служебным словом UNIQUE для подавления вывода последующих значений выражения, если они точно такие же, как и предыдущие. Чтобы задать такой режим для всех выражений в команде, нужно указать фразу ALL UNIQUE сразу же после слова SELECT. Если в этом случае вся строка выходной таблицы идентична предыдущей, она полностью подавляется.

Необязательная фраза ORDER BY вызывает сортировку сгенерированной выходной таблицы по значениям заданного выражения сортировки, располагая записи в указанном направлении (ASCENDING или AZ задают направление по возрастанию, а DESCENDING или ZA — по убыванию). В команде может быть задана комбинация двух, трех и более направлений.

Другой необязательной фразой команды SELECT является фраза GROUP BY ⟨разрыв⟩, где параметр ⟨разрыв⟩ содержит одно или более имен полей. Задание этой фразы вызывает приостановку выполнения команды всякий раз при изменении выходного значения поля прерывания. Если фраза ORDER BY есть в команде, остановка осуществляется по смене значений в выходной таблице. Каждая приостановка (прерывание) выполнения команды сопровождается распечаткой статистики.

Если команда SELECT использует только одну таблицу БД, то выходная таблица может быть упорядочена по любому из ее полей (включая те, величины которых не появляются в выходной таблице). Выражения, включающие эти поля, также допустимы при задании источника упорядочения. При мультитабличной обработке фраза "ORDER" также может включать любое из полей или выражения, содержащие любые поля, независимо от включения этих полей в выходную таблицу.

В прерываниях обработки (GROUP BY) может быть задано одно или более полей. Когда KMan обнаруживает прерывание (изменение значения полей выходной таблицы) и переменная E.STAT равна ИСТИНА, то последнее прерывание вычислит и выведет на экран всю статистику. Если E. SECB=FALSE, то после этого будет осуществлен "перевод листа". При задании прерываний слово GROUP является необязательным.

Если задание условий (фраза FOR) заменяется фразой PLUCK, ее синтаксис полностью идентичен синтаксису одноименной команды, за исключением того, что ссылка на таблицу здесь недопустима: таблица уже задана фразой FROM. Индекс, используемый для доступа к этой таблице, должен быть открыт.

3.3.19. Команда SHOW

Синтаксис команды

SHOW [⟨таблица⟩ | ⟨вр_табл⟩ | ⟨форма⟩ | FORM | ⟨макро⟩ | MACRO | ⟨перем⟩ | VARIABLE | ARRAY | ENVIRONMENT | FUNCTION | PERFORM]

Описание команды

Кроме базисной информации о структуре таблицы система KMan предоставляет пользователю информацию о шаблонах, которые были определены для полей, и о кодах доступа по чтению — записи таблицы. Однако в том случае, если таблица обладает кодом доступа по чтению — записи, которого у пользователя нет, то этот код не отображается.

Вместо спецификации имени таблицы можно использовать спецификацию имени макроса. В том случае, если имеются синонимы ключевых слов системы, которые представлены в тексте макроса, система покажет самый короткий синоним. Например, вместо ASCENDING будет показано AZ, а вместо SELECT — LIST. Другой альтернативой является задание имени формы, используемой в текущем сеансе работы с системой KMan. Это приводит к выводу на дисплей объявления формы. Если переменная E.ODSK имеет значение ИСТИНА, то объявление будет записано (т. е. сохранено) в файле, указанном переменной #DISKOUT. Это позволяет в последующем редактировать этот файл с помощью текстового редактора и использовать его в дальнейших сеансах работы с системой с помощью команды PERFORM.

Команда SHOW может быть использована для просмотра характеристик специфицированной переменной. С ее помощью может быть показана природа переменной (например, переменная окружения, рабочая переменная, массив, элемент массива, ячейка электронного бланка и т. д.), тип переменной и ее шаблон. При показе шаблона система KMan может использовать сокращенные обозначения. Например, %30r указывает на шаблон, состоящий из 30 указателей позиции типа r. Для переменных ячейки показываются также стиль представления (если он вообще имеется) и коды доступа (если пользователь ими обладает).

Команда SHOW может использоваться для получения списка имен всех макросов, форм или переменных, которые используются в текущем сеансе работы с системой, но не тех, которые вообще хранятся в памяти.

3.3.20. Команда SORT

Синтаксис команды

SORT ⟨таблица⟩ | ⟨вр_табл⟩} [TO "⟨файл.itb⟩"] BY "⟨порядок⟩"}

Описание команды

После того как в таблице были созданы записи, иногда возникает потребность в переупорядочении последовательности записей на основании значений некоторого поля (или полей). Этот процесс можно осуществить с помощью команды SORT, которая выполняет сортировку записей таблицы и располагает их в последовательности, основанной на значении некоторого поля (или полей или выражений).

Система помещает результирующую отсортированную версию таблицы в указанный файл. Имя файла должно быть полностью квалифицировано. В том случае, если целевой файл не указан, вновь отсортированная версия таблицы записывается вместо оригинальной версии таблицы. В противном случае эта команда приводит к созданию переупорядоченной версии таблицы в отдельном файле. Маркированные записи при сортировке пропускаются, если E. IMRK равна ИСТИНА. Эта переупорядоченная версия является новой таблицей, которая может быть использована точно так же, как и первоначальная таблица.

Предупреждение: нельзя специфицировать файл, содержащий первоначальную версию той таблицы, которая должна быть подвергнута сортировке, в качестве целевого файла в команде SORT.

Можно определить направление сортировки, которое будет использоваться для сортировки с помощью фраз ASCENDING (возрастание) или DESCENDING (убывание). Можно использовать AZ в качестве сокращенной записи слова ASCENDING и ZA в качестве сокращенной записи слова DESCENDING. После определения направления специфицируются выражения, которые будут использоваться при сортировке. Та последовательность, в которой специфицированы эти выражения, определяет их предшествование (порядок сортировки). Самое левое выражение обладает самой высокой степенью предшествования, за ним следует второе выражение слева и т. д. Если необходимо, чтобы система игнорировала различия между верхним и нижним регистрами при сортировке, следует задать переменной определения среды E.ICAS значение ИСТИНА. В том случае, если выражение состоит только из поля, шаблон поля (или размер) определяет количество символов, которое будет использоваться в целях сортировки. Для более сложных выражений значение переменной E.LSTR определяет то количество символов в значении выражения, которое будет использоваться при выполнении сортировки.

Текущая запись первоначальной таблицы остается текущей записью этой таблицы и после сортировки. Любой индекс, построенный для первоначальной таблицы, неприменим ко вновь отсортированной версии этой таблицы. Однако если задан целевой файл для отсортированной версии таблицы, все индексные файлы, поддерживающие оригинальную таблицу, сохраняют свою актуальность.

Таблица может быть отсортирована по комбинации возрастающего и убывающего направлений, задавая направление и последовательность из одного или более выражений, затем обратное (противоположное) направление и последовательность из одного или более выражений, снова первое направление и последовательность из одного или более выражений и т. д.

Текущая запись первоначальной таблицы остается текущей записью для этой же таблицы после сортировки.

3.3.21. Команда UNMARK

Синтаксис команды

UNMARK [IN {⟨таблица⟩ | ⟨вр_табл⟩} [WITH⟨выр⟩] [{FOR | WHERE | WHILE} {⟨условие⟩}] [⟨пределы⟩]

Описание команды

Команда UNMARK позволяет переотметить запись, ранее отмеченную с помощью TRUE. В результате запись отмечается как FALSE (т. е. ее служебное поле #MARK получает значение FALSE). Значение переменной E.IMRK не влияет на работу команды UNMARK.

3.3.22. Команда USE

Синтаксис команды

USE {⟨таблица⟩ | " файл.itb⟩" | ⟨с_выр⟩} [AS⟨вр_табл]
[WITH{"⟨файл.ind⟩"}*]

Описание команды

Перед тем как использовать таблицу в рамках сеанса работы с системой KMan, надо сообщить системе, что необходимо использовать (или открыть) данную таблицу. Это довольно просто сделать с помощью команды USE. Существует одно исключение: если таблица была определена (с помощью команд DEFINE или REDEFINE) в рамках данного сеанса работы с системой, то ее уже можно использовать и необходимость в применении команды USE отпадает.

Первая запись таблицы (если она существует) становится текущей, или, другими словами, той записью, на которой в данный момент времени спозиционирована система, и, следовательно, значение этой записи становится текущим значением соответствующих переменных полей.

Иногда может возникнуть потребность, чтобы файл, содержащий таблицу, временно оказался не на том дисководе, на котором он был первоначально специфицирован. Это можно легко осуществить командой USE посредством спецификации имени файла (в кавычках), а не имени таблицы. Неквалифицированная часть файла при этом будет использоваться в качестве имени таблицы. В том случае, если расширение не специфицировано, предполагается (по умолчанию) расширение файла .ITB. Б том случае, если специфицирован только разделитель расширения (точка "."), не предполагается никакое расширение. Система KMan полагает, что она обнаружит таблицу на дисководе, указанном именем файла; это верно до тех пор, пока не указано завершение обработки данной таблицы (с помощью команды FINISH). В том случае, если дисковод не указан, предполагается дисковод, заданный по умолчанию.

Нет никаких ограничений на количество таблиц, которые можно использовать одновременно.

В момент сообщения системе о необходимости открыть таблицу можно также специфицировать и временное альтернативное имя таблицы, используемое вместо обычного имени. Альтернативное имя удобно при использовании таблицы, имя которой слишком длинное. С помощью команды USE может быть специфицировано более короткое имя. Это альтернативное имя таблицы распознается системой до тех пор, пока не будет указан факт завершения работы с данной таблицей (команда FINISH).

После применения команды USE с указанием индекса текущей записью станет та запись, на которую выполняется ссылка с помощью индекса, а не первая физически запись таблицы.

3.4. Общеалгоритмические команды

3.4.1. Общие характеристики и возможности

3.4.1.1. Перечень команд

Процедура представляет собой последовательность команд, которые содержатся в файле операционной системы. Такой файл обычно называется командным файлом. В том случае, если имеется последовательность команд, которую нужно повторно выполнять, следует построить командный файл, содержащий такую процедуру. Такой файл обычно строится с помощью команды TEXT (при наличии дополнительной компоненты — редактора текстов пакета) или с помощью внешнего текстового редактора. Хотя данная процедура специфицируется только один раз, можно заставить пакет КМап выполнять ее несколько раз, а точнее — всегда, когда это нужно.

В данном разделе описаны следующие (общеалгоритмические) команды управления программой:

PERFORM — вызов процедуры

RETURN — возврат из процедуры, выполняемой в текущий момент времени с тем, чтобы выполнить команду, которая следует за PERFORM

STOP — остановка обработки всех процедур; выдача подсказки пакета

WAIT — остановка перед обработкой следующей команды

LET — присвоение переменной требуемого значения

ROOT — присвоение переменной значения корня выражения

RELEASE — освобождение памяти, занятой всеми или указанными макросами, формами и рабочими переменными

WHILE-DO — продолжение итерации специфицированного множества команд, до тех пор пока логическое выражение равно TRUE

TEST-CASE — проверка значения указанного выражения и определение соответствия этого значения одному из указанных случаев; выполнение того множества команд, которое указано для первого из случаев, значение которого совпадает с указанным (и для всех последующих случаев), до тех пор пока не будет обнаружена команда BREAK

IF-THEN-ELSE — вычисление логического выражения; если оно имеет значение TRUE — выполнение одного множества команд, в противном случае — альтернативного множества команд

CONTINUE — продолжение работы со следующей итерацией

BREAK — прерывание выполнения цикла или случая

SAVE — сохранение текущего состояния обработки в контекстном файле

LOAD — загрузка контекстного файла и возобновление обработки

BYE — завершение взаимодействия с системой

3.4.1.2. Множество команд в одной и той же строке

В одной строке может быть задано более одной команды при том условии, что команды отделяются друг от друга точкой с запятой (;).

3.4.1.3. Комментарии

Начало комментария обозначается посредством /*, а конец — *\. Комментарии не существуют в рамках строковых констант. Все комментарии игнорируются. Кроме того, восклицательный знак (!), не заключенный в двойные кавычки и не находящийся внутри комментария, приводит к тому, что остаток строки будет проигнорирован.

3.4.1.4. Локальные переменные, макросы и формы

В рамках любой процедуры можно определить рабочие переменные, которые будут локальными для данной процедуры. Это означает, что эти переменные могут существовать только при выполнении этой процедуры. Таким образом, переменная, являющаяся локальной для данной процедуры, неизвестна любым другим процедурам, которые случайно вызовут эту конкретную процедуру.

Любая рабочая переменная, которая явно не объявлена локальной, предполагается глобальной. Глобальная переменная не исчезает при завершении выполнения процедуры. Нерабочая переменная (т. е. переменная поля, переменная ячейки и предопределенная переменная) не может быть объявлена локальной для любой процедуры. Эти переменные всегда являются глобальными.

Для того чтобы объявить массив локальной переменной, слово LOCAL должно быть вставлено перед словом DIM в объявление массива. Хотя слово LOCAL используется для объявления одной переменной с единственным значением за один раз, она может быть использована для объявления многих локальных массивов. Следовательно,

```
LOCAL DIM X (10), Y (3,15)
```

эквивалентно

```
LOCAL DIM X (10)
LOCAL DIM Y (3,15)
```

В том случае, если массив явно не объявлен, он считается глобальным.

При определении макроса в рамках процедуры можно указать, что этот макрос должен быть локальным для данной процедуры. Как и в случае локальных переменных, это означает, что макрос распознается только при выполнении конкретной процедуры. При завершении выполнения этой процедуры все определенные в ней

локальные макросы исчезают. Глобальные макросы не изменяются. Макрос считается локальным в том случае, если при определении перед MACRO было указано LOCAL. Например,

LOCAL MACRO ФО фио, оклад

определяет локальный макрос с именем ФО.

Формы также могут быть объявлены локальными при их определении в рамках процедуры. Это можно осуществить с помощью простой вставки слова LOCAL перед словом FORM при определении формы. Например,

LOCAL FORM FIFORM . . . ENDFORM

определяет локальную форму с именем FIFORM. Локальная форма распознается только на протяжении выполнения той процедуры, в которой она была объявлена. При завершении выполнения процедуры все локальные для данной процедуры формы исчезают. Глобальные формы остаются без изменений.

3.4.2. Команда BREAK

В рамках последовательности команд WHILE или TEST команда BREAK приводит к тому, что пакет немедленно прерывает текущую итерацию команды WHILE и переходит к выполнению команды, которая следует непосредственно за соответствующим словом ENDWHILE, или последовательность командных операторов в команде TEST и переходит к обработке команды, которая следует непосредственно за ENDTEST.

3.4.3. Команда BYE

Эта команда приводит к завершению всех видов обработки и появлению на экране подсказки операционной системы.

Если не использовался меню-ориентированный интерфейс, все таблицы, контекстные и командные файлы закрываются. На экран выводится подсказка операционной системы. Если нужно сохранить текущую ситуацию обработки, перед вводом команды BYE следует вызвать команду SAVE.

Нажатие клавиши СНЯТИЕ (Esc) эквивалентно выполнению команды BYE.

3.4.4. Команда CONTINUE

Внутри команды WHILE команда CONTINUE приводит к тому, что пакет сразу же выполнит новое вычисление условий в операторе WHILE. В том случае, если значение логического выражения равно TRUE, система начнет новую итерацию последовательности командных операторов, которые следуют за словом DO. Если же значение логического выражения равно FALSE, то пакет прекратит обработку команды WHILE и приступит к обработке команды, которая следует непосредственно за фразой ENDWHILE.

Команда CONTINUE может быть использована вне области действия WHILE—ENDWHILE, но в области TEST—ENDTEST. В таком случае команда CONTINUE ведет себя точно так же, как и команда BREAK. В том случае, если команда CONTINUE используется в рамках команды TEST—ENDTEST, которая в свою очередь вложена в WHILE—ENDWHILE, пакет прекратит тестирование и перейдет к оператору, который следует за ENDTEST.

Пример.

```
WHILE I < 10 DO IF ARRAY (I) > 123 THEN I = I + 2; CON-
      TINUE;  ENDIF TOT = ARRAY (I) + TOT;  LET  I =
      = I + 1; ENDWHILE.
```

3.4.5. Команда IF...ENDIF

Синтаксис команды

IF {⟨условие⟩}* THEN {⟨оператор-1⟩}* [ELSE {⟨оператор-2⟩;}*]
ENDIF

Описание команды

Заданное условие представляет собой логическое выражение. Если значение этого выражения равно TRUE, то выполняются операторы, следующие за словом THEN; в противном случае выполняются операторы, которые стоят за фразой ELSE.

Каждый оператор перед ELSE и каждый оператор перед словосочетанием ENDIF должен заканчиваться точкой с запятой или символом конца строки. Фраза "IF ⟨условие⟩ THEN" должна располагаться в одной строке, но другие части этой команды могут быть расположены в различных строках без использования для этого символа обратной косой черты.

Команда IF позволяет осуществлять условное выполнение команд (или последовательности команд). Условия специфицируются в терминах логического выражения. Если значение выражения равно TRUE, то выполняется последовательность команд, которая следует за словом THEN до тех пор, пока не будет обнаружено слово ELSE или слово ENDIF; после этого будет выполняться команда, которая следует за словом ENDIF. Если значение логического выражения равно FALSE, то выполняется последовательность команд, которая следует за словом ELSE. В том случае, если необязательная фраза ELSE опущена, значение выражения FALSE приводит к тому, что система будет выполнять команду, которая следует непосредственно за словом ENDIF.

Любая из команд, которая следует за словом THEN или словом ELSE, сама может быть командой IF. Нужно помнить, что команда IF должна заканчиваться словом ENDIF. Такой тип вложенности может продолжаться на произвольную глубину. Любой из операторов команд может быть оператором WHILE, который должен заканчиваться словом ENDWHILE. Часто бывает удобным заменять последовательность команд в команде IF одной командой PERFORM.

3.4.6. Команда LET

Синтаксис команды

[LET] {⟨перем⟩ = ⟨выр-1⟩ [USING "⟨шаблон⟩"] | CELL ⟨яч⟩ =
= ⟨выр-2⟩}

С помощью комады LET можно указать то значение, которое нужно присвоить переменной. Присвоение может быть выполнено явно посредством спецификации константы или неявно — посредством спецификации выражения. При присвоении значения строковой переменной переменная E.LSTR не накладывает ограничений на длину строки. В строковой константе, присваиваемой переменной, может находиться до 255 символов. Переменной может быть присвоено строковое выражение, имеющее длину до 65 535 символов.

Переменная, которой присваивается значение, может быть рабочей переменной, переменной поля, переменной ячейки или предопределенной переменной (переменной описания среды или служебной переменной).

В случае переменной поля значение этого поля в текущей записи таблицы будет заменено значением, которое определяется значением выражения. Нельзя присвоить значение переменной виртуального поля.

При присвоении значения первому элементу массива можно опускать индекс(ы) массива. Например, A (1,1) = 3 эквивалентно A = 3, где A — массив. Аналогично можно опускать индекс(ы) при присвоении значения первого элемента массива переменной.

Примеры.

```
LET ФИО="ПЕТРОВ С. Д."
LET E.PMAR = 100
LET Z = ABS (A1 — 7)/18—5 * X / SQRT (A2)
```

Кроме присвоения значения переменной команда LET позволяет объявить для переменной шаблон. В том случае, если переменная уже имеет шаблон, указанный шаблон временно имеет высший приоритет, чем существующий шаблон этой переменной. Шаблон используется только во время данного конкретного присвоения значения этой переменной. Как только присвоено (или введено) другое значение этой переменной, шаблон, специфицированный во время предыдущего присвоения, становится недействительным.

В том случае, если за словом LET следует ключевое слово CELL, переменная должна быть переменной ячейки и указанное выражение становится новым определением для этой ячейки. При присвоении определения ячейке шаблон не должен быть специфицирован. В том случае, если ключевое слово CELL не используется с переменной ячейки, указанное выражение вычисляется и затем его значение присваивается ячейке.

Примеры.

```
LET #DATE = "110183" USING "DD/DD/DD"
LET #D = #C — #A
```

3.4.7. Команда LOAD

Синтаксис команды

```
LOAD {[ FROM ] "⟨файл .icf⟩" [ WITH "⟨режимы загрузки⟩" ] |
      FUNCTION "⟨файл⟩" | PERFORM "⟨файл .ipf⟩" | LIBRARY
      "⟨файл .ilf⟩"
```


Команда LOAD позволяет возобновить обработку ранее сохраненного контекста. Эта команда приводит к восстановлению определений всех ячеек, глобальных макросов, глобальных форм, значений переменных окружения и значений глобальных рабочих переменных, которые были ранее сохранены в указанном контекстном файле. В том случае, если контекст был сохранен с помощью команды \SAVE, на экране появляется электронная таблица в том виде, в каком она имела место непосредственно перед вводом команды \SAVE. Если же контекст был сохранен с помощью команды SAVE, то электронная таблица не появится и обработка может быть продолжена с того момента, который предшествовал (непосредственно) вводу команды SAVE.

Можно специфицировать один или более режимов загрузки одновременно с командой LOAD. В случае отсутствия спецификации режимов выполняется загрузка всех аспектов настоящего контекста, за исключением служебных переменных. При спецификации одного или более режимов загрузки выполняется загрузка только указанных аспектов контекста.

Команда LOAD PERFORM позволяет загрузить командный файл в память. При последующем выполнении операторов этого командного файла с помощью команды PERFORM обработка выполняется значительно быстрее, чем в том случае, когда командный файл предварительно не загружен в память. Таким образом, желательно загрузить часто используемые командные файлы в память.

Команда LOAD PERFORM загружает командный файл только в случае наличия достаточного количества памяти. Если памяти недостаточно, выдается диагностическое сообщение (его можно подавить, полагая E. SERR равным TRUE). В таком случае командный файл можно выполнить с помощью команды PERFORM обычным образом.

Команда RELEASE PERFORM позволяет освободить часть памяти, занятую загруженными командными файлами, команда SHOW PERFORM — увидеть, какие командные файлы находятся в настоящее время в памяти.

3.4.8. Команда PERFORM

Синтаксис команды

```
PERFORM "⟨файл .txt⟩" [ USING {⟨выр⟩} * ]
```

Описание команды

Использование команды PERFORM избавляет от необходимости повторного набора определений для каждого сеанса работы, в котором необходимо использовать эти определения.

Аналогично можно создать командный файл, образованный из установок окружения (среды). Каждая строка в этом файле может иметь команду LET, присваивающую нужное значение переменной определения среды. Всякий раз, когда файл выполняется, установки окружения корректируются. Если в командном файле переменная E.GUID равна TRUE, после выполнения всех команд файла появится главное меню пакета.

Использование аргумента ⟨файл .txt⟩ в командной строке, которая обеспечивает вызов пакета KMan, приводит к тому, что коман-

да PERFORM автоматически применяется к этому файлу, поэтому в данном случае ее не требуется явно использовать.

Команда PERFORM вызывает загрузку некоторого объема оперативной памяти. Этот локальный объем освобождается для других целей после завершения выполнения команды PERFORM.

Синонимом слова PERFORM является слово INCLUDE.

Для выполнения процедуры с параметрами нужно ввести

PERFORM "⟨имя файла⟩" USING ⟨список аргументов⟩

задавая полностью квалифицированное имя командного файла, содержащего ту процедуру, которую необходимо выполнить. Необязательная фраза "USING ⟨список аргументов⟩" указывает на определения, которые должны использоваться для параметров, появляющихся в процедуре.

Пакет может выполнять параметризованные процедуры. В последовательности команд, которые составляют тело процедуры, может быть использовано до 26 различных параметров. В рамках процедуры эти параметры обозначаются таким образом: #A, #B, #C, . . . , #Z. Параметр является локальным для данной процедуры. Когда процедура выполняется, ее параметры автоматически заменяются значениями аргументов команды PERFORM.

Каждый аргумент, специфицированный в команде PERFORM, может быть строковым, числовым или логическим выражением. Аргументы отделяются друг от друга запятыми. Значение первого аргумента становится определением параметра #A процедуры, значение второго аргумента — определением параметра #B и т. д. В случае, если количество специфицированных аргументов превышает количество параметров, используемых в процедуре, лишние параметры игнорируются; в случае, если специфицировано слишком мало параметров, неприсвоенные параметры остаются неопределенными. Слово USING является необязательным.

В том случае, если требуется неявная команда PERFORM (вызванная с помощью включения имени файла при вызове пакета), в процедуре не могут содержаться параметры.

3.4.9. Команда RELEASE

Синтаксис команды

RELEASE ⟨макро⟩ | ⟨форма⟩ | ⟨перем⟩ | ALL | LOCAL | FORM | VARIABLE | ARRAY | FUNCTION {⟨файл⟩ | ALL } | PERFORM {"файл .ipf⟩" | ALL} | LIBRARY "⟨файл .ipf⟩"

Описание команды

Команда RELEASE при ее использовании с параметром ALL приводит к тому, что система "забудет" обо всех формах, макросах и рабочих переменных. Это вызовет освобождение части оперативной памяти для других целей. Команда RELEASE вначале освобождает память, занятую локальными объявлениями, а затем память, занятую глобальными объявлениями. Команда RELEASE не оказывает воздействия на переменные поля, окружения и служебные переменные.

Кроме того, можно освободить память локальных переменных, макросов, форм, массивов и немассивов.

Далее, можно освободить память, занимаемую специфицированными рабочими переменными, формами или макросами. Для этого следует указать имя переменной, формы или макроса. Указанные переменные, форма или макрос исчезнут.

3.4.10. Команда RETURN

С помощью этой команды закрывается командный файл, содержащий процедуру, в которой было обнаружено слово RETURN. Управление возвращается той команде, которая следует непосредственно после команды PERFORM, вызвавшей данную процедуру.

В рамках процедуры команда RETURN используется с той целью, чтобы указать ту точку (точки), в которой система должна выйти из процедуры. Когда система встречает в процедуре слово RETURN, она закрывает командный файл процедуры. Все локальные переменные, макросы и формы, определенные в данной процедуре, пропадают. Вся локальная память, которая была использована в процессе выполнения процедуры, освобождается для других целей. После этого пакет возвращается к той команде PERFORM, которая осуществила вызов данной процедуры, и начинает обработку следующей команды PERFORM. Если такой команды нет и E. GUID равно FALSE, появляется стандартная подсказка KMan. Если же E.GUID равно TRUE, появляется основное меню.

3.4.11. Команда ROOT

Синтаксис команды

ROOT ⟨п_рабоч⟩, ⟨ч_выр-1⟩, ⟨ч_выр-2⟩, ⟨ч_выр-3⟩, ⟨ч_выр-4⟩

Описание команды

Вычисляется корень заданного выражения для заданной переменной. Это вычисление выполняется в диапазоне, заданном начальным и конечным значениями с указанным отклонением. Результирующий корень присваивается переменной.

Переменная должна быть скалярной рабочей переменной. Выражение должно быть задано в терминах этой переменной и может включать другие переменные. Это выражение рассматривается как функция от заданной переменной и приравнивается к нулю (т. е. F (переменная) = 0, где F должна быть непрерывной, но не обязательно дифференцируемой функцией). Затем результирующее уравнение решается относительно заданной переменной.

Поскольку уравнение может иметь несколько корней, необходимо задать верхнюю и нижнюю границы для нужного корня. Необходимо также задать уровень отклонения. Гарантируется, что значение, вычисленное с помощью команды ROOT, будет находиться в пределах действительного корня плюс/минус уровень отклонения. Если задано отклонение 0, предполагается .001.

Пример.

ROOT X, 0, 10, 0001, X * X + 5 * X — 24

присваивает значение 3 переменной X.

3.4.12. Команда SAVE

Синтаксис команды

SAVE [TO] "⟨файл .icf⟩" [WITH "⟨режимы сохранения⟩"]

Команда SAVE позволяет временно прекратить сеанс работы с системой, а позднее возобновить сеанс с этой же точки (см. команду LOAD). Команда SAVE приводит к тому, что все текущие определения ячеек электронной таблицы, глобальных макросов, глобальных форм, значения переменных окружения и переменных утилит будут сохранены в указанном контекстном файле. Локальные определения не сохраняются.

В команде SAVE можно задать один или более следующих режимов:

W — рабочие переменные (скалярные)
A — рабочие переменные (массивы)
M — макросы
F — формы
E — переменные окружения
C — определение ячеек
U — служебные переменные

Если не задан ни один код режима, сохраняются все, кроме значений служебных переменных.

3.4.13. Команда STOP

Команда используется в процедурах для указания точки (точек), в которых пакет должен осуществить выход из всех процедур, командные файлы которых в текущий момент открыты. При обнаружении в процедуре команды STOP пакет закрывает командный файл данной процедуры и все другие командные файлы, которые открыты в данный момент. Таким образом, в случае вложенных процедур команда STOP ведет себя подобно последовательности операторов RETURN, которая выполняется начиная с процедуры самого нижнего уровня (т. е. начиная с командного файла, открытого самым последним) и заканчивая процедурой самого высокого уровня. После этого пакет будет обрабатывать команды, которые следуют за той командой PERFORM, которая инициировала этот процесс для процедуры самого высокого уровня. Если таких команд нет и E.GUID равно TRUE, то на экране возникнет подсказка пакета. В том случае, если команда STOP вызывается не во вложенной процедуре, она работает точно так же, как и команда RETURN.

Нажатие клавиши СНЯТИЕ во время выполнения процедуры аналогично обнаружению (в теле процедуры) команды STOP, кроме случая, когда она нажимается во время выполнения команды GETFORM (при этом команда GETFORM прерывается и выполняется следующая команда процедуры).

3.4.14. Команда TEST...ENDTEST

Синтаксис команды

```
TEST ⟨выр-1⟩CASE⟨выр-2⟩ | {⟨оператор-1⟩}* [OTHERWISE |
     {⟨оператор-2⟩}* ]
     ENDTEST
```


В том случае, если значение тестируемого выражения совпадает со значением ⟨выражения-1⟩, будут выполняться команды ⟨операторов-1⟩, ⟨операторов-2⟩, . . . , ⟨операторов-N⟩, до тех пор пока не будет обнаружена команда BREAK. При обнаружении команды BREAK система переходит к обработке команды, следующей после ENDTEST. В том случае, если значения тестируемого выражения и ⟨выражения-1⟩ не совпадают, пакет проверяет ⟨выражение-2⟩. Если значения равны, то выполняются ⟨операторы-2⟩ и последующие операторы, до тех пор пока не будет обнаружена команда BREAK, и т. д. Каждый оператор должен заканчиваться точкой с запятой или символом конца строки.

В том случае, если не удовлетворяется ни один из явно специфицированных случаев, обработка продолжается с команды, которая следует непосредственно за командой ENDTEST. Существует одно исключение. В том случае, если специфицирован оператор OTHERWISE (этот оператор необязателен), перед выходом из команды TEST выполняются операторы, стоящие за фразой OTHERWISE.

Рекомендуется формат команды TEST, представленный выше, однако допустимы и другие форматы. Несколько из допустимых форматов показаны в примерах. В общем случае выражение, которое следует за словом TEST, должно находиться в той же строке, что и слово TEST, и должно заканчиваться точкой с запятой или символом конца строки. Кроме того, каждый оператор должен заканчиваться либо точкой с запятой, либо символом конца строки.

Примеры.

```
TEST TRUE
  CASE X LE 100; PERFORM "PROGR3" USING "X";
         Z = TRUE; BREAK;
  CASE X GT 100 & X LT 250; LET XY = 1.08*XY;
  BREAK; OTHERWISE: PERFORM "PROG5" USING "X—
  250"; ENDTEST TEST X
    CASE 1.0: BREAK
    CASE 3.0: OUTPUT X
ENDTEST
```

3.4.15. Команда WAIT

Посредством этой команды пакет останавливает обработку, до тех пор пока пользователь не нажмет любую клавишу на клавиатуре (кроме ВЫХОД и СНЯТИЕ).

3.4.16. Команда WHILE...ENDWHILE

Синтаксис команды

```
WHILE {⟨условие⟩} DO {⟨оператор⟩;} * ENWHILE
```

Описание команды

Если значение параметра ⟨условие⟩ равно TRUE, будет выполняться обработка указанных командных операторов. Каждый из этих операторов должен заключаться точкой с запятой (;) или кон-

цом строки. В конце каждой итерации выполнения этих операторов осуществляется повторная оценка условий. Если в результате получается значение FALSE, то система переходит к следующему оператору после оператора ENDWHILE. Фраза "WHILE ⟨условия⟩ DO" должна располагаться в одной строке. Нужно помнить, что для продолжения длинной строки можно использовать символ обратной косой черты.

Команда WHILE не может быть разделена между процедурами. Если команда WHILE возникла в процедуре, которую необходимо выполнить, то согласующая фраза ENDWHILE должна также находиться в этой процедуре.

Во время первой итерации команды WHILE каждая ссылка на макрос заменяется соответствующим текстом макроса.

Примеры.

```
WHILE I < 10 DO LET TOT = ARRAY (I) + TOT; LET I = I +1;
ENDWHILE
WHILE I < 10 DO
      LET ARRY (I, 1) = I * 5
      LET I = I + 1
ENDWHILE

WHILE RESPONSE= "YES" DO TALLY ENTRYFORM; GET-
      FORM ENTRYFORM; RESET ENTRYFORM; END-
      WHILE

WHILE STATUS=TRUE & VAR3>13 DO PERFORM "STATCHK"
USING "13"; ENDWHILE
```

Глава 4

ФУНКЦИИ

4.1. Общие положения

Пакет KMan предоставляет пользователю широкий набор стандартных функций. Каждая функция может иметь один или несколько параметров (аргументов), являющихся входными данными.

Ниже приведено подробное описание всех стандартных функций KMan. При описании параметров функций используются следующие условные сокращения:

ч. в. — числовое выражение
с. в. — строковое выражение
л. в. — логическое выражение
н. п. — начальная позиция (положительное целое число, указывающее позицию в строке, с которой должна начаться обработка)
дес. — положительное целое число, задающее количество цифр справа от десятичной точки

4.2. Числовые функции

Таблица 4.1

Функция	Назначение	Результат использования
ABS (ч. в.)	Вычисляет выражение и затем берет его абсолютное значение	ABS (7·3 — 9·5) = 2·2
ARCSIN (ч. в.)	Вычисляет выражение и берет арксинус полученного количества радиан	ARCSIN (.85) = 1.02
CURREC (таблица)	Возвращает номер обрабатываемой последней (текущей) записи в указанной таблице БД	CURREC (EMPLOYEE)
DRAIN ( )	Очищает все коды нажатий клавиш, которые находятся в буфере нажатий, и возвращает 0	DRAIN ( ) = 0
EXP (ч. в.)	Возводит число E (математическая константа, равная 2,71828 . . .) в степень, определяемую числовым выражением	EXP (1) = 2.71828
INIT (массив, ч. в.)	Присваивает каждому элементу массива значение, определяемое выражением. Возвращает значение этого выражения	INIT (MAS, 8**2) = 64
LASTREC (таблица)	Возвращает номер последней записи в указанной таблице БД (т. е. количество записей)	LASTREC (A) = 0, если таблица A пуста
LEN (с. в.)	Определяет количество символов в строковом выражении (т. е. длину строки)	LEN ("ABC" + "DEF") = = 6
LN (ч. в.)	Вычисляет натуральный логарифм аргумента	LN (8.2.*1.54) = 2,37
LOG (ч. в.)	Вычисляет логарифм по основанию 10 аргумента	LOG (8.2*25.45+26.87) = = 2.37
MATCH (с. в.1, с.в.2)	Определяет, начиная с какой позиции вторая строка расположена в первой Если вторая строка в первой отсутствует, функция возвращает 0. Если в качестве второй строки задана пустая строка (" "), то функция возвращает 1	MATCH ("ABCDEFGH", "EF") = 5
MAX (ч. в. 1, ч. в. 2)	Вычисляет два выражения и возвращает значение большего из них	MAX (5*4, 3*2) = 20

Продолжение табл. 4.1

Функция	Назначение	Результат использования
MEM ( )	Возвращает размер оставшейся свободной памяти (количество свободных байт). Если KMan необходима свободная память, он пытается сначала использовать память, полученную ранее. Если свободного пространства в распределенной части области данных недостаточно, то для использования выдается пространство, освобожденное ранее	
MIN (ч. в. 1, ч. в. 2)	Вычисляет два выражения и возвращает значение меньшего из них	MIN (10*2,100/4) = 20
NOABORT (л. в.)	Вычисляет логическое выражение и запрещает использование клавиши ESC для снятия команд (за исключением CREATE, BROWSE и обработки форм), если аргумент равен ИСТИНА (TRUE). Если аргумент равен ЛОЖЬ (FALSE), то клавиша ESC восстанавливается и обрабатывается обычным сбразом. Функция возвращает количество нажатий на клавишу ESC с момента предыдущего обращения к функции NOABORT	
NOEXIT (л. в.)	Если значением аргумента является ИСТИНА (TRUE), клавиша ВВОД не оказывает обычного действия в пакете KMan. Если аргумент — ЛОЖЬ (FALSE), то реакция на клавишу ВВОД восстанавливается. Функция NOEXIT возвращает количество нажатий на клавишу ВВОД с момента предыдущего вызова этой функции	
RAND (ч. в.)	Вычисляет выражение, используя его как начальное число для генерации случайного числа. Результатом функции является случайное число от 0 до 1. Аргумент,	RAND (X*2) = 0.4

Продолжение табл. 4.1

Функция	Назначение	Результат использования
	отличный от 0, генерирует новую последовательность случайных чисел и возвращает первое из них как результат. Вызов функции RAND с нулевым аргументом выдает следующее случайное число из последовательности	
SIN (ч. в.)	Вычисляет синус количества радиан, задаваемых значением аргумента	SIN (0.66) = 0.61
SQRT (ч. в.)	Возвращает значение квадратного корня аргумента, который должен быть положительным числом	SQRT (1 + 1.25) = 1.5
TOJUL (с. в)	Аргумент определяет дату. Функция вычисляет значение, равное количеству дней, прошедших до этой даты с момента принятия в Европе Юлианского календаря (15 октября 1582 г.).	(Если E.DTYP = "M") TOJUL ("4/31/87") = 0 TOJUL ("1/1/2001") = = 152750
	Любой нечисловой символ в дате воспринимается как разделитель. Формат задания даты определяется по текущему значению переменной E. DTYP. День и месяц могут быть заданы одной или двумя цифрами. Год — двумя, тремя или четырьмя (при задании двух цифр используется приставка «19», трех — «1»). Если аргумент задан некорректно, функция возвращает 0	TOJUL ("3:7:86") — — TOJUL ("3:21:86") = 14 TOJUL ("abcd") = 0
TONUM (с. в.)	Преобразует строку в число. Преобразование строки прекращается на первом символе строки, отличном от цифры или десятичной точки. Знак минус допускается только в первой позиции, иначе он тоже останавливает преобразование	TONUM ("12345") = 12345 TONUM ("A420") = 0 TONUM ("789*02") = = 789 TONUM ("— 123.45") = = — 123.45

Продолжение табл. 4.1

Функция	Назначение	Результат использования
TRUNC (ч. в)	Вычисляет выражение и отсекает все цифры после десятичной точки. Результат — целое число	TRUNC (5*3.3) = 16
VAL (с. в.)	Преобразует первый символ строки в число, соответствующее коду ASCII (КОИ-7) этого символа. Если первым в строке расположен непечатный символ, то функция возвращает число в диапазоне 128 — 255, зависимое от действующей на ПЭВМ кодовой таблицы. Если строка пуста, результат равен нулю	VAL ("ABC") = 97 VAL ("132") = 49 VAL (" ") = 0

4.3. Строковые функции

Таблица 4.2

Функция	Назначение	Результат использования
CHR (ч. в.)	Преобразует код ASCII, соответствующий числовому выражению, в его символьный эквивалент. Если значение аргумента находится в пределах 128 — 255, результатом будет символ, зависящий от аппаратуры. Эта функция может быть использована при посылке управляющих кодов на принтер (см. команду PRINT)	CHR (119) = "W" CHR (35) = "5"
CHSTR (с. в. 1, с. в. 2, с. в. 3)	Вычисляет все три выражения, а затем сканирует первую строку, пытаясь обнаружить в ней вхождения второй строки и заменить каждое из них значением третьей строки. Результатом функции будет измененный вариант первой строки.	CHSTR ("Изменить строк", "к", "ку") = "Изменить строку" CHSTR ("18.04.1987", "19", " ") = "18.04.87"

Продолжение табл. 4.2

Функция	Назначение	Результат использования
	Если вторая строка в первой не существует, функция возвращает неизмененную первую строку	
INIT (массив, с. в.)	Вычисляет значение строкового выражения и присваивает каждому элементу массива. Возвращает значение строкового выражения	INIT (A2, "A" + "B") = = "AB"
LABEL (переменная)	Возвращает метку переменной — поля таблицы БД. Если это поле не имеет метки, результат — пустая строка	LABEL (ФИО) = "Фамилия, И, О"
LOCASE (с. в.)	Переводит каждый символ значения аргумента в нижний регистр	LOCASE ("aBCpEo" = = "абсрео"
SUBSTR (с. в., н. п., длина)	Выделяет подстроку заданной длины из строкового выражения начиная с символа, заданного вторым аргументом. Выделенная подстрока является результатом функции	SUBSTR ("ABCDE", 2, 3)= = "BCD" SUBSTR (S, 1, 6) + + SUBSTR (S, 9, 2) = ="11.05.87", если S = "11.05.87"
TCDATE (таблица)	Возвращает дату создания заданной таблицы БД в формате, определенном значением E.DTYR	
TIME ()	Возвращает текущее время в виде строки следующего формата: «чч:мм:сс». Если ПЭВМ не имеет системных часов, результат не определен	
TMDATE (таблица)	Возвращает дату последней модификации заданной таблицы БД в формате, определенном значением E.DTYP	
TODATE (ч. в. 1, с. в., ч. в. 2)	Генерирует дату (как строку в формате, определенном текущим значением E.DTYP), которая отстоит от даты введения в Европе Юлианского календаря (15 октября 1582 г.) на число дней, заданное первым параметром. Второй аргумент должен быть одиночным символом; он используется как разделитель при выводе даты. Третий аргумент	(При E.DTYP = "M") TODATE (114458, ".", 4)= = "02.29.86" TODATE (134986,"/",3) = = "05/14/952" TODATE (A, "*", 2)=" "

Продолжение табл. 4.2

Функция	Назначение	Результат использования
	определяет, сколько цифр отведено для вывода года. Если ч. в. 1 — не целое, оно округляется. Если первый аргумент задан некорректно, генерируется пустая строка	TODATE (TOJUL ("2/9 — 87"), " ", 2) = = "02 09 87"
TOSTR (ч.в., длина, дес.)	Вычисляет числовое выражение и преобразует его в строку заданной длины с указанным количеством цифр после десятичной точки (для этого может происходить округление числа). Если результат не помещается в строку заданной длины, выдается сообщение об ошибке	TOSTR (34.5678, 5,2) = = "34.56" TOSTR (34.56,2,0) = "34" TOSTR (1234.5678,5,2) выдаст сообщение об ошибке
TRIM (с. в.)	Вычисляет выражение и убирает все хвостовые пробелы. Результат возвращается как значение функции	TRIM ("Петр" + + "Иванов") = "Петр Иванов"
TYPE (переменная)	Определяет и выдает тип указанной переменной («STR», «NUM» или «LOGIC»)	TYPE (ФИО) = "STR"
UPCASE (с. в.)	Переводит все символы значения строкового выражения в верхний регистр и возвращает полученную строку как результат	UPCASE ("aBcpE") = = "ABCPE"

4.4. Логические функции

Таблица 4.3

Функция	Назначение	Результат использования
ALPHASTR (с. в.)	Возвращает значение TRUE, если заданная строка содержит только алфавитные символы; иначе возвращается FALSE	ALPHASTR ("abcd") = = TRUE ALPHASTR ("ab1d") = = FALSE
EOT (таблица)	Возвращает TRUE, если при обработке указанной таблицы был достигнут ее физический конец (последняя запись)	

Продолжение табл. 4.3

Функция	Назначение	Результат использования
FILEX (с.в.)	Возвращает TRUE, если заданный аргументом файл существует (имя файла может содержать имя дисковода, иначе поиск осуществляется на активном)	FILEX ("b:/exe/pr.1") =TRUE, если файл pr. 1 существует на диске B: в каталоге/exe
INIT (массив, л. в.)	Вычисляет логическое выражение и присваивает его значение каждому элементу массива. Функция возвращает значение выражения	INIT (mas1, 5 > 2) = = TRUE
INUSE (таблица)	Возвращает TRUE, если заданная таблица БД открыта (используется) в настоящий момент времени	
ISALPHA (с. в.)	Возвращает значение TRUE, если первый символ результата заданного выражения является буквой	ISALPHA ("abcd") = = TRUE ISALPHA (".75ab") = = FALSE
ISDIGIT (с. в.)	Возвращает значение TRUE, если первый символ результата заданного выражения является цифрой	ISDIGIT ("abcd") = = FALSE ISDIGIT (".75ab") = = TRUE
NUMSTR (с. в.)	Возвращает значение TRUE, если заданная строка содержит только цифры; иначе возвращается FALSE	NUMSTR ("93a6, d") = = FALSE NUMSTR ("123456") = = TRUE NUMSTR ("93a6, d")
PASTEND (таблица)	Возвращает значение TRUE, если последняя из команд OBTAIN или PLUCK пыталась получить запись после конца указанной таблицы; иначе (если запись была найдена внутри таблицы) возвращается FALSE. Результат корректен только в том случае, если OBTAIN или PLUCK была последней из команд, выданной для получения записи из таблицы	

Часть III

РАЗВИВАЮЩАЯСЯ ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ СИСТЕМА SMALLTALK-80

Глава 1

ОСНОВНЫЕ ПОЛОЖЕНИЯ

1.1. Общая характеристика

Система Smalltalk-80 (в дальнейшем сокращенно ST) представляет собой совокупность новых взглядов, методов и средств проектирования и эксплуатации современных систем обработки информации, базирующихся на ряде новых концепций в понимании и модельном представлении вычислительного процесса (точнее, процесса автоматической переработки данных). Авторами ST являются сотрудники исследовательской группы фирмы XEROX (США) (Software Concept Group), работающие над этим проектом с 1970 г. [10].

Авторы ST рассматривают свой проект в четырех следующих аспектах:

ST является новым взглядом на эффективное взаимодействие человека и компьютерных систем;

ST базируется на минимальном числе концепций, однако выраженных необычной терминологией;

ST представляет пользователю интерактивную графическую программную среду;

ST является большой системой.

С первым аспектом (мировоззренческим) связано новое понимание информационных коммуникаций человека и вычислительной среды, включающее вопросы хранения, переработки и обмена информации в ЭВМ. В отличие от традиционного понимания и реализации процесса обработки информации, при котором ведущими элементами процесса являются фиксированная конфигурация процессоров, выполняющих заданные программы, и соответствующая конфигурация памяти для обрабатываемых данных, ST-машина динамически организует произвольные конфигурации процессоров, обрабатывающих данные в ассоциативной среде памяти.

Второй аспект (аспект формализации) заключается в выборе средств формального языкового представления ST-машины. Авторы ST, исходя из ряда концепций языков моделирования типа СИМУЛА-67 [11] в отношении классов и динамически порождаемых экземпляров классов, объединили новейшие концепции абстракции типов данных (языки CLU, ADA, МОДУЛА-2 [12—14]) и так называемый объектно-ориентированный подход (ООП) [15] и выработали четкое и лаконичное выражение синтаксиса и семантики ST. С языковых позиций ST представляет собой расширяющийся объектно-ориентированный язык, который путем задания новых классов объектов создает произвольную вычислительную среду, ориентированную на данный класс приложений.

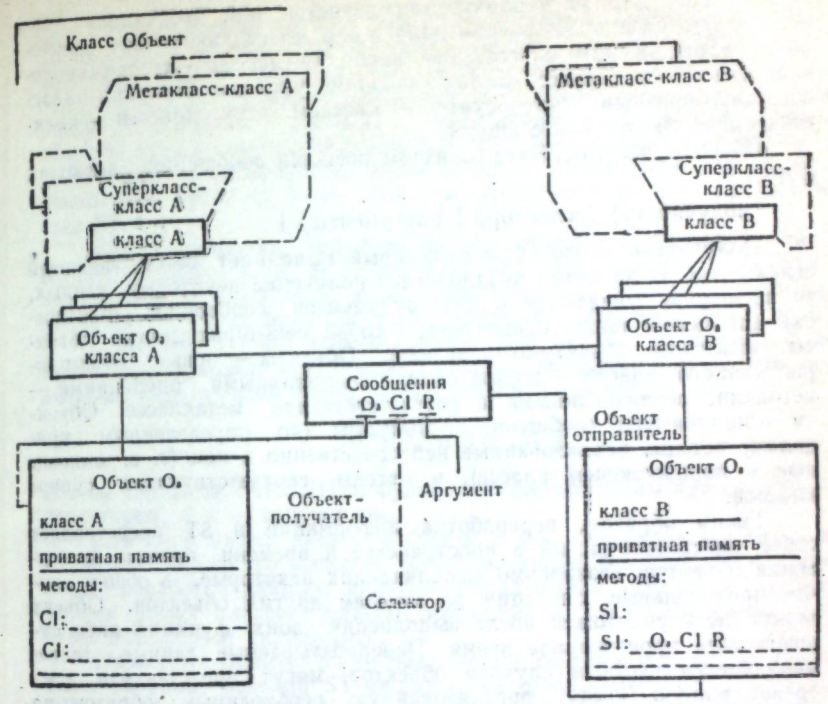


Рис. 6. Основные понятия ST.

Третий аспект (интерфейсный) определяет реальный уровень взаимодействия пользователей (профессиональных и непрофессиональных) с ST-машиной, пока реализованной на мини/микро-ЭВМ традиционной неймановской структуры. С этим аспектом (по существу, не главным) связано большинство ссылок на ST [16]. Диалог пользователя и ST-машины осуществляется в идеологии многооконного взаимодействия, при котором технология активизации окон системного браузера, рабочего поля, окон меню соответствует иерархии мышления человека, иерархии процессов настройки ST-машины на конкретные приложения и собственно эксплуатационные сеансы.

Последний, четвертый, аспект (системотехнический) заключается в реалистическом взгляде на многие существенные достоинства ST, которые достигаются путем весьма сложной программной реализации системы с вытекающими отсюда ресурсными проблемами.

На рис. 6 изображены первичные понятия ST. Объекты являются однородными экземплярами некоторого класса. Объект можно представить некоторой машиной, способной выполнять заданный набор операций, называемых методами, и имеющей некоторую приватную память. Одновременно может существовать произвольное количество классов, связанных в некоторую двойную иерархию подчинения. Первая иерархия классов использует терминологию класс — суперкласс, т. е. данному классу в принципе доступны операции

суперкласса. Вторая иерархия определяется тем, что единая концепция объектов в ST рассматривает сам класс как объект некоторого порождающего класса. Эта иерархия использует терминологию класс — метакласс. Самым внешним уровнем иерархии классов, своеобразным универсумом — классом всех классов — является специальный класс Object.

Объекты взаимодействуют путем посылки сообщений, имеющих форму

⟨получатель⟩ ⟨селектор⟩ [⟨аргументы⟩]

Получатель — это объект, который выполняет метод, заданный селектором. Если метод предполагает получение некоторых данных, то последние передаются в виде аргументов сообщения. Фактически аргументы также представляют собой некоторые другие объекты, известные объекту-отправителю. Объекты — новые экземпляры данного класса — порождаются специальными операциями — методами, реализованными в соответствующем метаклассе. Объекту — получателю сообщения — доступны (по определенным правилам) методы, реализованные непосредственно в нем (т. е. описанные в порождающем классе), и методы соответствующих суперклассов.

Таким образом, переработка информации в ST представляет собой рассредоточенный в пространстве и времени процесс порождения объектов, независимо выполняющих некоторые, в общем случае произвольные, операции по заказам других объектов. Объект может быть уничтожен после выполнения своих функций либо существовать произвольное время. Перерабатываемые данные, также являющиеся частным случаем объектов, могут представлять рассредоточенную среду, организованную необходимым образом на требуемое время обработки ассоциативными ссылками.

Языковые средства ST позволяют при задании метода определять произвольные алгоритмы, поэтому в ST практически естественным образом могут быть представлены все задачи, программируемые на традиционных языках программирования.

Необычность терминологии и соответствующей нотации ST можно проиллюстрировать следующим образом.

С точки зрения математики процесс обработки информации на ЭВМ задается некоторым оператором

$$X = f(Y, Z)$$

где f — вид осуществляемого преобразования, Y, Z — исходные данные, X — результат.

В терминологии ST этот оператор рассматривается связанным с некоторым выполняющим его объектом, в качестве которого может быть выбран параметр Y. Тогда выполнение оператора осуществляется путем посылки сообщения с селектором f объекту Y с аргументом Z:

Y f Z

Результат выполнения оператора f над Y и Z представляет собой также некоторый объект, и если необходимо его сохранить с некоторым именем, то запись в ST будет следующей:

X <— Y f Z

Отметим, что эта нотация вполне естественна для традиционных выражений типа

a <— x + 1 ('a' присвоить 'x + 1')

где x — объект (числовой), в котором реализован метод сложения чисел с обозначением этой операции знаком "+" (селектор), а 1 — числовой объект — константа единица — является аргументом сообщения x + 1. Результат операции, т. е. сумма значения x с единицей, связывается с именем a, относящимся к приватной памяти объекта, посылающего сообщение о выполнении операции сложения. Хотя подобная интерпретация для данного случая выглядит усложненно, она имеет глубокие корни и определяет многие преимущества ST.

Более необычной является запись знакомых функций типа sin (x), sqrt (x) и т. п. В нотации ST они записываются как сообщения вида x sin, x sqrt, т. е. объектом, выполняющим операции, является параметр функции x, а аргумент сообщения отсутствует.

Естественно, в ST можно сохранить и для рассмотренных элементарных функций привычную нотацию, например определить некоторый специальный класс, реализующий их вычисление:

ВЫЧИСЛИТЬ sin x

В этом случае ВЫЧИСЛИТЬ будет экземпляром этого класса, sin — селектором, а x — аргументом.

Для того чтобы начать работать с ST, необходимо некоторое начальное множество классов, обеспечивающих базовый набор операций обработки данных, включая манипуляции, связанные с созданием новых классов. Классы, входящие в это начальное множество, называются системными.

Известные практические реализации системных классов (другими словами, базовой ST-машины) на однопроцессорных микроЭВМ, требующие 500 Кбайт оперативной памяти, уже сейчас обеспечивают богатые возможности обработки данных как в традиционных приложениях, так и в режиме «самораскрутки» (путем создания новых классов в имеющейся иерархии базовой машины) для широкого спектра прикладных систем, ориентированных на круг пользователей от домохозяек, использующих подобные настроенные персональные ЭВМ для учета домашних расходов, до профессионалов, создающих сложные системы для моделирования, решения задач искусственного интеллекта и т. п.

Общая технология работы с базовой ST-машиной заключается в следующем. Пользователю первоначально представляются на экране дисплея два перекрывающихся окна: окно так называемого системного браузера, через которое пользователь создает новые классы, и окно рабочего поля, через которое производится непосредственная посылка начального сообщения, вызывающего требуемые действия по обработке информации.

Активизация окна, т. е. выдвижение видимой в нем информации на передний план экрана, и разрешение на последующее взаимодействие с ним (ввод информации с клавиатуры, сдвиги информации в окне и т. п.) производятся быстрой дистанционной установкой специального указателя (курсора) на поле требуемого окна. Активизация требуемого окна фактически задает последующие режимы работы, при этом возможные дальнейшие технологические действия визуализируются в поле активного окна некоторым иерархическим образом по типу меню. Если выбранное из предлагаемых в данный момент действий требует дальнейшей детализации, то соответствующая подсказывающая информация появляется в новом окне, размещенном в поле активного окна в виде, не мешающем осуществлять необходимые основные действия. При этом на экране дисплея оста-

ются видимые части необходимых пассивных окон, которые можно активизировать соответствующей установкой указателя.

Технология подобной диалоговой работы в целом может быть использована (и в определенной степени используется) и в традиционной программной среде, однако в ST-машине она позволяет производить настройку на новые приложения с производительностью, на порядки превышающей производительность подобных работ с традиционными пакетами программ, трансляторами и т.п., за счет единой технологии и инструментария объединения и расширения разнородных объектов. Так, например, даже современный интегрированный пакет FrameWork-II [9], в котором произведена удачная попытка объединения разнородных информационных объектов, обрабатываемых системами управления базами данных, текстовыми редакторами, процессорами электронных таблиц и т. п., обладает существенными ограничениями концептуального характера. Отсутствие единого формального аппарата для операций над разнородными структурами данных не позволяет, в частности, создание новых типов фреймов.

По поводу графического интерфейса с пользователем необходимо отметить также наличие продуманной и эстетичной иерархии системных классов, обеспечивающих экранную графику. ST позволяет удобно выводить на экран различные графические объекты (диаграммы, таблицы, рисунки и др.) в цветном или многотоновом растре, включая и динамические объекты типа мультипликаций.

Более глубокий анализ ST показывает, что это тщательно продуманная фундаментальная разработка, не имеющая прямых аналогов в традиционной практике производства программной продукции, поскольку охватывает на единой концептуальной основе все известные программно-аппаратные уровни виртуальной машины пользователя. При этом микропроцессорная (аппаратная) реализация основных системных классов может не только значительно опередить современный уровень развития ЭВМ, но и обеспечить эффективную реализацию 5-го и, возможно, дальнейших поколений ЭВМ.

1.2. Описание реализации класса

Основной программной единицей ST является описание класса. Существуют два понятия описания класса: описание реализации и описание протоколов. В описании реализации непосредственно специфицируются алгоритмы, реализующие методы данного класса, указываются приватная память и взаимосвязь с существующими классами. Описание протоколов представляет собой спецификацию вида (шаблона) сообщения к данному классу. Другими словами, протоколы поясняют пользователю «что?», а реализация — «как?», т. е. реализация адресована системе и проходит этапы трансляции, после чего ST-машина способна удовлетворять запросы пользователя в соответствии с протоколами.

Реализуемые методы можно разделить на два вида: методы класса и методы экземпляров класса. Методы класса выполняются в метаклассе и связаны с операциями над классами как объектами (например, создание нового экземпляра (объекта) данного класса). Методы экземпляров класса выполняются самими объектами — экземплярами класса — независимо друг от друга. В связи с этим различаются также протоколы: протоколы класса (методы класса) и экземплярные протоколы (методы экземпляров класса). Кроме этого в практических целях методы группируются в так называемые

категории и соответственно различаются категории классов и категории сообщений.

При описании реализации, таким образом, существует следующая иерархия: совокупность категорий классов (ST-машина); классы данной категории классов; совокупность категорий сообщений (методов) данного класса; методы данной категории сообщений; описание реализации данного метода, имеющего структуру,— образец сообщения, реализующий алгоритм.

Эта иерархия соответствует диалоговому режиму ввода частей описания реализации класса в режиме системного браузера.

В целом структура описания реализации класса следующая:

```
class name                  — имя класса
superclass                  — имя суперкласса
[instance variable names    — список имен приватных экземплярных переменных]
[indexed instance variables — указание на индексные переменные]
[class variable names       — список имен общих переменных класса]
[shared pools               — список имен переменных общего пула нескольких классов ]

class methods:
        ⟨список категорий сообщений класса⟩
instance methods:
        ⟨список категорий сообщений экземпляра⟩
    EOD "конец описания"
```

Синтаксически методы класса и методы экземпляров (сообщения) одинаковы. Каждая категория имеет структуру

⟨название категории класса (или) экземпляра⟩
⟨список описаний методов⟩

Ниже будет введен строгий синтаксис метода и приведены примеры (см. п. 1.4 настоящей главы).

1.3. Описание протоколов

Как указывалось выше, описание протоколов (класса и экземпляра) ориентировано на пользователя и имеет следующую структуру:

⟨имя класса⟩ class protocol
⟨наименование категорий класса⟩
⟨образец сообщения⟩ ⟨комментарий, поясняющий семантику сообщения метаклассу⟩
⟨имя класса⟩ instance protocol
⟨наименование категорий экземпляра⟩
⟨образец сообщения⟩ ⟨комментарий, поясняющий семантику сообщения экземпляру класса⟩

Примеры (см. п. 1.4 настоящей главы).

1.4. Синтаксис описания метода

Диалоговый режим ввода описания реализации класса позволяет последовательно вводить основные элементы структуры описания, большинство которых являются тривиальными с точки

врения синтаксического анализа (т. е. ключевые слова class, variables . . . и т. д. и идентификаторы — имена). Относительно сложную синтаксическую структуру имеет метод (т. е. описание его реализации) внутри категории сообщения.

С целью упрощения описания синтаксиса и в связи с ориентированностью описания на реализацию соответствующих анализаторов приведем три группы синтаксических определений в БНФ. В первой группе определены основные понятия, а некоторые автоматные подграфы грамматики оставлены как терминальные множества. В последующих группах синтаксис детализируется до алфавита (т. е. собственно терминального словаря).

Ниже используются следующие обозначения:

EOM — символ окончания описания метода
ENTER — символ «конец ввода»
SHIFT — символ вертикальной стрелки
TAB — символ табуляции
VL — символ вертикальной черты
SPACE — символ пробела

В качестве метасимвола «ИЛИ» в синтаксических правилах используется символ«/».

1.1. ⟨метод⟩ : : = ⟨заголовок⟩ ⟨операторы⟩ EOM
1.2. ⟨операторы⟩ : : = ⟨пусто⟩ / ⟨список операторов⟩
1.3. ⟨список операторов⟩: := ⟨оператор⟩/⟨список операторов⟩ ⟨оператор⟩
1.4. ⟨оператор⟩::= ⟨левая часть⟩ ⟨выражение⟩
1.5. ⟨выражение⟩::= ⟨первичное⟩/⟨выражение сообщения⟩
1.6. ⟨выражение сообщения⟩::= ⟨сообщение⟩/⟨выражение сообщения⟩; ⟨дополнение⟩
1.7. ⟨сообщение⟩::= ⟨унарное сообщение⟩/⟨бинарное сообщение⟩/⟨ключевое сообщение⟩
1.8. ⟨дополнение⟩::= ⟨унарный селектор⟩/⟨бинарная посылка⟩/⟨список ключевых посылок⟩
1.9. ⟨унарное сообщение⟩::= ⟨унарный объект⟩ ⟨унарный селектор⟩
1.10. ⟨унарный объект⟩::=⟨первичное⟩/⟨унарное сообщение⟩
1.11. ⟨бинарное сообщение⟩::= ⟨бинарный объект⟩ ⟨бинарная посылка⟩
1.12. ⟨бинарный объект⟩::=⟨унарный объект⟩/⟨бинарное сообщение⟩
1.13. ⟨бинарная посылка⟩:: = ⟨бинарный селектор⟩ ⟨унарный объект⟩
1.14. ⟨ключевое сообщение⟩::= ⟨бинарный объект⟩ ⟨список ключевых посылок⟩
1.15. ⟨список ключевых посылок⟩ : : = ⟨ключевая посылка⟩/⟨список ключевых посылок⟩⟨ключевая посылка⟩
1.16. ⟨ключевая посылка⟩:: = ⟨ключевой селектор⟩ ⟨бинарный объект⟩
1.17. ⟨первичное⟩ :: = ⟨литерал⟩/⟨переменная⟩/⟨блок⟩/(⟨выражение⟩)

Нераскрытые нетерминальные символы являются терминальными множествами и определяются следующей группой синтаксических правил.

2.1. ⟨заголовок⟩ :: = ⟨образец сообщения⟩ ⟨описание переменных⟩
2.2. ⟨образец сообщения⟩::= ⟨унарный селектор⟩/⟨бинарный шаблон⟩/⟨ключевой шаблон⟩
2.3. ⟨бинарный шаблон⟩::= ⟨бинарный селектор⟩ ⟨параметр⟩
2.4. ⟨ключевой шаблон⟩::= ⟨ключевой селектор⟩ ⟨параметр⟩/⟨ключевой шаблон⟩ ⟨ключевой селектор⟩ ⟨параметр⟩
2.5. ⟨описание переменных⟩::= ⟨пусто⟩/VL ⟨список переменных⟩ VL
2.6. ⟨список переменных⟩::= ⟨переменная⟩/⟨список переменных⟩ ⟨разделитель⟩ ⟨переменная⟩
2.7 ⟨левая часть⟩::= ⟨указатель возврата⟩ ⟨присваивание⟩
2.8. ⟨указатель возврата⟩::= ⟨пусто⟩/SHIFT
2.9. ⟨присваивание⟩::= ⟨пусто⟩/⟨список левой части⟩
2.10. ⟨список левой части⟩::= ⟨переменная⟩ ⟨— /⟨список левой части⟩ ⟨переменная⟩ ⟨—
2.11. ⟨блок⟩::= [⟨аргументы⟩ ⟨операторы⟩]
2.12. ⟨аргументы⟩::= ⟨пусто⟩/⟨список аргументов⟩ VL
2.13. ⟨список аргументов⟩::= : ⟨аргумент⟩/⟨список аргументов⟩ : ⟨аргумент⟩

В последней группе правил определяются оставшиеся терминальные множества.

3.1. ⟨пусто⟩::=
3.2. ⟨унарный селектор⟩::= ⟨разделитель⟩ ⟨идентификатор⟩
3.3. ⟨бинарный селектор⟩::=—/⟨специальный знак⟩/⟨специальный знак⟩ ⟨специальный знак⟩
3.4. ⟨ключевой селектор⟩::=⟨разделитель⟩ ⟨идентификатор⟩:
3.5. ⟨переменная⟩::= ⟨идентификатор⟩
3.6. ⟨параметр⟩::=⟨идентификатор⟩
3.7. ⟨аргумент⟩::=⟨идентификатор⟩
3.8. ⟨литерал⟩::=⟨число⟩/ ⟨символьная константа⟩/⟨знаковая константа⟩/⟨строка⟩/⟨массив-константа⟩
3.9. ⟨число⟩::=⟨основание⟩ ⟨знаковая часть⟩ ⟨число без знака⟩
3.10. ⟨основание⟩::=⟨пусто⟩/⟨целое⟩ r
3.11. ⟨знаковая часть⟩::=⟨пусто⟩/ —
3.12. ⟨число без знака⟩::=⟨целое⟩ ⟨дробная часть⟩ ⟨порядок⟩
3.13. ⟨целое⟩::=⟨цифра⟩/⟨целое⟩ ⟨цифра⟩
3.14. ⟨цифра⟩::=0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
3.15. ⟨дробная часть⟩::=⟨пусто⟩/.⟨целое⟩
3.16. ⟨порядок⟩::=⟨пусто⟩/e ⟨знаковая часть⟩ ⟨целое⟩
3.17. ⟨символьная константа⟩::=#⟨символ⟩
3.18. ⟨символ⟩::=⟨идентификатор⟩/⟨унарный селектор⟩/⟨бинарный селектор⟩/⟨список ключевых селекторов⟩
3.19. ⟨список ключевых селекторов⟩::= ⟨ключевое слово⟩/⟨список ключевых селекторов⟩ ⟨ключевое слово⟩
3.20. ⟨ключевое слово⟩::=⟨идентификатор⟩:
3.21. ⟨знаковая константа⟩::= $ знак⟩/ $'/ $ ' '
3.22. ⟨знак⟩::= [/]/{/}/(/)/⟨— /SHIFT/, /; / $ /\ /#/:/ ⟨цифра⟩/⟨буква⟩/⟨специальный знак⟩

3.23. ⟨строка⟩::='⟨список строчных⟩'
3.24. ⟨список строчных⟩::=⟨строчное⟩/⟨список строчных⟩ ⟨строчное⟩
3.25. ⟨строчное⟩::=⟨знак⟩/''/"/
3.26. ⟨массив-константа⟩::=#⟨массив⟩
3.27. ⟨массив⟩::=(⟨список элементов⟩)
3.28. ⟨список элементов⟩::= ⟨пусто⟩/⟨элемент⟩/⟨список элементов⟩ ⟨разделитель⟩ ⟨элемент⟩
3.29. ⟨элемент⟩ ::=⟨число⟩/⟨символ⟩/⟨строка⟩/⟨знаковая константа⟩/⟨массив⟩
3.30. ⟨идентификатор⟩::=⟨буква⟩/ ⟨идентификатор⟩ ⟨буква⟩ /⟨идентификатор⟩ ⟨цифра⟩
3.31. ⟨буква⟩::=A/B/. . ./Z/a/b/. . ./z
3.32. ⟨специальный знак⟩::=+/./\/*/~/</>/=/@/%/VL/&/?/!
3.33. ⟨разделитель⟩::= ⟨символ разделителя⟩ / ⟨разделитель⟩ ⟨символ разделителя⟩
3.34. ⟨символ разделителя⟩::=SPACE/TAB/ENTER

Рассмотрим примеры соответствующих конструкций:

числа: 3 30.45 —3 0.005 —14.0 13772 8r377 8r34.1 16r—1.C 2r11e6;

знаковая константа: $a $M $ — $$ $1;

строка: 'hi' 'can"t" 'the Smalltalk-80';

символьная константа: #bill #M63;

массив-константа: # (1 2 3) #(9 'nine' $9 nine (0 'zero' ( ) 'e' $f 'g' $ h 'i'));

переменная: index Rectangle bin14Total;

унарный селектор: SPACEnext SPACEa «next a»;

бинарный селектор: — + < =

ключевой селектор: at: put:

образец сообщения: next «унарное»
+var «бинарное»
at : index put : dict «ключевое»

унарное сообщение: a size
x sin arcsin

бинарное сообщение: a + b
a + b — (c * d — e)

ключевое сообщение: expenditures at: reason put: (a b/2) —10

выражение: 1 «первичное-литерал»
(a b c) «первичное-массив_константа»
a b: cd: e «ключевое сообщение»
A b; +1; size: x «выражение с дополнением »

оператор: a

SHIFTb "^b"
SHIFTc ⟨— d + e "c ⟨— d + e"

Метод: cashOnHand
self
SHIFTcashOnHand "^ cashOnHand"

блок: [ ]
[: array VL total ⟨—total + array size]

В тексте описания метода могут быть комментарии в виде «комментарий».

В очевидных случаях символы SHIFT заменяются символом «^», а символы VL — символом «\». Будут также использоваться русские буквы. Например,

[:массив\ ⟨итого⟩ —итого + массив размер]

1.5. Семантика основных конструкций

1.5.1. Общие положения

Описание метода состоит из образца сообщения, возможного описания приватных имен переменных метода и списка операторов, реализующих описываемый метод. Образец сообщения представляет собой шаблон, т. е. указание селекторов и возможных параметров (для бинарных и ключевых) сообщения, посылаемого объекту. Этот шаблон используется при описании соответствующего протокола, например:

class name FinancialHistory
superclass Object
instance methods
запросы «категория сообщений»
cashOnHand «образец унарного сообщения»

⟨операторы⟩ EOM

totalReceivedFrom: source «образец ключевого сообщения»

⟨операторы⟩ EOM

Для этого класса экземплярный протокол будет следующим:

Financial History	instance protocol
запросы	«категория сообщений»
cashOnHand	возвращает значение суммы наличных денег
totalReceivedFrom: source	возвращает значение суммы денег, полученных из источника, задаваемого параметром

Предположим, что рассматриваемый клас FinancialHistory реализует задачу учета доходов и расходов некоторого лица, включая учет их по соответствующим статьям. Пусть создан объект — экземпляр этого класса с именем Smith, в котором реализуется учет доходов и расходов мистера Смита. В этом случае посылка сообщения

Smith cashOnHand

вызовет вывод значения суммы наличных денег, а

Smith totalReceivedFrom: 'salary'

вызовет вывод значения общей суммы доходов по статье «зарплата» ('salary').

Реализация метода определяется последовательностью операторов, разделенных символом «.» (точка). Синтаксис допускает

пустой список операторов с точки зрения удобства занесения описания реализации по частям в режиме диалога, а также для удобного блокирования некоторых методов соответствующих суперклассов своеобразными «заглушками» в процессах отладки.

Оператор состоит из выражения, которому может предшествовать левая часть. Выражение может быть первичным либо представлять собой сообщение. Первичное выражение может быть литералом, переменной, блоком либо представлять собой произвольное выражение, заключенное в круглые скобки (с традиционным смыслом изменения порядка вычисления).

1.5.2. Литералы

Литералы представляют собой конструкции, вычисление которых заключается в порождении новых объектов — экземпляров соответствующих классов (о системных классах, экземплярами которых могут быть все виды литералов). Другими словами, интерпретация литерала ST-машиной в процессе выполнения выражения соответствует посылке сообщений к метаклассам, в результате которых порождаются новые объекты требуемых классов, задающих литералы. Например,

```
2          — порождается объект класса SmallInteger со
             значением 2
$m         — порождается объект класса Character со значением m
'hell'     — порождается объект класса String со значением 'hell'
#St        — порождается объект класса Symbol со значением St
#(a b c) — порождается объект класса Array со значением (a, b, c)
```

1.5.3. Переменные

Следующим видом первичного является переменная, задаваемая идентификатором. В отличие от традиционных языков программирования переменная ST означает только некоторое имя, которое динамически может быть связано с произвольными объектами и поэтому не имеет заранее определенных атрибутов.

Имеется 6 типов переменных, задаваемых идентификаторами:

экземплярные переменные, существующие в течение всего жизненного цикла объекта — экземпляра класса, в заголовке которого они описаны;

временные переменные, описываемые внутри данного метода, существующие только во время выполнения данного метода;

переменные класса, описываемые в описании реализации класса, доступные всем экземплярам данного класса;

глобальные переменные, доступные всем экземплярам всех классов, описываемые в специальном объекте — словаре Smalltalk, экземпляре класса Dictionary;

переменные пула, описываемые в описании реализации класса и доступные экземплярам некоторого подмножества классов;

псевдопеременные, специальные зарезервированные идентификаторы, указывающие на специальные объекты. Будут вводиться ниже.

Переменные класса, глобальные переменные и переменные пула называются разделяемыми переменными. Экземплярные и временные переменные являются приватными. В ST принято соглашение, что идентификаторы первых двух типов переменных, а также псевдопеременных начинаются с маленькой (строчной) буквы, а разделяемые — с большой (прописной). Например,

```
class name                  FH
superclass                  Object
instance variable name      in
                            out
class variable names        Sales
                            Rate
shared pools                Constant
    .
    .
    .
методы
    .
    .
    .
methodl : rll               «образец сообщения»
\x y z\
    .
    .
    .
```

В данном примере FH заносится в словарь Smalltalk и является глобальной переменной; in, out — экземплярные переменные; Sales, Rate — переменные класса; Constant — переменная пула; а x, y, z — временные переменные метода methodl.

Идентификаторы ST также используются для представления параметров сообщения, которые описываются в образце сообщения и, таким образом, выполняют роль временных переменных внутри операторов, реализующих метод.

Аналогично идентификаторами обозначаются аргументы (см. ниже. . ..). Как параметры метода они соответствуют временным переменным (внутри блока) и начинаются со строчной буквы.

Синтаксически идентификаторами обозначаются также унарные селекторы. Синтаксис выражений-сообщений позволяет однозначную интерпретацию идентификаторов — унарных селекторов.

Псевдопеременные представляют собой специальный случай ссылок на объекты, при котором связанные с ними объекты всегда одинаковы. Например,

```
nil  — псевдопеременная, указывающая на специальный
       объект nil, используемый для указания отсутствия ка-
       кого-либо другого значения переменных
true — псевдопеременная, указывающая на объект true, ис-
       пользуемый как логическое значение «истина»
self — псевдопеременная, ссылающаяся на объект-получатель
       сообщения. Например,
                    class name FH
                        .
                        .
                        .
                    method
                        self x
                        .
                        .
                        .
```

В случае посылки объекту-экземпляру FH унарного сообщения method будет выполняться оператор self x, представляющий собой унарное сообщение с селектором x самому себе. Предполагается, что соответствующий метод с селектором x описан в этом же классе. В некоторых классах допускаются экземплярные индексные переменные. Их идентификация производится указанием индекса без имени. Этот тип переменных будет рассмотрен в п. 4.

1.5.4. Сообщения

Сообщения ST могут быть простыми и каскадированными. Простые сообщения делятся на три вида: унарные, бинарные и ключевые.

Унарные сообщения представляют собой конструкцию, задающую унарный объект-получатель сообщения и унарный селектор-идентификатор, определяющий метод, который необходимо выполнить получателю. Унарный объект может в свою очередь представлять собой унарное выражение, что приводит к последовательной слева направо интерпретации унарных сообщений. Например,

a b — объекту a посылается сообщение с селектором b
a b c — объекту a посылается сообщение с селектором b
В результате будет получен некоторый новый объект (допустим, r), к которому будет послано сообщение с селектором c

Бинарное сообщение представляет собой конструкцию, специально введенную для привычного представления арифметико-логических операций. Селектор бинарного сообщения представляется одним-двумя специальными знаками. Получателем сообщения является бинарный объект, а параметром — унарный объект. Бинарный объект является унарным объектом, или бинарным сообщением, что синтаксически определяет приоритетность вычисления сначала унарных объектов, а затем формирование собственно бинарного сообщения. Например,

a+b — объекту a посылается сообщение с селектором + и параметром b
a+b*c — объекту a посылается сообщение +b, а выработанному объекту (сумме a+b) далее посылается сообщение *c

В последнем примере правила вычисления не будут совпадать с общепринятыми, для сохранения нужной приоритетности операций сложения и умножения выражение должно быть записано a+(b*c) и унарный объект (b*c) должен быть вычислен для посылки сообщения с селектором +.

M size + a b c — 1 — бинарному объекту M size, представляющему унарное сообщение объекту M с селектором size, будет послано сообщение +⟨параметр⟩, где ⟨параметр⟩ является унарным объектом — результатом унарного сообщения a b c. Полученному объекту — результату бинарного сообщения — будет послано бинарное сообщение с селектором — и параметром 1.

Ключевое сообщение представляет собой конструкцию, допускающую несколько параметров. Селекторы этого сообщения задаются идентификаторами, заканчивающимися символом двоеточие

(:), и служат фактическими разделителями параметров. Получателем и параметрами ключевого сообщения являются бинарные объекты, что синтаксически определяет порядок вычисления: сначала определяются соответствующие бинарные объекты (что в свою очередь предопределяет вычисление соответствующих унарных объектов), после чего посылается ключевое сообщение. Например,

DFH at: index put: 1 — объекту DFH посылается сообщение с селектором at: put: и параметрами index и 1
x—a b : c d : e + 1.3 f — объекту — результату бинарного сообщения x—a посылается сообщение с селектором b: d: и параметрами c и объектом — результатом бинарного сообщения e + 1.3 f, в котором параметр является результатом унарного сообщения 1.3 f

Имеется специальный вид сообщений (всех трех простых типов), называемый каскадированным. Этот вид конструкции употребляется для лаконичной записи группы сообщений, посылаемых одному объекту. В этом случае основное сообщение имеют дополнения, разделенные символом точка с запятой (;). Например,

a b; c; d — соответствует
a b
a c
a d

1.5.5. Блок

Блоки представляют собой объекты специального вида, вычисление которых (т. е. выполнение списка внутренних операторов) производится путем посылки к ним сообщений с селектором value. Например,

```
actions at:  'mp'
      put: [a b c.
            e f g]
```

представляет ключевое сообщение объекту actions с селектором at: put:, параметрами 'mp' и блоком [a b c. e f g]. Составляющие операторы блока в момент посылки сообщения не выполняются. Здесь предполагается, что данное сообщение интерпретируется объектом actions как ассоциирование строки 'mp' и нового объекта-блока [. . .]. На этот объект можно сослаться путем посылки сообщения

actions at: 'mp'

Таким образом,

(actions at: 'mp') value

будет унарным сообщением, которое вызовет вычисление блока, т. е. выполнение операторов-сообщений a b c. e g h

Синтаксически список операторов блока может быть пуст, в частности, сообщение

[ ] value

вызовет выработку объекта nil

Блок может иметь аргументы. В этом случае сообщение для его вычисления будет ключевым с селекторами value:. Например,

[: array \ array x] value: # (1 2)
[: x : y \ c+x—y] value: 10.3 value : —15.43

На практике блок как новый объект связывается (см. ниже) с именем некоторой переменной, которой в требуемом месте явно или неявно посылается сообщение value.

1.5.6. Левая часть

Оператор, основную часть которого составляют выражения рассмотренных типов, может иметь так называемую левую часть, указывающую на операцию присваивания имени переменной значения (объекта) выражения, а также на возвращаемое значение. Например,

a⟨—b — объект b связывается с именем a и ссылка на него допускается в дальнейшем как на a (в соответствии с правилами области действия переменной a)

a⟨—b⟨—c+1 — объект, результат бинарного сообщения c+1 связывается с именами переменных a и b

↑a — объект a будет возвращен как результирующее значение сообщения посланного методу, содержащему ↑a

↑a⟨—b — объект b будет связан с именем a и возвращен пославшему сообщение как результат

1.6. Интерпретация метода

В качестве обобщенного иллюстрирующего примера рассмотрим строгую интерпретацию выполнения сообщения. Пусть описание реализации метода будет следующим:

```
class name                  Example
superclass                  Object
instance variable names     sumBlock
                            incrementBlock
                .
                .
                .
instance methods
sum: index
    \sum\                   «образец ключевого сообщения»
incrementBlock ⟨— [index ⟨— index + 1]
sumBlock ⟨— [sum + (index * index)]
sum ⟨—0
sum ⟨— sumBlock value
incrementBlock value
↑ sumBlock value
EOM
```

Посылка объекту-экземпляру описанного класса сообщения:

```
E1 sum: 2
```

вызовет следующие действия.

1. Создать по литералу—параметру сообщения новый объект 2.
2. Инициализировать выполнение метода с ключевым селектором sum: в объекте E1 — экземпляре класса Example. При этом параметр index связывается с объектом — константой 2.
3. Создать новый объект-блок: [index ⟨— index +1].
4. Связать этот объект с именем экземплярной переменной incrementBlock.
5. Создать новый объект-блок: [sum + (index * index)]

6. Связать этот объект с именем экземплярной переменной sumBlock.
7. Создать новый объект-константу 0.
8. Связать этот объект с именем временной переменной sum.
9. Послать унарное сообщение с селектором value объекту, поименованному переменной sumBlock, что вызовет выполнение действий:
   9.1. Послать объекту index бинарное сообщение * index, результатом которого будет новый объект-константа 4.
   9.2. Послать объекту sum бинарное сообщение +4, результатом которого будет новый объект-константа 4.
10. Связать объект — результат вычисления блока (т. е. 4) с именем sum.
11. Послать унарное сообщение с селектором value объекту, поименованному переменной incrementBlock, что вызовет такую последовательность действий:
    11.1. Послать объекту index бинарное сообщение +1, результатом которого будет новый объект-константа 3.
    11.2. Связать этот объект с именем index.
12. Послать унарное сообщение с селектором value объекту, поименованному переменной sumBlock, что вызовет следующую последовательность действий:
    12.1. Послать объекту index бинарное сообщение * index (т. е. * 3), результатом которого будет новый объект-константа 9.
    12.2. Послать объекту sum бинарное сообщение +9, результатом которого будет новый объект 13.
13. Возвратить объект-константу 13 в качестве значения отправителю.

Отметим, что после выполнения метода все промежуточные объекты, создаваемые при вычислении выражений, уничтожаются

1.7. Иерархия классов

Как указывалось в предисловии, классы ST организованы в некоторую двухмерную иерархию подчинения, одна из которых: подкласс — класс— суперкласс определяет правила наследования доступа экземплярных переменных и методов класса в соответствующих подклассах. Вторая иерархия — класс — метакласс будет рассмотрена ниже.

Общие правила наследования свойств суперкласса (т. е. экземплярных переменных и методов) заключаются в следующем. Сообщение, посылаемое объекту-экземпляру некоторого класса, вызывает первоначальный поиск соответствующего метода в этом экземпляре (т. е. описанного в реализации данного класса).

Если метод отсутствует, то происходит поиск его в ближайшем суперклассе, затем в суперклассе суперкласса и т. д. В случае отсутствия метода во всей цепи (т. е. до внешнего класса) инициализируется сообщение об ошибке. Эта тривиальная интерпретация правил наследования несколько усложнена в случае использования псевдопеременных self и super.

Псевдопеременная self обозначает объект, которому послано внешнее сообщение, т. е. от объекта, не входящего непосредственно в иерархию объекта-получателя. В этом случае метод, реализованный в суперклассе данного объекта и использующий псевдопеременную self, может передать сообщение внутрь, т. е. в соответствующие подклассы.

Псевдопеременная super определяет, что поиск метода должен начаться в ближайшем суперклассе получателя, даже если метод реализован в классе получателя.

Эти правила проиллюстрируем следующим примером:

```
class name                  One
superclass                  Object
instance  methods
          test
            ^1
          result1
            ^self test

class name                  Two
superclass                  One
instance  methods
          test
            ^2

class name                  Three
superclass                  Two
instance  methods
          result2
            ^self result1
          result3
            ^super test

class name                  Four
superclass                  Three
instance  methods
          test
            ^4
```

Пусть созданы экземпляры описанных классов с именами ex1, ex2, ex3, ex4 соответственно.

Тогда выражение ex3 test вызовет поиск метода test в цепочке классов Three—Two. Найденный в классе Two метод test возвратит результат 2.

Сообщение: ex4 result1 вызовет цепочку поиска Four — Three—Two — One. Найденный в классе One метод result1 посылает сообщение self test. В этом случае псевдопеременная self обозначает объект ex4, но не ex1, и поэтому возвращаемым значением на исходное сообщение будет 4.

Следующие сообщения вызовут получение соответствующих результатов:

```
ex3 result2 — 2
ex4 result2 — 4
ex3 result3 — 2
ex4 result3 — 2
```

Правила наследования экземплярных переменных заключаются в том, что объект — экземпляр подкласса обладает копией набора всех экземплярных переменных соответствующих суперклассов, которые могут быть связаны с индивидуальным для данного экземпляра набором объектов.

Если выполняется метод, найденный в суперклассе, в котором употребляются имена переменных этого или охватывающих суперклассов, то фактические манипуляции будут производиться с объектами, связанными с комплектом имен объекта — экземпляра подкласса, которому послано исходное сообщение.

В ряде случаев необходимы объекты, обладающие свойствами пересечения некоторых классов, но не включения, т. е.

C1 не является подклассом C2,

C2 не является подклассом C1,

пересечение C1 и C2 не пусто.

В таком случае создается специальный суперкласс обоих классов, описывающий их общую часть. Этот суперкласс называется абстрактным классом и не создает своих экземпляров, а только служит для наследования общих методов. Иллюстрацией подобной ситуации может служить организация словарей, таблиц и т. п. Так, например, целесообразно иметь различные организации словарей, в одной из которых реализован быстрый хэш-поиск, а в другой за счет уменьшения скорости поиска экономится память. Тогда специфика организации может быть отражена в описании двух классов, например SmallDictionary и FastDictionary, а собственно внешние операции обращения к словарю (чтение по ключу, запись по ключу) вынесены в описание реализации внешнего абстрактного класса, например Dictionary.

Для того чтобы избежать нежелательных эффектов от непредусмотренного наследования методов суперклассов (иначе при разработке новых классов необходимо было бы хранить слишком много сведений о состоянии ST-машины), которые могут быть особенно тонкими при использовании в методах псевдопеременной self, в универсальном классе OBJECT реализованы два унарных сообщения с селекторами

subclassResponsibility — вызывает вывод информации о том, что данный метод должен быть реализован в подклассе,

shouldNotImplement — вызывает вывод информации о том, что сообщение не должно было быть послано данному объекту.

Ниже будут рассмотрены примеры использования этих сообщений для обрамления области иерархии классов, в которых необходимо определить требуемый набор методов.

Вторая иерархия класс — метакласс связана с рассмотрением классов как объектов — экземпляров соответствующих метаклассов. Введение этой иерархии связано с необходимостью выполнения таких операций-методов, как создание новых экземпляров классов и инициализация значений переменных классов (т. е. общих для всех экземпляров класса). Сохранение концептуального единства построения ST-машины в отношении всех видов описаний и операций привело к следующей схеме иерархии.

При создании нового класса автоматически порождается соответствующий метакласс. Считается, что этот метакласс имеет только один экземпляр — данный класс. Созданный метакласс не имеет имени, и доступ к нему осуществляется посылкой унарного сообщения class к имени основного класса.

Методы этого метакласса приводятся в описании реализации класса под заголовком class methods. При этом для создаваемых классов — метаклассов сохраняется иерархия классы — суперклассы.

Например, рассмотрим следующие классы:

```
class name          One
superclass          Object
    .
    .
    .
class method
    new1            «создание нового экземпляра»
    .
    .
    .
instance method
    m1:
    .
    m2:
    .
    m3:
class name          Two
superclass          One
    .
    .
    .
class method
    new 2           «создание нового экземпляра»
    .
    .
    .
instance method
    m2:
    .
    m4:
```

При вводе класса One в ST-системе будет создан метакласс One class, в котором реализован метод с селектором new1.

При вводе класса Two будет создан метакласс Two class, в котором реализован метод с селектором new2. Метакласс One class является суперклассом метакласса Two class, поскольку эта иерархия зафиксирована в основных классах One, Two. Класс Object, являющийся, как указывалось ранее, классом всех классов, также имеет формально свой метакласс Object class, который является суперклассом класса One class.

В ST введен специальный абстрактный класс Metaclass, являющийся суперклассом всех метаклассов и отражающий наиболее общие свойства классов.

Для завершения этой иерархии введены также класс Metaclass class, экземплярами которого являются непоименованные метаклассы всех используемых классов, но который в свою очередь имеет свой непоименованный метакласс, ссылка на который входит в иерархию суперкласса Class. А для объединения классов и их экземпляров введен абстрактный класс, называемый Class Description. Ближайшими его подклассами являются Metaclass и Class.

Для введенных выше двух классов One и Two рассмотренные иерархии приведены на рис. 7.

В прямоугольных рамках представлены классы, а точки внутри прямоугольников обозначают экземпляры соответствующих классов. Так, класс Two содержит три объекта-экземпляра, One — два. Этим классам соответствуют их собственные метаклассы Two

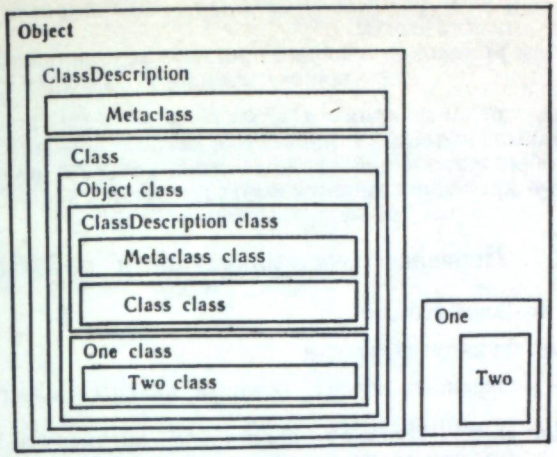


Рис. 7. Иерархия классов.

class и его суперкласс One class. Эти метаклассы имеют соответственно по одному экземпляру-объекту, т. е. сами классы One, Two.

Введение классов Object, ClassDescription, Metaclass и Class соответственно порождает свои метаклассы, находящиеся в соответствующей иерархии (класс — суперкласс), Object class, ClassDescription class, Metaclass class и Class class. Таким образом, класс Metaclass class содержит 6 экземпляров метаклассов указанных классов, и они имеют соответствующий абстрактный суперкласс Class.

Поскольку всеобъемлющим классом является класс Object, не имеющий суперкласса, то его метакласс Object class имеет суперкласс Class. Как указывалось ранее, классы Class и Metaclass имеют суперкласс ClassDescription, и окончательно — классы One и Class Description имеют суперкласс Object.

Глава 2

КЛАСС OBJECT

В п. 1.3 было введено понятие протоколов как средства представления возможностей ST-машины для пользователя. Здесь и ниже мы рассмотрим ряд протоколов (классов и экземпляров), характеризующих основные системные классы ST.

Некоторые наиболее общие действия реализованы в классе Object, что означает доступность этих действий для любых объектов ST.

С точки зрения классификации и группирования методов класса ранее было введено понятие категории методов, т. е. наименование группы методов, являющихся не столько синтаксическим элементом описания классов (и соответствующих протоколов), сколько принятым характером диалоговой дезагрегации элементов описания класса. Ниже категории методов и комментарии будут вводиться на русском языке, а образцы сообщений — как в [10].

Object instance protocol — означает экземплярный протокол класса Object

Class class protocol — означает протокол класса (реализованный в соответствующем метаклассе) для класса Class

Методы, реализованные в классе Object, сгруппированы в следующие шесть категорий: проверка функциональности объектов; сравнение объектов; копирование объектов; доступ к частям объектов; печать и хранение объектов; обработка ошибок.

2.1. Проверка функциональности объектов

Object instance protocol

проверка функциональности

class — возвращает объект, который является классом получателя

isKindof: aClass — ответ — является ли аргумент aClass классом или суперклассом получателя (true, false)

isMemberOf: aClass — ответ — является ли получатель экземпляром аргумента aClass

respondsTO: aSymbol — ответ — есть ли в цепи получатель и его суперклассы метод с селектором aSymbol

Примеры.

Сообщение	Результат
3 class	SmallInteger
# (this is an array) isKindOf: Collection	true
#ST isMemberOf: Array	false
Object respondsTo: #class	true

2.2. Сравнение объектов

Object instance protocol

сравнение

== anObject — ответ — true, если получатель и аргумент тождественны (т. е. один и тот же объект), false — в противном случае

= anObject — ответ — true, если получатель и аргумент равны, false — в противном случае

~ = anObject — true, если получатель и аргумент не равны

~ ~ anObject — true, если получатель и аргумент не тождественны

hash — целое, вычисленное в соответствии с представлением получателя для хэширования. Два одинаковых объекта дают одно значение hash. Разные объекты могут давать или не давать равные хэш-значения

isNil — true, если получатель nil

notNil — true, если получатель не nil

Примеры.

nil isNil	true
3 = (6/2)	true
#(1 2 3) class == Array	true
3 isNil	false

2.3. Копирование объектов

Object instance protocol

копирование

Copy — возвращает копию получателя

ShallowCopy — возвращает копию получателя, разделяющую экземплярные переменные получателя

deepCopy — возвращает копию получателя с собственным комплектом экземплярных переменных

Примеры.

a ← #('first' ' second' 'third')	(('first' 'second' ' third')
b ← a copy	('first' ' second' ' third')
a = b	true
a == b	false

2.4. Доступ к частям объектов

Данная категория методов связана со специальным видом приватной памяти для некоторых классов объектов, задаваемой так называемыми индексными переменными. Индексные переменные не поименованы, и ссылка на них производится с помощью специальных сообщений.

Предполагается, что наличие для данного класса экземплярных индексных переменных определяется следующим описанием в заголовке класса:

```
class name Array
instance variable names x y
indexed instance variables
```

При этом каждый экземпляр описанного класса будет иметь одинаковое количество простых экземплярных переменных (т. е. x y), но может иметь различное количество индексных переменных. Так, например, в классе Array (см. п. 4.5) протокол класса определяет ключевое сообщение с селектором new: anInteger, по которому создается новый экземпляр класса Array (массив) с количеством индексных переменных (т. е. по сути размером массива), равным аргументу anInteger.

В классе Object определены общие операции над индексными переменными объектов.

Object instance protocol

доступ

at: index — возвращает значение индексной переменной, индекс которой определяется параметром index. Некорректность индекса вызывает ошибку

at : index put: anObject — записывает в качестве значения переменной с индексом index объект anObject

basicAt: index — то же, что и at: index, но не может быть модифицирован в подклассах

basicAt: index put: atObject — то же, что и at : put:, но не может быть модифицирован в подклассах

size — возвращает максимально доступное значение индекса переменной

basicSize — то же, что и size, но не может быть модифицирован в подклассах

Примеры.

Пусть f — экземпляр массива #(a, b, c, d, e), тогда

```
f size                  5
f at : 3                c
f at : 4 put: #f        f    f = #(a, b, c, f, e)
```

2.5. Печать и хранение объектов

Object instance protocol

печать

printString — печать строки (String), являющейся описанием получателя

printOn: aStream — добавление к потоку (Stream), заданному параметром, строки (String), являющейся описанием получателя

хранение

StoreString — возвращает строчное (String) представление получателя

StoreOn: aStream — добавляет к потоку, заданному аргументом, строчное представление получателя

Различие в рассмотренных парах методов заключается в том, что хранение предполагает специальный вид записи, который может быть интерпретирован как сообщение, конструирующее объект. Например, объект — множество трех элементов $a, $b, $c — может печататься как Set ($a, $b, $c), а храниться — как каскадированное сообщение (Set new add: $a; add : $b; add : $c).

2.6. Обработка ошибок

Object instance protocol

обработка ошибок

doesNotUnderstand: aMessage — ответ пользователю, что получатель не понимает аргумент aMessage как сообщение

error: aString — общее сообщение об ошибке при выполнении ST-сообщения; параметр является текстовой частью информации об ошибке

primitiveFailed — ошибка при выполнении метода (системных примитивов)

shouldNotImplement — ответ пользователю, что класс получателя не может обеспечить подходящей реализации данного метода

subclassResponsibility — ответ пользователю, что метод, реализованный в суперклассе, должен быть реализован в классе получателя

Глава 3

СКАЛЯРНЫЕ ДАННЫЕ

Название данного раздела является в определенной степени данью традиционной терминологии программирования, базирующейся на понятиях структур данных и управляющих структур, которая прямо не принята в ST-терминологии, однако рассматриваемые ниже системные классы по смыслу близки к этой терминологии, в особенности к понятиям абстракций типов данных, используемых в современных языках программирования.

3.1. Класс Magnitude

Класс Magnitude (Величина) является подклассом класса Object и определяет разнообразные объекты, которые допускают сравнения по величине. Он фактически охватывает традиционные скалярные объекты. Иерархия подклассов данного класса приведена на рис. 8. В этом классе реализованы следующие общие методы.

Magnitude instance protocol

сравнение

< aMagnitude — true, если получатель меньше аргумента

<= aMagnitude — true, если получатель меньше или равен аргументу

> aMagnitude — true, если получатель больше аргумента

>= aMagnitude — true, если получатель больше или равен аргументу

between : min and : max — true, если получатель находится в диапазоне (> = min & < = max)

= aMagnitude — переопределенный метод: self subclassResponsibility

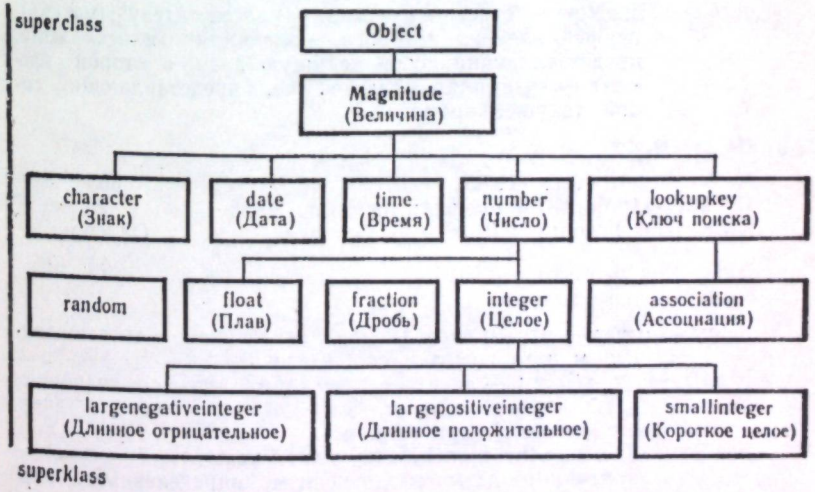


Рис. 8. Иерархия класса Magnitude.

проверка

min: aMagnitude — ответ — получатель или аргумент, имеющие наименьшее значение
max: aMagnitude — ответ — получатель или аргумент, имеющие наибольшее значение

Примеры с использованием подкласса Integers, наследующего соответствующие методы своих суперклассов:

3(=4	true
3>4	false
5 between: 2 and: 6	true
34 max: 45	45

3.2. Класс Date

Объекты класса Date представляют собой даты в форме: «день, месяц, год».

Date class protocol "Методы метакласса"
общие запросы

dayOfWeek: dayName — ответ — индекс дня недели (1, 2, ... , 7), заданного аргументом
nameOfDay: dayIndex — ответ — символ-имя дня недели, заданного аргументом-индексом
indexOfMonth: monthName — ответ — индекс месяца (1, 2, . . . , 12), заданного аргументом
nameOfMonth: monthIndex — ответ — символ-имя месяца, заданного аргументом
daysInMonth: monthName forYear: yearInteger — ответ — число дней в месяце, заданном первым аргументом для года, заданного вторым аргументом
daysInYear: yearInteger — ответ — число дней в году, заданном аргументом
leapYear: yearInteger — ответ — 1 для високосного года и 0 в противном случае
dateAndTimeNow — ответ — экземпляр класса Array (массив), первый элемент которого — экземпляр класса Date, представляющий собой текущую дату, а второй элемент — экземпляр класса Time, представляющий собой текущее время

Примеры.

Date daysInYear: 1986	365
Date daysInMonth: #Февраль for Year: 1986	28
Date nameOfMonth: 10	Октябрь

Date class protocol
создание экземпляров

today — ответ — экземпляр даты со значением, соответствующим дате посылки сообщения
fromDays: dayCount — ответ — экземпляр даты со значением даты, отсчитанной от 1.01.1901 на количество дней, определенной аргументом
newDay: day month: monthName year: yearInteger — ответ — экземпляр даты со значением, определенным аргументом

newDay: dayCount year: yearInteger — ответ — экземпляр даты со значением, отсчитанным от начала заданного года.

Примеры.

Date today	10 июня 1986
Date fromDay : 200	20 июля 1901
Date newDay : 13 month: #Мая year: 1985	13 мая 1985
Day newDay: 7 year: 1986	7 января 1986

Отметим, что созданные объекты — даты после инициализации — меняют свои значения синхронно с течением времени в среде ST.

Сообщения к объектам — экземплярам класса Date — сгруппированы в следующие категории: доступ, запросы, арифметика, печать.

Категории доступа и запросов позволяют следующие операции: индексы дня, месяца, года; число секунд, дней, месяцев от заданной даты; число дней в месяце, году, заданных датой; число оставшихся до конца месяца (года) дней; первый день месяца, заданного датой имена дней недели или месяца; дата конкретного дня недели, предшествующего заданному экземпляром.

Date instance protocol

арифметика

addDays: dayCount — ответ — дата, полученная сложением даты-получателя с количеством дней-аргументом
subtractDays: dayCount — ответ — дата, полученная вычитанием из даты-получателя количества дней, заданных аргументом
subtractDate: aDate — ответ — целое (Integer), представляющее собой число дней — разность получателя и аргумента

Ниже (см. п.4) после введения необходимых управляющих структур будут рассмотрены примеры.

3.3. Класс Time

Объекты класса Time представляют собой значения секунд для дня, отсчитываемые (в реальном относительном времени) от полуночи.

Time class protocol
общие запросы

millisecondClockValue — ответ — число миллисекунд, показываемых системными часами
millisecondsToRun: timedBlock — ответ — число миллисекунд, затраченных на выполнение блока-аргумента (таймирование блока)
timeWords — ответ — количество секунд, прошедших с 1.01. 1901 (по Гринвичу)
totalSeconds — ответ — количество секунд, прошедших с 1.01. 1901 с учетом временного пояса и сезонного времени

создание экземпляров

now — ответ — экземпляр класса, значение которого — секунда посылки сообщения

fromSeconds: secondCount — ответ — экземпляр Time, значение которого задается при создании по значению аргумента

Time instance protocol

запросы

получение значений часов (hours), минут (minutes) и секунд (seconds), определяемых экземпляром-получателем

арифметика

addTime: timeAmount — ответ — экземпляр Time, значение которого равно сумме получателя и аргумента

subtractTime: timeAmount — ответ — экземпляр Time, значение которого равно разности получателя и аргумента

преобразования

asSeconds — ответ — число секунд от полуночи, представляемое получателем

Существует также аналогичное сообщение категории преобразования, посылаемое объектом класса Date.

Date instance protocol

преобразования

asSeconds — ответ — число секунд от 1.01.1901 до дня, определенного получателем

Отметим, что аргументом арифметической категории методов класса Time (timeAmount) могут быть объекты классов Date и Time.

Пример. Выражение

(Time now subtractTime: startTime) minutes > 30

соответствует следующим действиям:

создание экземпляра часов /Time now/ со значением текущего времени;

посылка этому экземпляру сообщения с селектором subtract: startTime, где аргумент startTime является объектом класса Time, фиксирующим некоторое начальное время, результатом чего будет объект класса Time, показывающий разность между текущим временем и начальным (зафиксированным в startTime);

посылка последнему объекту унарного сообщения minutes, результатом которого является числовой объект со значением минут в получателе;

посылка этому числовому объекту бинарного сообщения >30, в результате чего вырабатывается значение true, если число минут больше 30.

Так, если startTime равно 8 ч 25 мин 30 с, а посылка сообщения произведена в 9 ч 30 мин 30 с, то результат будет false.

3.4. Класс Character

Этот класс имеет 256 экземпляров, ассоциированных с кодами ASCII.

Character class protocol

создание экземпляров

value: anInteger — ответ — экземпляр, значение которого соответствует аргументу и ассоциировано с кодом ASCII

digitValue: anInteger — ответ — экземпляр, цифровое значение которого равно аргументу

Примеры.

value : 65 — символ «A»
digitValue : 10 — $A (как 16-ричная цифра)

Character instance protocol

доступ

asciiValue — ответ — число, соответствующее ASCII коду получателя

digitValue — ответ — число, соответствующее получателю (см. digitValue : anInteger)

проверка

isAlphaNumeric — ответ — true, если получатель — буква или цифра

isDigit — ответ — true, если получатель — цифра

isLetter — ответ — true, если получатель — буква

isLowercase — ответ — true, если получатель находится в регистре строчных букв

isUppercase — ответ — true, если получатель находится в регистре прописных букв

isSeparator — ответ — true, если получатель является разделителем (см. синтаксис)

isVowel — ответ — true, если получатель — гласная буква (a, e, i, o, u) в любом регистре

преобразования

asLowercase — преобразование в регистр строчных букв

asUppercase — преобразование в регистр прописных букв

asSymbol — преобразование в объекты класса Symbol

3.5. Класс Number

В этом классе определены операции над числами, однако большинство из них должны быть реализованы в соответствующих подклассах целых, вещественных и других типов чисел, поскольку выполнение операций связано со способами представления соответствующих типов чисел.

Number instance protocol

арифметика

+ aNumber — сложение получателя и аргумента

— aNumber — вычитание

* aNumber — умножение
/ aNumber — деление
// aNumber — целочисленное деление с усечением в сторону минус бесконечность
\\ aNumber — модуль получателя по основанию-аргументу с усечением в сторону минус бесконечность
abs — абсолютное значение
negated — противоположное значение получателя
quo: aNumber — целочисленное деление с усечением в сторону 0
rem: aNumber — модуль получателя по основанию-аргументу с усечением в сторону 0
reciprocal — обратная величина

Примеры.

5.6—3	2.6
(—4) abs	4
7/2	(7/2) дробь
7 reciprocal	1/7
—7 quo: 2	—3
—7 // 2	—4
—7\\2	1

математические функции

exp — e **x, где x — получатель
ln — ln x
log: aNumber — логарифм с основанием-аргументом
floorLog: radix — логарифм по основанию-аргументу с усечением до целого в сторону минус бесконечность
raisedTo: aNumber — возведение получателя в степень, заданную аргументом
raisedToInteger: anInteger — возведение в целочисленную степень, заданную аргументом
sqrt — извлечение квадратного корня
squared — квадрат получателя

проверка

even — ответ — true, если получатель четный
odd — ответ — true, если получатель нечетный
negative — ответ — true, если получатель меньше 0
positive — ответ — true, если получатель больше или равен 0
strictlyPositive — ответ — true, если получатель строго больше 0
sign — ответ — 1, если получатель больше 0; 0, если равен 0, и —1, если меньше 0.

усечение и округление

ceiling — ответ — ближайшее (к получателю) целое в направлении к плюс бесконечности
floor — ближайшее целое в направлении к минус бесконечности
truncated — ближайшее целое в направлении к 0.
truncateTo: aNumber — ближайшее кратное аргументу в направлении к 0
rounded — (округление), ближайшее к получателю целое
roundTo: aNumber — ближайшее к получателю кратное аргументу

Примеры.

16.32 ceiling	17
—16.32 floor	—17
16.32 floor	16
16.32 truncated	16
—16.32 truncated	—16
16.32 truncateTo : 5	15
16.32 roundTo : 6.3	18.9

преобразование

degreesToRadians — преобразование значений в градусах в радианы
radiansToDegrees — преобразование радианов в градусы

Как было рассмотрено ранее, класс чисел (Number) включает собственные подклассы разных типов чисел (Float, Fraction и т. д.), смысл которых аналогичен традиционным языкам программирования и для которых определены традиционные отношения предпочтения и общности, т. е. в арифметических операциях над разными типами результат относится к более общему типу, например, выражение 2+3.5, где 2 — объект класса SmallInteger, а 3.5 — класса Float, предполагает результат в более общем классе Float. Следующая категория методов относится к учету факторов принадлежности операндов к разным подклассам чисел и позволяет производить необходимые приведения.

приведения типов

coerce: aNumber — ответ — число, заданное аргументом, имеющее класс получателя
generality — ответ — число, представляющее ранг получателя в иерархии числовых классов
retry: aSymbol coercing: aNumber — выполнение операции, заданной аргументом aSymbol, для разных классов получателя и аргумента aNumber. При выполнении операции выбирается минимальный класс

Следующая категория дает возможность получить в классе чисел элементы класса Interval (см. ниже).

интервалы

to: stop — ответ — интервал как последовательность чисел от получателя до аргумента stop с шагом 1
to: stop by: step — то же, но с шагом, определяемым аргументом step
to: stop do: aBlock — создание интервала от получателя до аргумента stop с шагом 1 и вычисление блока для каждого элемента интервала
to: stop by: step do: aBlock — то же, но с шагом, определяемым аргументом step

Последние два метода соответствуют известным понятиям циклов типа арифметической прогрессии и предполагают, что если вычисляемый блок (тело цикла) имеет аргумент, то значения интервала передаются этому аргументу.

Например:

a <— 0
10 to : 100 by : 10 do : [:each \ a <— a+each] значение a будет равно 550

3.6. Классы Float и Fraction

Эти классы являются представлениями нецелых чисел. Экземпляры класса Float — вещественные числа типа

0.3
4.27e—30
—12.987654e12

Экземпляры класса Fraction (дробь) представляют рациональные числа в виде дробей, что позволяет получать точные результаты.

3.7. Класс Integer

Эти классы включают три подкласса SmallInteger (короткие целые), LargePositiveInteger (длинные положительные целые), LargeNegativeInteger (длинные отрицательные целые), экземпляры которых обеспечивают соответствующее экономичное представление в памяти. Кроме основных операций, наследуемых из класса Number, класс Integer допускает протокол преобразования (методы с селекторами: asCharacter, asFloat, asFraction), протокол печати (printOn : aStreambase : b, radix : baseInteger, например 8 radix : 2 = = 2r1000) и протокол перечислений (циклы timesRepeat: aBlock). Другие категории следующие.

Integer instance protocol

факторизация и делимость

factorial — факториал получателя
gcd: anInteger — наибольший общий делитель получателя и аргумента
lcm: anInteger — наименьшее общее кратное получателя и аргумента

битовые манипуляции

allMask: anInteger — ответ — true, если все биты 1 в получателе равны 1 в аргументе
anyMask: anInteger — ответ — true, если какой-нибудь бит 1 в получателе — 1 в аргументе
noMask: anInteger — ответ — true, если ни один из битов 1 получателя не равен 1 в аргументе
bitAnd: anInteger — ответ — integer, значение которого — поразрядная конъюнкция получателя и аргумента
bitOr: anInteger — поразрядная дизъюнкция
bitXor: anInteger — поразрядная неравнозначность.
bitAt: index — ответ — бит в позиции, заданной аргументом
bitInvert — инверсия битов получателя
highBit — ответ — индекс старшего значащего бита в двоичном представлении получателя
bitShift: anInteger — двоичный сдвиг на число позиций аргумента. Для отрицательного аргумента сдвиг вправо, для положительного — влево

3.8. Класс Random

Экземпляры этого класса являются генераторами случайных чисел с равномерным распределением в диапазоне 0.0—1.0.
Создание нового экземпляра производится посылкой сообщения new, например

rand <— Random new

Получение нового значения производится посылкой сообщения next, например

rand next

Отметим, что результат rand next число — экземпляр класса Float, хотя системный класс Random является подклассом Stream.

Глава 4

ГРУППОВЫЕ ДАННЫЕ

Основные групповые данные определяются в системном классе Collection. Соответствующая иерархия в этом классе представлена на рис. 9.

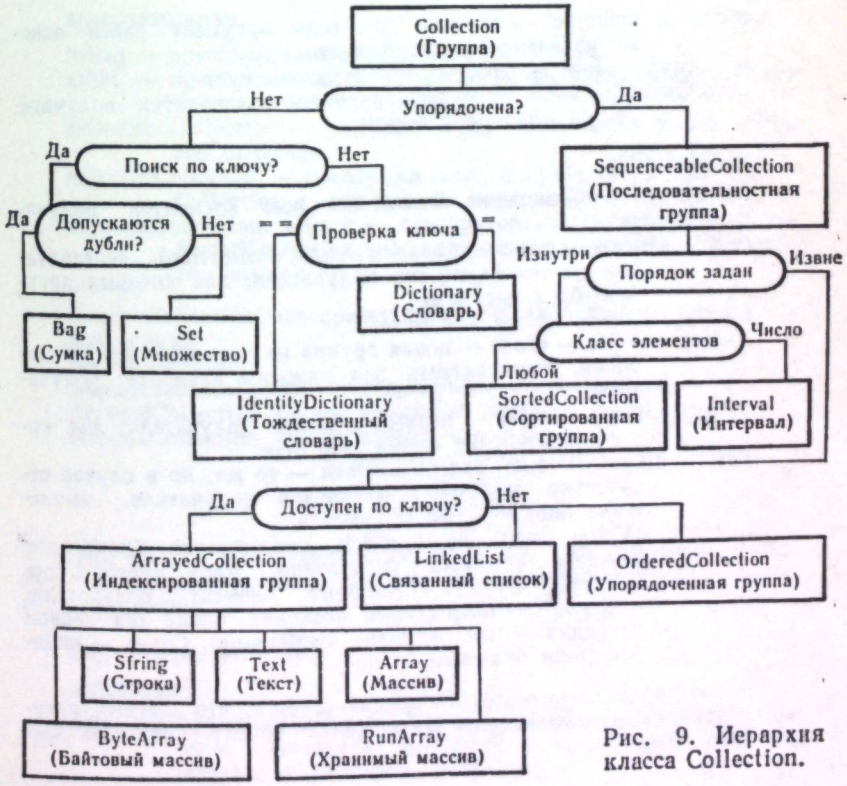


Рис. 9. Иерархия класса Collection.

4.1. Класс Collection

Класс Collection является подклассом класса Object и включает реализацию основных операций добавления, удаления и проверки элементов группы.

Collection instance protocol

добавление

add: newObject — добавляет к получателю новый объект, заданный аргументом
addAll: aCollection — добавляет к получателю группу элементов, заданную аргументом

удаление

remove: oldObject — удаляет из получателя элемент, заданный аргументом
remove: oldObject ifAbsent: anExceptionBlock — то же, если удаляемый объект отсутствует, то вычисляется блок, заданный вторым аргументом
removeAll: aCollection — удаляет все элементы, заданные аргументом

проверка

includes: anObject — ответ — true, если аргумент равен одному из элементов получателя
isEmpty — ответ — true, если получатель пуст
occurencesOf: anObject — ответ — число элементов получателя, равных аргументу

перечисление

do: aBlock — вычисление блока для всех элементов получателя
select: aBlock — ответ — новая группа (Collection), составленная из тех элементов получателя, для которых аргумент-блок дает true
reject: aBlock — то же для false
collect: aBlock — ответ — новая группа из значений аргумента-блока, вычисляемых для каждого элемента получателя
detect: aBlock — ответ — первый элемент получателя, для которого вычисление блока дает true
detect: aBlock ifNone: exceptionBlock — то же, но в случае отсутствия требуемых элементов получателя, вычисление блока — 2-го аргумента.
inject: thisValue into: binaryBlock — вычисление блока для каждого элемента получателя. Блок имеет 2 аргумента. Второй — текущий элемент получателя, а первый — предыдущее значение блока (на первой итерации — 1-й аргумент сообщения). Ответ — окончательное значение блока

В категории «перечисление» предполагается, что первый аргумент блока принимает в качестве значения объект — текущий элемент получателя.

Collection class protocol

создание экземпляров

with: anObject — ответ — экземпляр класса Collection (или его подклассов), содержащий anObject
with: firstObject with: secondObject — ответ — экземпляр класса Collection, содержащий в качестве элементов firstObject и secondObject
with: firstObject with: secondObject with: thirdObject — ответ — экземпляр класса Collection, содержащий в качестве элементов firstObject, secondObject и thirdObject
with: firstObject with: secondObject with: thirdObject
with: fourthObject — ответ — экземпляр класса Collection, содержащий в качестве элементов firstObject, secondObject, thirdObject и fourthObject

Например, для класса Set, являющегося подклассом Collection, сообщение

Set with: $s with: $e with: $t

вызовет создание экземпляра множества из трех элементов — знаков s, e, t.

Collection instance protocol

преобразования

asBag — преобразование получателя в тип Bag
asSet — преобразование получателя в тип Set (одинаковые элементы заменяются одним)
asOrderedCollection — преобразование получателя в тип OrderedCollection
asSortedCollection — преобразование получателя в тип SortedCollection с отношением порядка <=
asSortedCollection: aBlock — преобразование получателя в тип SortedCollection по алгоритму, заданному аргументом aBlock

Класс Collection непосредственно включает подклассы:
Bag (сумка)
Set (множество)
SequenceableCollection (последовательностная группа)
ArrayedCollection (индексированная группа)
MappedCollection (отображаемая группа).

4.2. Класс Bag

Этот класс представляет собой простейший вид групповых данных, не упорядоченных и не имеющих внешних ключей поиска (идентификации).

Bag instance protocol

добавление

add: newObject withOccurrences: anInteger — включает в состав получателя newObject в количестве, заданном вторым параметром (anInteger)

Например, пусть класс Product задает некоторые пары наименование продукции — стоимость и экземпляры этого класса создаются сообщением вида Product of: name at: price. Унарное сообщение price к экземплярам этого класса вызывает получение цены продукции. Тогда накопление информации о различной продукции может быть выражено следующими сообщениями:

```
sack ⟨— Bag new
sack add: (Product of : #steak at : 5.80)
sack add: (Product of : #potatoes at : 0.50)  withOccurrences : 6
```

После выполнения этих сообщений в некоторой сумке (sack) оказывается бифштекс (steak) за 5.80 р. и 6 кг картофеля (potatoes) по цене 0.50 р. за килограмм.

Текущая стоимость продуктов может быть вычислена следующими сообщениями:

```
amount <— 0
sack  do: [:eachProduct\amount ⟨— amount+eachProduct price]
```

или

```
sack inject: 0
     into: [:amount:  eachProduct\amount+eachProduct price]
```

Отметим, что первое сообщение add: , а также do:, inject:, into:, посылаемые к sack, заимствованы из суперкласса Collection.

4.3. Класс Dictionary

Экземпляры этого класса представляют собой множество пар (ассоциаций) ключ — значение. Сами элементы, т. е. пары, являются экземплярами класса Association.

Dictionary instance protocol

доступ

at: key ifAbsent: aBlock — поиск значения по ключу, заданному первым аргументом. При отсутствии выполняется блок, заданный вторым аргументом

associationAt: key — поиск пары ключ — значение по заданному ключу

associationAt: key ifAbsent: aBlock — то же, но в случае отсутствия выполняется блок aBlock

keyAtValue: value — поиск по значению, ответ — ключ для первого найденного значения value или nil при его отсутствии

keyAtValue: value ifAbsent: exceptionBlock — то же, но в случае отсутствия value выполняется блок, заданный вторым аргументом

keys — ответ — множество (Set), содержащее ключи получателя

values — ответ — групповой объект класса Bag, содержащий все значения (в том числе дублированные) получателя

проверка словаря

includesAssociation: anAssociation — ответ — есть ли в получателе пара ключ — значение, заданная аргументом

includesKey: key — ответ — есть ли в получателе ключ, заданный аргументом

удаление

removeAssociation: anAssociation — удаление заданной пары

removeKey: key — удаление пар с заданным ключом

removeKey: key ifAbsent: aBlock — то же, но в случае отсутствия пар с заданным ключом выполняется блок aBlock

перечисление

associationsDo: aBlock — вычисление аргумента-блока для каждой пары получателя

keysDo: aBlock — вычисление аргумента-блока для каждого ключа получателя

4.4. Класс SequenceableCollection

Этот класс (последовательностные группы) задает групповые объекты, элементы которых имеют отношение порядка и могут индексироваться, т. е. иметь внешнюю идентификацию целыми числами — индексами.

Все объекты данного класса наследуют операции из класса Object, позволяющие основные манипуляции доступа к переменным с индексами, в частности at:, at : put: и size. Операциями, определенными в данном классе, являются следующие.

SequenceableCollection instance protocol

доступ

atAll: aCollection put: anObject — ассоциирует каждый элемент аргумента aCollection (целочисленный внешний ключ) со вторым аргументом anObject

atAllPut: anObject — помещает аргумент в качестве каждого элемента получателя

first — первый элемент получателя

last — последний элемент получателя

indexOf: anElement — вычисление первого индекса элемента получателя, равного аргументу

indexOf: anElement ifAbsent: exceptionBlock — то же, но в случае отсутствия заданного элемента выполняется блок, заданный вторым аргументом

indexOfSubCollection: aSubCollection startingAt: anIndex — поиск совпадения группы aSubCollection с соответствующей подгруппой получателя начиная от индекса anIndex. Ответ — индекс первого элемента подгруппы при совпадении, в противном случае 0

indexOfSubCollection: aSubCollection startingAt: anIndex ifAbsent: exceptionBlock — то же, но в случае отсутствия совпадения выполняется блок — третий аргумент

replaceFrom: start to: stop with: replacementCollection — замещение в получателе элементов с индексами от start до stop значениями группы replacementCollection

replaceFrom: start to: stop with: replacementCollection startingAt: repStart — то же, но замещаемой является группа replacementCollection начиная с индекса reStart

Примеры.

'aaaaaaaaaa' atAll: (2 to : 10 by: 2) put $b — результат 'abababababab'

'The cow jumped' replaceSrom: 5 to: 7 with 'dog' — результат 'The dog jumped'

SequenceableCollection instance protocol

копирование

, aSequenceableCollection — конкатенация получателя и аргумента

copyFrom: start to: stop — вырезка получателя от индекса start до индекса stop

copyReplaceAll: oldSubCollection with: newSubCollection — ответ — копия получателя с заменой элементов, заданных аргументом oldSubCollection, элементами, заданными аргументом newSubCollection

copyWith: newElement — ответ — копия получателя, дополненная последним элементом-аргументом newElement

copyWithout: oldElement — ответ — копия получателя с удаленными элементами, равными аргументу

перечисление

findFirst: aBlock — вычисление блока aBlock для каждого элемента получателя, до тех пор пока значение блока не станет true. Ответ — индекс этого элемента

findLast: aBlock — то же, но ответ — индекс последнего элемента получателя, для которого вычисление блока дало true

reverseDo: aBlock — то же, что для do:, но в обратном порядке элементов получателя

with: aSequenceableCollection do: aBlock — блок aBlock имеет два аргумента, которые заменяются элементами получателя и параметра aSequenceableCollection, и вычисляется для всех соответствующих пар

Пример. Образовать словарь антонимов.

```
opposites< — Dictionary new
# (come cold front hot push stop)
   with: # (go hot back cold pull start)
     do: [:key: value\ opposites at : key put: value]
```

В результате данного сообщения будет образован словарь антонимов:

```
come go
cold hot
front back
hot cold
push pull
stop start
```

Класс SequenceableCollection имеет соответствующие подклассы: OrderedCollection (упорядоченная группа), LinkedList (связной список), Interval (интервал).

Упорядоченная группа используется в качестве стеков или очередей. Подклассом этого класса является класс SortedCollection (сортированная группа). Создание новых экземпляров класса SortedCollection вызывается указанием блока, содержащего отношение предшествования элементов.

Так, например, сообщение

```
SortedCollection sortBlock: [:a: b\ a< =b]
```

вызывает создание нового экземпляра класса SortedCollection, элементы которого упорядочены по возрастанию.

Связной список отличается от упорядоченной группы тем, что элементами этой группы могут быть экземпляры класса Link, т. е. структуры с явными указателями на предыдущий и последующие элементы.

Интервал определяет группу числовых значений вида арифметической прогрессии. В частности, сообщение do: к интервалу обеспечивает функции обычного цикла типа for, используемых в традиционных языках программирования. Например, цикл АЛГОЛа

```
for i = 10 step 6 until 100 do
   begin
        ⟨операторы⟩
   end
```

в нотации ST выглядит сообщением

```
(10 to : 100 by : 6) do: [:i\⟨операторы⟩]
```

4.5. Класс ArrayedCollection

Данный класс представляет собой группу, элементы которой имеют фиксированный порядок внешних ключей идентификации элементов в виде целочисленных индексов.

Класс ArrayedCollection включает пять собственных подклассов: Array (массив), String (строка), Text (текст), RunArray (хранимый массив), ByteArray (байтовый массив).

Элементами экземпляров класса Array могут быть произвольные объекты.

Класс String представляет собой группу элементов класса Character.

В классе Text экземпляры String представляются совместно с видами и типами шрифтов (жирный, курсив, подчеркнутый и т. п.). Компактное кодирование видов шрифтов для представления строковых массивов осуществляется в классе RunArray. При этом экземпляр класса Text имеет две экземплярные переменные, которые связываются с соответствующими экземплярами классов String и RunArray.

Класс ByteArray представляет собой группу элементов, которые являются целыми в диапазоне 0—255, и используется в системных целях.

4.6. Класс MappedCollection

Экземпляры этого класса представляют собой механизм доступа к именованным элементам или подгруппам некоторых групп (т. е. экземпляров Collection) через отображение последних на другие имена. Основная идея этого класса заключается в создании некоторого множества пар область определения — область значений, где область определения представляет собой группу с непрямым

доступом, в область значений — множество внешних ключей доступа

Например, пусть кроме словаря антонимов opposites (см. п. 4.4) создан другой словарь синонимов некоторых ключей словаря opposites. Этот словарь связан с именем alternates и содержит следующие пары:

cease	stop
enter	come
scalding	hot
shove	push

Тогда сообщение

```
words      ⟨— MappedCollection collection:opposites
map: alternates
```

создает экземпляр класса MappedCollection, через который антонимы словаря alternates доступны по ключам синонимам. Например:

Сообщения	Результаты
words at:#cease	start
words at:#cease put:#continue	continue
words contents	continue go cold pull

Глава 5

ПОТОКИ И УПРАВЛЕНИЕ

Рассмотренные в предыдущей главе группы допускают представление данных в линейной или другой форме, при этом протоколы групповых классов (экземпляров) допускают прямой или последовательный доступ к элементам. Однако в операциях данных классов непосредственно не допускается одновременное смешивание операций доступа и записи с прерыванием во времени. Для разделенного доступа к последовательности элементов необходимо дополнительно хранить в некоторой внешней памяти информацию о последнем прочитанном (или записанном) элементе последовательности.

Класс Stream предоставляет возможность поддержания позиций ссылок на элементы групп путем наложения на группу некоторого потока, позиционирующего определенным образом элементы группы. Подобное позиционирование позволяет производить операции перечисления, сохранения (записи) и доступа (чтения) элементов смешанно и в разные моменты времени. Путем создания нескольких потоков (Stream) над одной группой можно сохранять множественные ссылки на позиции элементов.

В ST используются три основных способа позиционирования групп. Первый способ удобен для всех последовательностных групп (SequenceableCollection) и использует целочисленный индекс для сохранения текущей позиции элемента группы. Второй способ использует принцип "верна" (или "семени"), заключающийся в том, что сохраняемой позицией является значение элемента генерируемой последовательности, на основании которого генерируется следующее значение (объект). Этот способ применяется в классе Random, рассмотренном п. 3.8. Третий способ используется на древовидных или подобных орграфовых структурах, где позиция связывается с текущим узлом.

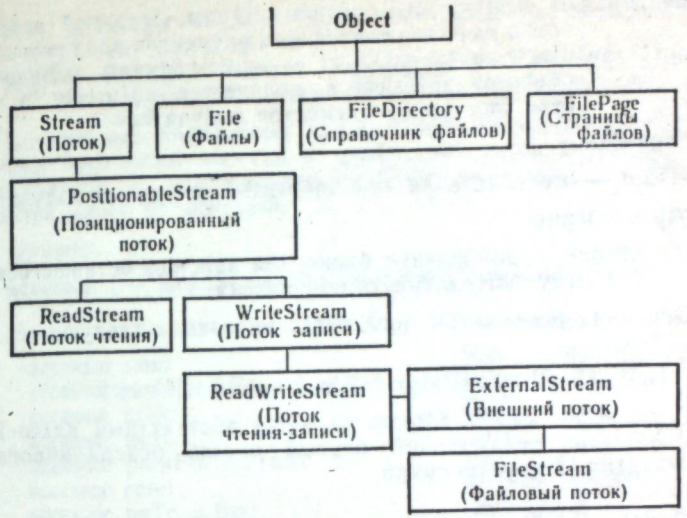


Рис. 10. Классы потоков и файлов.

Понятие потоков связано с реальным аппаратным уровнем ЭВМ, т. е. устройствами ввода — вывода и внешней памяти.

В данном разделе будут также рассмотрены вопросы организации вычислительного процесса.

На рис. 10 изображена структура и взаимосвязь основных классов, относящихся к реализации потоков.

5.1. Класс Stream

Этот класс является подклассом Object и является суперклассом, определяющим протоколы доступа к позиционированным группам. Он включает подкласс PositionableStreams, имеющий подклассы ReadStream и WriteStream, ReadWriteStream.

Stream instance protocol

доступ — чтение

next — ответ — следующий элемент получателя
next: anInteger — ответ — группа следующих элементов получателя, число элементов которой задано аргументом anInteger
nextMatchFor: anObject — доступ к следующему элементу получателя и ответ, равен ли он аргументу
contents — ответ — группа доступных получателю элементов

доступ — запись

nextPut: anObject — запись аргумента anObject в качестве следующего элемента получателя

nextPutAll: aCollection — запись группы, заданной аргументом в качестве следующих элементов получателя
next: anInteger put: anObject — запись группы одинаковых объектов anObject в количестве anInteger в качестве следующих элементов получателя

проверка

atEnd — ответ, есть ли еще элементы, доступные получателю

перечисление

do: aBlock — вычисление блока для каждого оставшегося доступного элемента получателя

Описание реализации последнего протокола следующие:

```
do: aBlock
  [self atEnd] whileFalse: [aBlock value: self next]
```

Подклассом класса Stream является абстрактный класс PositionableStream, отражающий первый способ позиционирования (целочисленный индекс) групп.

PositionableStream class protocol

создание экземпляров

on: aCollection — создание экземпляра потока, позиционирующего группу, заданную аргументом aCollection
on: aCollection from: firstIndex to: lastIndex — создание экземпляра потока, позиционирующего подгруппу, выделенную из аргумента aCollection от элемента firstIndex до элемента lastIndex

PositionableStream instance protocol

проверка

isEmpty — ответ — true, если получатель не имеет элементов

доступ

peek — ответ — следующий элемент получателя без сдвига указателя текущей позиции
peekFor: anObject — поиск (вперед) элемента получателя, совпадающего с аргументом anObject. Если элемент есть, то ответ — true со сдвигом указателя позиции на найденный элемент
upTo: anObject — ответ — группа, выделенная от следующего элемента получателя до элемента, совпадающего с аргументом anObject, или, при его отсутствии, до конца
reverseContents — ответ — копия получателя с элементами в обратном порядке

позиционирование

position — получение значения текущей ссылки
position: anInteger — установка указателя позиции получателя, равной значению аргумента anInteger
reset — установка указателя позиции на первый элемент получателя
setToEnd — установка указателя позиции на последний элемент получателя

skip: anInteger — сдвиг указателя позиции на заданное аргументом число
skipTo: anObject — сдвиг указателя на элемент, заданный аргументом, ответ, существует ли такой элемент

Конкретным подклассом класса PositionableStream, обеспечивающим протоколы доступа к элементам, наследуемые из своих суперклассов, за исключением nextPut:, next : put:, nextPutAll:, является класс ReadStream.

Пример.

```
accessor ← ReadStream on: # (Bob Dave Earl Frank)
```

Выражение	Результат
accessor next	Bob
accessor nextMatchFor:#Dave	true
accessor peek	Earl
accessor next	Earl
accessor peekFor:#Frank	true
accessor reset	accessor (cam)
accessor upTo:#Earl	(Bob Dave)
accessor skipTo:#Frank	true

Класс WriteStream является подклассом PositionableStream, предоставляющим доступ к записи элементов в группу. Этот класс не наследует сообщения next, next: do: из своих суперклассов. Он используется в ST как часть методов печати и записи строковых описаний любых объектов. Каждый объект в ST может отвечать на сообщения printOn:aStream и storeOn:aStream, определенные в классе Object. Методы, реализующие эти сообщения, состоят из последовательности сообщений к аргументу (класса Stream), которые заносят требуемые элементы описания в соответствующую группу, позиционированную потоком.

Для точного выражения записываемых символов в данном классе реализованы следующие операции:

WriteStream instance protocol
запись символов
cr — запись в получатель символа RETURN (ENTER, BK)
crTab — запись в получатель двух символов: ENTER и TAB
crTab: anInteger — запись в получатель символа ENTER и символов TAB в количестве, определяемом аргументом
space — запись в получатель символа пробела
tab — запись в получатель символа TAB

Подклассом класса WriteStream является класс ReaWriteStream, который наследует протоколы суперклассов, позволяющие читать и писать элементы в позиционированной группе.

В заключение данного параграфа рассмотрим описание реализации класса Random, являющегося подклассом класса Stream, реализующего второй способ позиционирования элементов группы (описание протокола см. в п. 3.8).

class name	Random
superclass	Stream
instance variable names	seed
class methods	

```
создание экземпляров
    new
        ^ self basicNew setSeed
instance methods
проверка
    atEnd
        ^ false
доступ
    next
        \temp\
        "линейный конгруэнтный метод Лемера (Lehmer) с модулем m=2 raisedTo: 16, a=27181 odd и 5=a\\8, c=13849 odd и c\m приблизительно 0.21132"

        [seed ← 13849+(27181*seed) bitAnd:8r177777
        temp ← seed/65536.0
        temp = 0] whileTrue
        ^ temp
приватный
    setSeed

        "для начального значения seed взять 16 младших битов системных часов"

        seed ← Time millisecondClockValue bitAnd: 8r177777
```

5.2. Внешние потоки

Класс Stream предполагает, что позиционируемая группа содержит произвольные объекты. Однако при работе с реальными устройствами ввода-вывода, для которых элементами потоков являются числа, строки, слова, байты, существенна также практическая организация хранения информации во внешней памяти, в частности в виде файлов.

Поэтому в ST рассматривается класс ExternalStream, являющийся подклассом класса ReadWriteStream и предусматривающий протокол операций доступа и позиционирования неоднородных элементов группы (символов, слов, чисел и др.). Подклассом этого класса является класс FileStream, экземпляры которого представляют возможности доступа к внешним файлам.

Непосредственное представление файловой системы ST-машины производится в классах File, FileDirectory и FilePage, где экземпляры класса FileDirectory обеспечивают манипуляции со справочниками файлов, экземпляры класса File являются файлами, а FilePage — записями или страницами файлов.

Предполагается, что пользовательская программа использует файлы и их записи не непосредственно, а через поток символов, определяемый связанным с файлом экземпляром класса FileStream.

5.3. Процессы

ST-машина допускает выполнение нескольких параллельных процессов, которые могут быть независимыми (асинхронными) и зависимыми (синхронизируемыми). Общее обеспечение механизма муль-

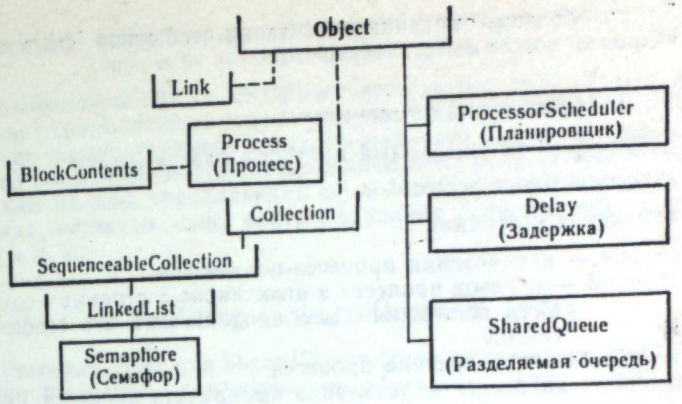


Рис. 11. Классы управления.

типроцессности производится специальными объектами классов Process, ProcessorScheduler, Semaphore, Delay, SharedQueue. Их структура показана на рис. 11.

Процесс представляет собой последовательность действий, описанных ST-выражениями и выполняемых ST-машиной. Ряд процессов управляют асинхронными устройствами, например клавиатурой, часами, манипуляторами и т. д., или контролируют доступ к ресурсам типа оперативной памяти.

Наиболее существенным процессом является исполнение выражений (сообщений), специфицированных непосредственно пользователем, и этот процесс в дальнейшем взаимодействует с другими, в том числе служебными, процессами.

Новый процесс создается путем посылки унарного сообщения fork блоку. Например:

```
[EasternTime display
MountainTime display
PacificTime display] fork    вызывает отображение восточного
                             альпийского и тихоокеанского времени
```

В отличие от сообщения блоку value сообщение fork создает новый независимый процесс вычисления содержимого блока параллельно процессу, пославшему сообщение fork.

Другим способом создания новых процессов является посылка блоку сообщения newProcess. В этом случае создается экземпляр процесса, но выражения блока-получателя не вычисляются, а результатом сообщения является сам блок. Этот вид процессов может быть поименован и называется неактивным. Активизация его осуществляется посылкой унарного сообщения resume соответствующему процессу. Например;

```
clockDisplayProcess ← [EasternTime display] newProcess
sortingProcess ← [alphabeticalList ← namelist sort]
newProcess
                создание двух процессов отображения времени
                и сортировки списка
sortingProcess resume
                активизация процесса сортировки
```

Таким образом, описание реализации сообщения fork в классе BlockContext может быть следующим:

```
fork
    self newProcess resume
```

Рассмотрим экземплярный протокол класса Process.

Process instance protocol

изменение состояния

resume — активизация процесса-получателя
suspend — перевод процесса в неактивное состояние с сохранением состояния для продолжения его сообщением resume
terminate — исключение процесса
priority: anInteger — установка приоритета процесса

Важным классом, обеспечивающим протоколы управления процессами, является класс ProcessorScheduler — собственный подкласс класса Object. Для однопроцессорной реализации предполагается один экземпляр этого класса, имеющий глобальное имя Processor.

ProcessorScheduler instance protocol

доступ

activePriority — получение приоритета текущего процесса
activeProcess — ответ — текущий процесс

изменение состояния

terminateActive — исключение текущего активного процесса
yield — выбор в качестве активного процесса, имеющего высший приоритет
highIOPriority — ответ — приоритет, необходимый наиболее критичному по времени процессу ввода-вывода
lowIOPriority — ответ — приоритет, при котором могут работать все процессы ввода-вывода
systemBackgroundPriority — ответ — приоритет для фонового системного процесса
timingPriority — ответ — приоритет для процесса, обслуживающего таймирование
userInterruptPriority — ответ — приоритет процесса пользователя, для которого необходима немедленная активизация

В общем случае процессы выполняются как последовательности независимых асинхронных действий. Основным механизмом синхронизации процессов являются объекты класса Semaphor, представляющего подкласс класса LinkedList.

Semaphor instance protocol

взаимодействия

signal — посылка через получателя сигнала некоторым ожидающим процессам, после чего активизируется процесс с максимальным временем ожидания.
wait — активный процесс приостанавливается до получения соответствующего сигнала

взаимное исключение

critical: aBlock — выполнение блока aBlock только в том случае, если не выполняются другие критические блоки

Экземпляры класса Semaphore используются также для обслуживания прерываний от аппаратуры виртуальной ST-машиной. Появление соответствующих сигналов вызывает возобновление (активацию) некоторых системных процессов, например обеспечивающих реакцию на ввод управляющих символов пользователем, обслуживающих системные часы, контролирующих размеры доступной памяти и т. п.

5.4. Классы SharedQuene и Delay

Экземпляры класса SharedQuene (разделяемая очередь) обеспечивают безопасные коммуникации объектов между разными процессами.

SharedQuene instance protocol

доступ

next — ответ — первый элемент очереди, а если очередь пуста, то активный процесс приостанавливается до появления в очереди первого объекта
nextPut: value — добавление в очередь-получатель значения, заданного аргументом value. Если процесс был приостановлен, то он активируется

Экземпляры класса Delay (задержка) позволяют приостанавливать процесс на заданное время.

Delay class protocol

создание экземпляров

forMilliseconds: millisecondCount — создание нового экземпляра, связанного с активным процессом. Когда будет послано сообщение wait (см. ниже), то он будет приостановлен на время (в миллисекундах), заданное аргументом
forSeconds: secondCount — то же, но задержка задается в секундах
untilMilliseconds: millisecondCount — приостановка активного процесса на заданный аргументом интервал в миллисекундах

общие запросы

millisecondClockValue — ответ — текущее значение миллисекундных часов

Delay instance protocol

доступ

resumptionTime — ответ — значение времени миллисекундных часов, в которое задержанный процесс был активирован

задержка процесса

wait — приостановка активного процесса до заданного получателем времени (см. forMilliseconds:)

Пример реализации простых секундных часов:

```
[[true]whileTrue:
  [Time now printString displayAt : 100@100
(Delay forSeconds: 1) wait]]fork
```

Глава 6

СРЕДСТВА ГРАФИКИ

6.1. Базовые классы графики

Общая схема получения графического изображения заключается в создании некоторого битового (для черно-белого изображения) образа кадра в памяти и последующей визуализации его на экране дисплея (или другом устройстве). Формирование кадра может проводиться некоторым набором базовых операций (примитивов).

В соответствии с этим в ST введены три группы классов, связанных с получением графических образов. Класс Form (форма) определяет некоторый буфер данных, которые должны быть визуализированы. Экземпляры класса Bitmap (битовое отображение) являются последовательностями развернутого битового изображения, интерпретируемого конкретными физическими устройствами. Основные операции над формами производятся специальными объектами класса BitBlt (передача битового растра).

На рис. 12 представлены основные классы графики и их подчиненность.

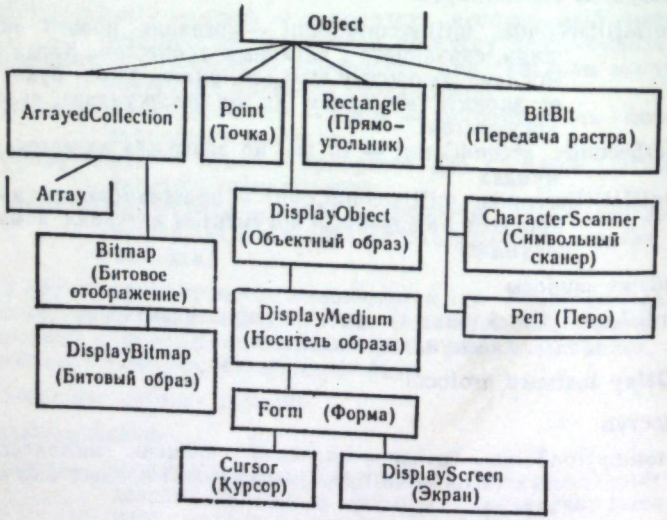


Рис. 12. Основные классы графики.

6.1.1. Класс Point

Экземпляры класса Point (точка) представляют пару числовых координат (x — y) для двухмерного изображения, указывающих местоположение графического элемента.

Объекты Point создаются обычно посылкой бинарного сообщения @ к числам. Например,

```
200 @ 150
```

является точкой с координатами x = 200 и y = 150. Кроме этого протокол класса Point определяет создание точек сообщениям x : xInteger y : yInteger. Например:

```
Point x : 200 y : 150
```

Point instance protocol

доступ

x — ответ — координата x
x: aNumber — установка координаты x, равной аргументу aNumber
y — ответ — координата y
y: aNumber — установка координаты y

сравнение

⟨aPoint — ответ — является ли получатель выше и левее аргумента
⟨= aPoint — ответ — не является ли получатель ниже или правее аргумента
⟩aPoint — ответ — является ли получатель ниже и правее аргумента
⟩= aPoint — ответ — не является ли получатель выше или левее аргумента
max: aPoint — ответ — юго-восточный угол прямоугольника диагонального, преобразуемого получателем и аргументом
min: aPoint — ответ — северо-западный угол аналогичного прямоугольника

Примеры.

Выражение	Результат
(175 @ 270) ⟩ (45 @ 230)	true
(45 @ 230) max: (175 @ 200)	175 @ 230
(45 @ 230) min: (175 @ 200)	45 @ 200

Point instance protocol

арифметика

*scale — новая точка, координаты которой являются произведением координат получателя и аргумента
+delta — новая точка, координаты которой являются суммой получателя и аргумента
-delta — то же, но определяются разностью получателя и аргумента
\scale — то же для деления
\\scale — то же для целочисленного деления с округлением в сторону минус бесконечность

abs — новая точка с координатами, равными абсолютному значению координат получателя

округление

rounded — новая точка с округленными координатами получателя

truncateTo: grid — новая точка, полученная путем усечения координат получателя с шагом дискретности, задаваемым получателем

Примеры.

Выражение	Результат
(45 @ 230) + (175 @ 300)	220 @ 530
(45 @ 230) + 175	220 @ 405
(160 @ 240) \ 50	(16 \ 5) @ (24 \ 5)
(160 @ 240) \\ 50	3 @ 4
((45@230)—(175@300)) abs	130 @ 70
(120. 5 @ 220. 7) rounded	121 @ 221
(160 @ 240) truncateTo:50	150 @ 200

Point instance protocol

точечные функции

dist: aPoint — ответ — расстояние между получателем и аргументом

dotProduct: aPoint — скалярное произведение получателя и аргумента

grid: aPoint — ответ —ближайшая к получателю точка в узлах решетки, образованной аргументом (округление)

transpose — новая точка, координата x которой равна координате y получателя, и наоборот

truncatedGrid: aPoint — то же, что и grid:, но с операцией усечения

Примеры.

Выражение	Результат
(160 @ 240) dotProduct: 50 @ 50	20000
(160 @ 240) grid: 50 @ 50	150 @ 250
(160 @ 240) truncatedGrid: 50 @ 50	150 @ 200

6.1.2. Класс Restangle

Экземпляры этого класса представляют собой прямоугольные области для графических элементов. Для удобства будем использовать следующие обозначения:

C — центр прямоугольника
TL (top left) — северо-западный угол
TC (top center) — середина верхнего угла
TR (top right) — северо-восточный угол
LC (left center) — середина левого ребра
RC (right center) — середина правого ребра

BL (bottom left) — юго-западный угол
BC (bottom center) — середина нижнего угла
BR (bottom right) — юго-восточный угол
W (width) — ширина
H (height) — высота
E (extent) — размер (ширина, высота)

В частности, в классе Point могут быть созданы прямоугольники — объекты класса Rectangle.

Point instance protocol

преобразование

corner: aPoint — создание прямоугольника с координатами TL, равными получателю, и BR, заданными аргументом

extent: aPoint — создание прямоугольника с координатами TL, равными получателю, и размерами E, заданными аргументом

Создание экземпляров класса Rectangle предусмотрено также следующим протоколом класса.

Rectangle class protocol

создание экземпляров

left: leftNumber right: rightNumber top: topNumber bottom: bottomNumber — создание прямоугольника с границами, заданными аргументами

origin: originPoint corner: cornerPoint — создание прямоугольника с TL — первым аргументом и BR — вторым аргументом

origin: originPoint extent: extentPoint — создание прямоугольника с TL — первым аргументом и E-вторым аргументом

Rectangle instance protocol

доступ

topLeft — точка TL
topCenter — точка TC
topRight — точка TR
rightCenter — точка RC
bottomRight — точка BR
bottomCenter — точка BC
bottomLeft — точка BL
leftCenter — точка LC
center — точка C
area — площадь получателя
width — ширина
height — высота
extent — пара E
top — верхнее ребро (координата y верхнего ребра)
right — правое ребро (координата x правого ребра)
bottom — нижнее ребро
left — левое ребро
origin — точка TL
corner — точка BR

Например,

frame ⟨— Rectangle origin 100 @ 100 extent: 150 @ 150

Тогда

Выражение	Результат
frame topLeft	100 @ 100
frame top	100
frame center	175 @ 175
frame area	22500

Rectangle instance protocol

доступ

origin: originPoint corner: cornerPoint — передача аргументам значений TL и BR точек получателя

origin: originPoint extent: extentPoint — передача аргументам значений TL (originPoint) и ширины и высоты получателя

amountToTranslateWithin: aRectangle — ответ — пара значений (aPoint) приращений, с помощью которых получатель может быть помещен внутрь прямоугольника, заданного аргументом

areasOutside: aRectangle — ответ — группа прямоугольников, заключающих часть получателя, не лежащую внутри аргумента

expandBy: delta — ответ — прямоугольник, сдвигаемый от получателя с помощью аргумента delta, который может быть прямоугольником, точкой или скаляром

insetBy: delta — то же, но сдвиг происходит в сторону уменьшения значений координат TL и BR

intersect: aRectangle — ответ — прямоугольник, образуемый пересечением получателя и аргумента

merge: aRectangle — ответ — минимальный прямоугольник включающий получатель и аргумент

Примеры.

A ⟨— 50 @ 50 corner: 200 @ 200
B ⟨— 120 @ 120 corner: 260 @ 240
C ⟨— 100 @ 300 corner: 300 @ 400

Выражение	Результат
A amountToTranslateWithin: C	50 @ 250
A areasOutside: B	OrderedCollection ((50 @ 50 corner: 200 @ 120) (50 @ 120 corner: 120 @ 200) 90 @ 290 corner; 310 @ 410 110 @ 320 corner: 290 @ 380 100 @ 120 corner: 300 @ 400
C expandBy: 10	
C insetBy: 10 @ 20	
B merge: C	

Rectangle instance protocol

проверка

contains: aRectangle — ответ — содержит ли получатель полностью все точки аргумента

containsPoint: aPoint — ответ — содержит ли получатель внутреннюю точку aPoint

intersects: aRectangle — ответ — пересекаются ли получатель и аргумент

округление

rounded — ответ — прямоугольник, TL и BR которого получены округлением к ближайшему целому соответствующих координат получателя

преобразования

moveBy:aPoint — сдвиг BR-точки таким образом, чтобы размер прямоугольника (ширина, высота) определялся аргументом

moveTo: aPoint — сдвиг прямоугольника таким образом, чтобы TL-точка совпала с аргументом

scaleBy: scale — масштабирование получателя с масштабным коэффициентом — аргументом. Ответ — новый прямоугольник

translateBy: factor — ответ — прямоугольник, полученный алгебраическим сложением текущих координат TL, BR с аргументом

Примеры.

Выражение	Результат
A moveBy: 50 @ 50	100 @ 100 corner: 250 @ 250
A moveTo: 200@300	200 @ 300 corner: 350 @ 450
A scaleBy: 2	400 @ 600 corner: 700 @ 900
A translateBy:—100	100 @ 200 corner: 250 @ 250

6.1.3. Классы Form и Bitmap

Произвольные образы, отображаемые на дисплее, представляются экземплярами класса Form (форма). Форма имеет высоту и ширину, а также битовую маску, создающую конкретное изображение чередованием черно-белых точек. Новая форма может быть получена путем комбинирования образов, содержащихся в других формах. Изображение может быть и обычным текстом, полученным комбинацией форм, содержащих изображение символов (букв, цифр и др.). Мультипликационные эффекты могут быть получены путем смены отображаемых форм, при этом во время отображения очередного статического кадра — формы, готовится другая форма — следующий кадр.

Form instance protocol

инициализация

fromDisplay: aRectangle — копирование битового (точечного) растра экрана дисплея внутри прямоугольника, определенного аргументом, в содержимое получателя

доступ

extent: aPoint — установка значений ширины и высоты получателя, равных аргументу

extent: extentPoint offset: offsetPoint — установка значений высоты и ширины получателя такими, чтобы они были координатами аргумента extentPoint, и смещения, определенного аргументом offsetPoint

образ

valueAt: aPoint— ответ — бит 0 или 1, содержащийся в точке aPoint получателя
valueAt: aPoint put: bitCode — запись в заданную первым аргументом точку получателя бита, заданного вторым аргументом (bitCode)

Form class protocol

создание экземпляров

fromDisplay: aRectangle — ответ — новая форма, чья битовая маска является копией экрана в границах, определенных аргументом

константы

black — ответ — форма для обозначения черного фона
darkGray — то же для темно-серого фона
gray — то же для серого фона
lightGray — то же для светло-серого фона
white — то же для белого фона

Изображение внутри формы образуется из элементов, которые являются экземплярами класса Bitmap. Bitmap учитывает физическую природу памяти и в современных реализациях предполагает 16-битовые слова. Каждая строка формы имеет связанный растр в виде целого, обозначающего количество слов, и соответствующих 16-битовых слов, хранящих изображение данной строки. Количество слов в строке называется размером растра.

6.1.4. Класс BitBlt

Экземпляры данного класса содержат переменные, необходимые для спецификации BitBlt (Bit Block Transfer — передача битового блока).

В операциях используются следующие обозначения параметров (аргументов):

destForm — форма, в которой формируется изображение
sourceForm — форма, из которой копируются элементы изображения
halftoneForm — форма, содержащая образцы полутонов
combinationRule — целое, специфицирующее правило комбинирования элементов изображения исходной S (source) и результирующей D (destination) форм:
1 : S AND D
3 : S
6 : S XOR D
7 : S OR D
destX, destY, width, height — характеристики (TL, W, H) прямоугольной зоны результирующей формы
clipX, clipY, clipWidth, clipHeight — координаты (TL, W, H) «вырезки» (окна) в результирующей форме, в области которой будут выполнены операции
sourceX, sourceY — координаты TL зоны исходной формы, из которой будут копироваться элементы изображения

BitBlt class protocol

создание экземпляра

destForm: destination sourceForm: source
halftoneForm: halftone combinationRule: rule
destOrigin: destOrigin sourceOrigin: sourceOrigin
extent: extent clipRect: clipRect — ответ — экземпляр BitBlt со значениями, установленными в соответствии с аргументами, где rule — целое; destination, source, halftone — формы; destOrigin, sourceOrigin и extent — точки, а clipRect — прямоугольник

BitBlt instance protocol

доступ

sourceForm: aForm — установка в качестве исходной формы получателя формы-аргумента
destForm: aForm — то же в отношении результирующей формы
mask: aForm — то же в отношении формы фона (halftone)
combinationRule: anInteger — установка номера правила комбинации изображения исходной и результирующей форм (0—15)
clipHeight: anInteger — установка высоты окна (вырезки) получателя
clipWidth: anInteger — установка ширины окна получателя
clipRect — ответ — прямоугольник-вырезка, установленный в получателе
clipRect: aRectangle — установка вырезки в получателе
clipxX: anInteger — установка координаты x для TL-точки вырезки получателя
clipY: anInteger — то же для координаты y
sourceRect: aRectangle — установка для исходной формы, связанной с получателем прямоугольной области, заданной аргументом
sourceOrigin: aPoint sourceX: anInteger sourceY: anInteger — то же для TL-точки
destRect: aRectangle — установка для результирующей формы, связанной с получателем прямоугольной области, заданной аргументом
destOrigin: aPoint destX: anInteger destY: anInteger height: anInteger width: anInteger — то же для TL-точки, высоты и ширины

копирование

copyBits — выполнение копирования установленной области битов из исходной формы в результирующую

построение линий

drawForm: startPoint to: stopPoint — построение прямой линии между заданными точками с продолжением визуализации исходной формы и режимов отображения

6.2. Класс Pen

Данный класс (ручка) является подклассом класса BitBlt и представляет некоторый достаточно высокий уровень доступа к построению изображений. Экземпляры этого класса также связаны с исходными и результирующими формами и правилами комбинации. Можно считать, что исходная форма является своеобразным инструментом письма, а результирующая форма представляет собой пишущую поверхность. Последняя при этом соответствует видимому на экране изображению.

Экземпляры класса Pen создаются сообщением new с начальной позицией в центре экрана в предположении, что исходная форма — одна черная точка и состояние направления «вверх».

Pen instance protocol

инициализация

defaultNib: shape — определение умолчания для «кончика пера» получателя Аргумент задает размеры соответствующей исходной формы

Пример.

bic ⟨— Pen new defaultNib : 2
создает экземпляр класса Pen в форме черного пятна размером 2× 2

Pen instance protocol

доступ

direction — ответ — текущее направление ручки (270 — вверх экрана)
location — текущее положение получателя
frame — ответ — прямоугольник, в котором получатель может рисовать
frame: aRectangle — установка прямоугольника, внутри которого получателю разрешается рисовать

Примеры.

Пусть экран имеет размеры 600 × 800 точек, а bic — экземпляр Pen, созданный сообщением предыдущего примера. Тогда

Выражение	Результат
bic direction	270
bic location	300 @ 400
bic frame: (50 @ 50 extent: 200 @ 200)	
bic location	300 @ 400

Pen instance protocol

перемещение

down — установка состояния получателя «вниз» с последующей отметкой следа при движении
up — установка состояния «вверх» и холостым (без отметок) последующим движением
turn: degrees — поворот на заданный аргументом угол
north — установка направления «вверх экрана»
go: distance — движение в текущем направлении на количество

битов, заданное аргументом с отметкой следа или без нее в зависимости от состояния (up, down)
goto: aPoint — перемещение в заданную точку (отметка следа в зависимости от состояния)
place: aPoint — переход в заданную точку без отметки
home — установка получателя в центр области, где допускается рисунок

Экземплярные протоколы данного класса используют в основном возможности и технику создания графических изображений по принципу рисующей «черепахи», представленной известным языком Logo.

6.3. Методы создания геометрических образов

Рассмотрим примеры создания геометрических фигур с помощью описанных ранее средств. Пусть необходимо нарисовать равносторонний многоугольник с n сторонами (nSides) и заданной длиной стороны (length). Сообщение

4 timesRepeat: [bic go : 100.bic turn : 90]

вызывает построение квадрата с длиной стороны 100 точек. Сообщение

nSides timesRepeat: [bic go : 100 bic turn : 360\\nSides]

вызовет построение nSide-угольника со стороной 100.

Опишем специальный класс Poligon, экземпляры которого могут рисовать n-угольники с заданной длиной стороны.

```
class name                  Poligon
superclass                  Object
instance variable names     drawinsPen
                            nSides
                            length

class methods

создание экземпляров
new
\ super new default

instance methods

рисование
draw
  drawingPen black
  nSides timesRepeat: [drawing-
  Pen go: length
  turn: 360\\nSides]

доступ
  length : l
    length ⟨— l
  sides : n
    nSides ⟨— n

приватные
  default
    drawingPen ⟨— Pen new
    self length : 100
    self sides : 4
```

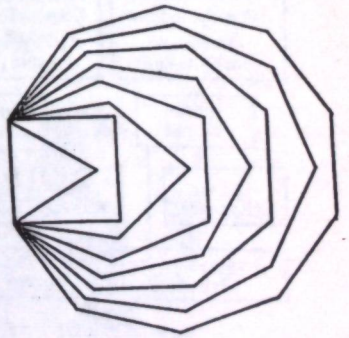


Рис. 13. Результат работы объекта класса Poligon.

Сообщения

poly ← Poligon new
3 to: 10 do: [:sides\poly sides : sides poly draw]
вызовут результат, показанный на рис. 13.

Глава 7

ОБЪЕКТЫ ОТОБРАЖЕНИЯ

Класс Form, рассмотренный в предыдущем разделе, представляет один из объектов отображения. Общая реализация отображения определяется иерархией подклассов класса DisplayObject. На рис. 14 представлена структура этих классов.

Объект отображения представляет собой образ, имеющий ширину и высоту и координаты северо-западного угла TL=0 @ 0. Указанные в других образах смещения отсчитываются от этой координаты. Имеются три следующих основных подкласса DisplayObject:

DisplayMedium (носитель отображения), представляющий тоновую раскраску элементов образа (или его прямоугольного фрагмента);

DisplayText (образ текста), представляющий текстовые фрагменты изображения;

Path (путь), представляющий композицию образов.

Класс Form является подклассом класса DisplayMedium. В нем образ дополняется своим битовым представлением, а для конструирования изображений обеспечивается исходная и результирующая информация.

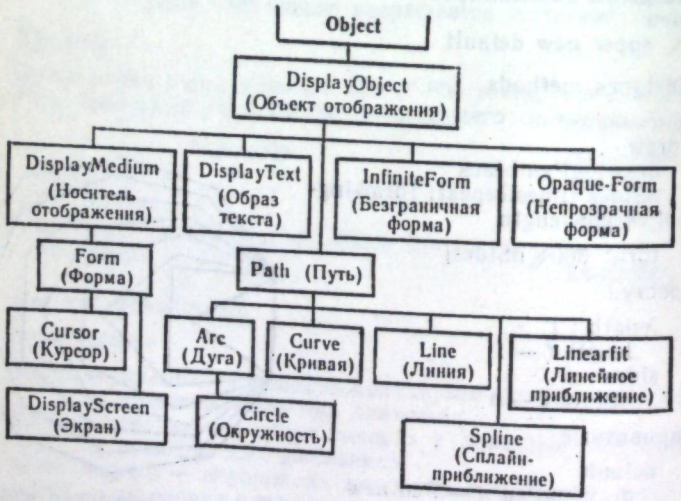


Рис. 14. Структура классов отображения

7.1. Класс Display Object

Рассмотрим основные протоколы данного класса.

DisplayObject instance protocol

доступ

width — ответ — ширина прямоугольника, представляющего границы получателя
heigth — ответ — высота получателя
extent — ответ — точка (a Point), содержащая ширину и высоту получателя
offset — ответ — точка (a Point), содержащая смещение получателя при его визуализации
offset: aPoint — установка смещения получателя по аргументу
rounded — установка смещения получателя путем целочисленного округления текущего смещения

преобразования

scaleBy: aPoint — масштабирование смещения получателя по аргументу
translateBy: aPoint — перемещение смещения получателя по аргументу

отображение

boundingBox — ответ — прямоугольная область, представляющая собой границы пространства для образа получателя
displayOn: aDisplayMedium at: aDisplayPoint
clippingBox: clipRectangle rule: ruleInteger mask: aForm — отображение получателя с точки aDisplayPoint с правилами комбинации тонов ruleInteger и маской допустимых тонов расцветки aForm. Отображается информация на пересечении получателя и аргумента clipRectangle
displayAt: aDisplayPoint — отображение получателя с точки, заданной аргументом аналогично предыдущему сообщению с параметрами области и раскраски по умолчанию
display — то же с точки 0 @ 0

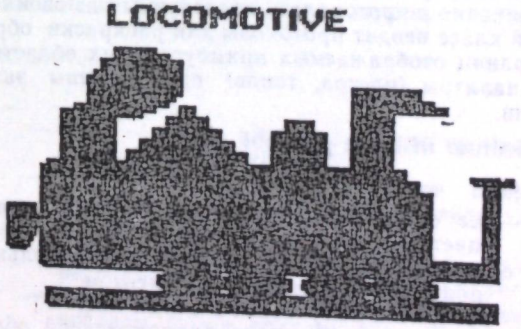


Рис. 15. Объект класса Form.

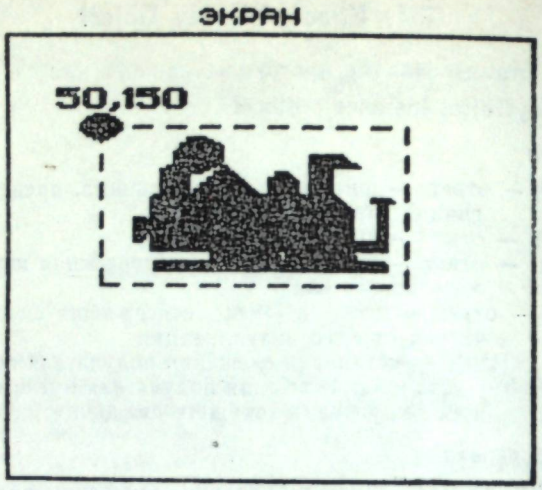


Рис. 16. Результат сообщения displayAt.

Последние два сообщения удачны для вывода текста (в графическом режиме). Например:

'это текст, который отображается' displayAt: 100 @ 100

Образ получателя будет выведен на экран с TL-точки 100 @ 100. Предполагается существование некоторых предварительных соглашений о размерах и стилях выводимого шрифта.

Пусть объект класса Form имеет имя locomotive и выглядит, как показано на рис. 15. Тогда сообщение

locomotive displayAt: 50 @ 150

вызовет размещение изображения на экране, как показано на рис. 16.

7.2. Класс DisplayMedium

DisplayMedium является подклассом класса DisplayObject и вводит объекты — носители некоторых образов.

В дополнение к протоколам, наследуемым из своих суперклассов, данный класс вводит протоколы для раскраски образов и размещения границ отображаемых прямоугольных областей. Соответствующие палитры (цветов, тонов) представлены экземплярами класса Form.

DisplayMedium instance protocol

раскраска

black — все области получателя раскрашиваются в черный цвет
black: aRectangle — черная закраска прямоугольной области получателя, заданной аргументом
white — раскраска получателя в белый цвет
white: aRectangle — раскраска прямоугольной области получателя в белый цвет

gray
gray: aRectangle — то же для серого цвета
lightGray
lightGray: aRectangle — то же для светло-серого цвета
darkGray
darkGray: aRectangle — то же для темно-серого цвета

Аналогично можно дополнить протоколы раскраски другими цветами и палитрами.

DisplayMedium instance protocol

заполнение

fill: aRectangle mask: aHalftoneForm — изменение области получателя, заданной первым аргументом путем заполнения ее 16 × 16 битовыми образами, содержащимися в экземпляре класса Form, заданном вторым аргументом
fill: aRectangle rule: anInteger mask: aHalftoneForm — то же, но вторым аргументом задается правило комбинации копирования маски (см. ниже)

Под правилами комбинации понимаются комбинаторные возможности переноса изображения от исходной формы к отображаемой, кратко введенные в п. 6.1.4. Будем рассматривать возможности черно-белых изображений и учитывать сложившуюся традицию знакогенерации в виде 16 × 16 битового растрового представления символов (в общем случае не обязательно букв или цифр). Экземпляр класса Form для подобной формы знакоместа (16 × 16) называется маской.

Таким образом, при построении образа с учетом комбинирования с существующим могут использоваться 4 элемента: исходная форма (Source), маска (Halftone mask), существующий образ (Destination) и результирующий образ (Result), что для черно-белого варианта точек дает 16 комбинаций, которые могут кодироваться целыми в диапазоне 0—15. Эти целые используются в протоколах под названием правил комбинаций.

При использовании цветной графики комбинаторные возможности значительно увеличиваются и удобно вводить специальные классы для представления комбинаций смешивания цветов. Например, для удобства представления определенных правил комбинирования протокол класса Form содержит следующие категории:

Form class protocol

правила

erase — ответ — целое, соответствующее правилу очистки получателя (белый цвет)
over — ответ — целое, соответствующее правилу, при котором черные области получателя заменяются содержимым маски
reverse — ответ — целое, соответствующее правилу инвертирования битов получателя (позитив-негатив)
under — ответ — целое, соответствующее правилу, при котором белые области получателя заменяются содержимым маски

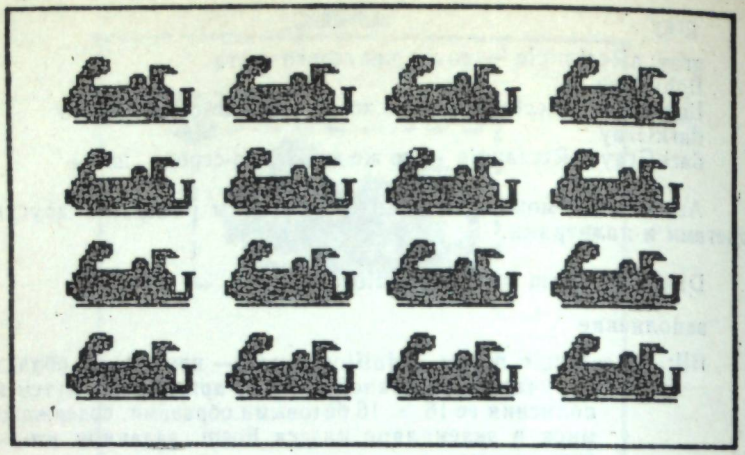


Рис. 17. Результат сообщения fill: box mask: locomotive.

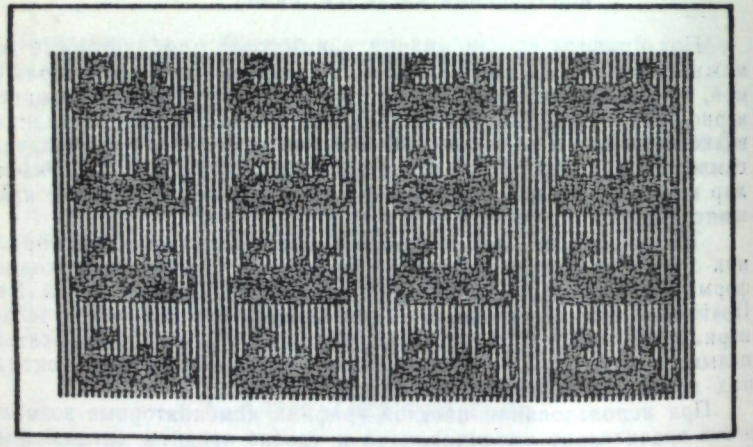


Рис. 18. Результат сообщения fill:box rule:Form mask: locomotive.

В качестве примера рассмотрим следующие сообщения в предположении, что приведенный на рис. 15 образ представляет маску

```
box <— 16 @ 16 extent 64 @ 64
picture fill : box mask : locomotive
```

Результат вычисления будет выглядеть, как показано на рис. 17. Результат вычисления следующих двух сообщений показан на рис. 18.

```
picture lightGray : box
picture fill : box rule : Form under mask : locomotive
```

DisplayMedium instance protocol

инвертирование

reverse: aRectangle mask: aHalftoneForm — изменение битов области получателя, заданной первым аргументом таким образом, что инвертированию подлежат только биты, для которых биты маски равны 1 (черные)

reverse: aRectangle — инвертирование битов изображения получателя в области, заданной аргументом

reverse — инвертирование битов получателя

выделение границ

border: aRectangle withRectangle: borderWidth mask: aHalftoneForm — граница области получателя, ограниченная первым аргументом, закрашивается полосой с шириной, заданной вторым аргументом в цвета, определяемые третьим аргументом

В заключение укажем два подкласса класса Form: класс Cursor (курсор) и DisplayScreen (экран дисплея).

Класс Cursor порождает экземпляры графического изображения курсора и включает протоколы визуализации и перемещения курсора.

Класс DisplayScreen обычно содержит один экземпляр объектов, соответствующий полному экрану. Протокол класса включает операции начальной установки размеров экрана. В некоторых случаях реализации полноэкранной мультипликации для совмещения процессов заполнения нового кадра и развертки текущего может использоваться несколько экземпляров этого класса.

7.3. Методы отображения

В общем случае практически используемые графические отображения формируются как комбинации нарисованных графических образов и текстов в различных шрифтах.

В ST рассматриваются следующие классы, обеспечивающие типичные элементы изображения.

Класс DisplayText (отображение текста) является подклассом класса DisplayObject. Экземпляры этого класса связывают объекты класса Text (см. п. 4.1) и объекты класса TextStyle. TextStyle связывает индекс шрифта, обеспечиваемого протоколами класса Text, с конкретными множествами графического представления символов шрифтов.

Класс Path (траектории) является также подклассом класса DisplayObject. Он представляет собой упорядоченную группу (OrderedCollection) точек (Points) и форму (Form), которая визуализируется в указанных точках. Этот класс является суперклассом классов характерных графических образов, показывающих различные геометрические траектории.

Path instance protocol

доступ

form — текущая форма, связанная с получателем

form: aForm — установка новой формы для получателя

at: index — точка (Point) в упорядоченной группе, связанной с получателем с заданным индексом

at: index put: aPoint — запись новой точки

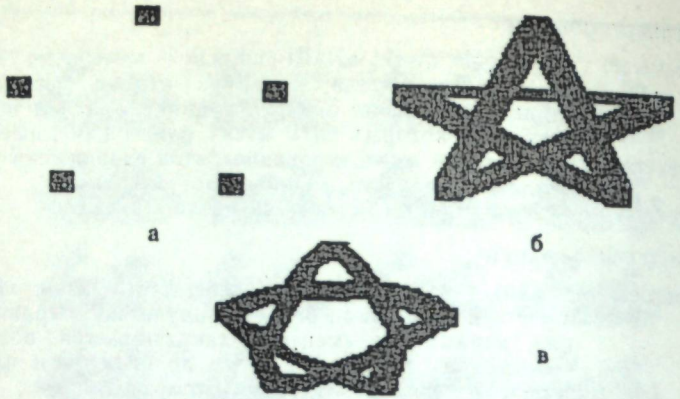


Рис. 19. Пример изображений.

size — количество точек в связанной с получателем упорядоченной группе

проверка

isEmpty — ответ — является ли пустой связанная с получателем группа

изменение группы

add: aPoint — добавление точки в качестве последнего элемента
removeAllSuchThat: aBlock — вычисление блока-аргумента для каждой из точек, связанной с получателем группы. Удаление тех точек, для которых значение блока равно true

перечисление

do: aBlock — вычисление аргумента-блока для каждой точки получателя
collect: aBlock — вычисление аргумента-блока для каждой точки. Результаты вычисления образуют новую группу
select: aBlock — вычисление аргумента-блока для каждой точки получателя. Результат — новая группа из тех точек получателя, для которых вычисление блока дает true

В качестве примера рассмотрим следующие сообщения:

```
aPath ← Path new form: dot
aPath add: 150 @ 285
aPath add: 400 @ 285
aPath add: 185 @ 430
aPath add: 280 @ 200
aPath add: 375 @ 430
aPath add: 150 @ 285
aPath display
```

На рис. 19 представлены три изображения, соответствующие экземпляру класса Path, созданному вышеприведенной последовательностью сообщений для маски dot (19, *а*), экземпляру класса LineFit, использующему ту же группу точек (19, *б*), и экземпляру класса Spline, использующему ту же группу точек (19, *в*).

Отметим, что образы произвольной сложности удобно строить путем применения хорошо классифицированных операций композиции образов, а также операций преобразования существующих образов. Средства ST позволяют удобно группировать эти операции в соответствующих классах. Так, например, в классе Form реализованы операции масштабированного увеличения формы (magnity: aRectangle by:scale), поворота (rotate), раскраски области (shapeFill:aMask interiorPoint: interiorPoint) и ряд других.

Глава 8

ПРОТОКОЛ ДЛЯ КЛАССОВ

Универсальность концепции объекта как экземпляра некоторого класса ST вызывает определенные трудности рекурсивного характера при рассмотрении классов как экземпляров некоторых метаклассов. В ST предложены следующие варианты решений:

каждый класс является подклассом класса Object, за исключением самого класса Object, не имеющего своего суперкласса;

каждый объект является экземпляром класса;

каждый класс является экземпляром метакласса.

Как уже упоминалось в п. 1.6, метакласс не имеет собственного имени, как другие классы, а ссылка на него производится путем посылки унарного сообщения class к экземпляру метакласса (т. е. к самому классу). Например, на метакласс класса Collection ссылаются следующим сообщением:

```
Collection class
```

Метакласс имеет только один экземпляр объектов — описываемый класс — и порождается автоматически при создании нового класса, при этом методы, относящиеся к категориям class methods (в отличие от instance methods), введенные в описание реализации, являются методами, реализованными в соответствующем метаклассе.

Определенная группа классов введена для того, чтобы обеспечить ввод новых классов. Это классы Behavior (поведение), его подкласс ClassDescription (описание класса) и подклассы последнего Class (класс) и MetaClass (метакласс).

8.1. Класс Behavior

Класс Behavior определяет информацию, необходимую для интерпретатора ST, и включает описание существующей в системе иерархии классов, словаря методов, экземпляров в кодированных терминах и представление их переменных.

Протокол для класса Behavior разделен на четыре категории: создание, доступ, проверка и перечисление. Общая структура методов следующая:

1. Создание: создание словаря методов, создание экземпляров, создание иерархии классов.

2. Доступ: доступ к содержимому словаря методов, доступ к экземплярам и переменным (экземплярным, класса, пула), доступ к иерархии классов.

3. Проверка: проверка содержимого словаря методов, проверка формы экземпляров, проверка иерархии классов.
4. Перечисление: перечисление подклассов и экземпляров.

Behavior instance protocol

создание словаря методов

methodDictionary: aDictionary — запись аргумента в качестве словаря методов получателя
addSelector: selector withMethod: compiledMethod — добавление в словарь методов пары: селектор, заданный первым аргументом (объект класса Symbol), метод, заданный вторым аргументом
removeSelector: selector — удаление из словаря методов метода, ассоциированного с заданным селектором
compile: code — компиляция метода с записью объектного кода в переменные получателя
compile: code notifyings: requestor — компиляция метода с записью результата в словарь методов получателя. В случае ошибки — посылка подходящего сообщения второму аргументу
recompile: selector — компиляция метода, ассоциированного с заданным селектором
decompile: selector — декомпиляция метода (получение исходного текста) для метода, ассоциированного с селектором
compileAll — компиляция всех методов, представленных в словаре методов получателя
compileAllSubclasses — компиляция всех методов, представленных в словаре методов подклассов получателя

создание экземпляров

new — создание нового экземпляра получателя без индексных переменных
new: anInteger — создание нового экземпляра получателя с индексными переменными, число которых задано аргументом

создание иерархии классов

superclass: aClass — установка в качестве суперкласса получателя аргумента
addSubclass: aClass — установка аргумента в качестве подкласса получателя
removeSubclass: aClass — удаление аргумента из списка подклассов получателя

доступ к словарю методов

selectors — ответ — множество селекторов сообщений, специфицированных в локальном словаре методов получателя
allSelectors — ответ — множество селекторов сообщений, специфицированных в словарях методов получателя и его суперклассов
compiledMethodAt: selector — ответ — компилированный метод, ассоциированный с заданным аргументом, селектором в локальном словаре методов получателя

sourceCodeAt: selector — ответ — исходный текст (String) аналогично предыдущему сообщению
sourceMethodAt: selector — ответ — исходный текст (Text) аналогично предыдущему сообщению

доступ к экземплярам и переменным

allInstances — ответ — множество (Set) всех прямых экземпляров получателя
someInstance — ответ — существующий экземпляр получателя
instanceCount — число текущих экземпляров получателя
instVarNames — ответ — массив (Array) имен экземплярных переменных, специфицированных в получателе
subclassInstVarNames — ответ — множество (Set) имен экземплярных переменных в подклассах получателя
allInstVarNames — ответ — массив (Array) имен экземплярных переменных получателя и его суперклассов
classVarNames — ответ — множество (Set) имен переменных класса, специфицированных для получателя
allClassVarNames — ответ — множество (Set) имен переменных класса получателя и его суперклассов
sharedPools — ответ — множество (Set) имен пулов (словарей), которые специфицированы для получателя
allSharedPools — то же, но включая суперклассы получателя

Примеры.

Выражение	Результат
OrderedCollection instVarNames	('firstindex' 'lastindex')
OrderedCollection SubclassInstVarNames	Set ('sortBlock')
SortedCollecrion allInstVarNames	('firstindex' 'lastindex' 'sortBlock)
String classVarNames	Set (StringBlter)
String allClassVarNames	Set(StringBlter DependentsFields ErrorRecursion)

Протоколы доступа к иерархии классов включают унарные сообщения для получения подклассов и суперклассов получателя. Например, сообщение

String superclass

вызовет результат ArrayedCollection, а сообщение

ArrayedCollection subclasses

вызовет результат Set (Array ByteArray RunArray Bitmap String Text).

Протоколы проверки включают семь сообщений контроля словаря методов, семь сообщений проверки типов экземплярных переменных и два сообщения проверки иерархии классов.

Выражение	Результат
OrderedCollection includesSelectot # addFirst SortedCollection	true

```
wichClassIncludesSelector:# size     OrderedCollection
OrderedCollection instSize           2
String inheritsFrom:Collection       true
```

Протокол перечисления включает шесть сообщений, которые определяют по некоторым правилам список объектов. Элементы этого списка могут быть аргументами вычисленного блока.

Behavior instance protocol

перечисление

allSubclassesDo: aBlock — выполнение блока для каждого подкласса получателя
allSuperclassesDo: aBlock — то же для суперклассов получателя
allInstancesDo: aBlock — то же для каждого текущего экземпляра получателя-класса
allSubInstancesDo: aBlock — то же для текущих экземпляров подклассов получателя
selectSubclasses: aBlock — ответ — множество (Set) тех подклассов получателя, для которых вычисление блока дает true
selectSuperclasses: aBlock — то же для суперклассов получателя

Например, для понимания поведения объектов — экземпляров групповых классов — полезно знать, в каком из подклассов класса Collection реализовано сообщение добавления элементов allFirst: Это позволит протрассировать последовательность операций и выяснить, какой метод действительно обработал данное сообщение. Следующие выражения определят в переменной subs классы, реализующие указанное сообщение:

```
subs ⟨— Set new
Collection allSubclassesDo:
[:class/(class includesSelector:# addFirst:)
ifTrue: [subs add:class]]
```

8.2. Класс ClassDescription

Этот класс является подклассом класса Behavior и определяет имена классов, комментарии классов и имена экземплярных переменных. По общему правилу он наследует методы своих суперклассов и имеет дополнительный протокол, уточняющий и структурирующий некоторые вопросы физической реализации ST.

ClassDescription instance protocol

доступ к описанию класса

name — ответ — имя получателя (String)
comment — ответ — комментарий к получателю (String)
comment: aString — задание комментария к получателю по аргументу
addInstVarName: aString — добавление аргумента в качестве одной из экземплярных переменных получателя
removeInstVarName: aString — удаление имени экземплярной переменной, заданного аргументом

организация сообщений и классов

category — ответ — категория системной организации для получателя
category: aString — назначение получателю системной категории, заданной аргументом
removeCategory: aString — удаление заданной аргументом категории и всех сообщений в этой категории
whichCategoryIncludesSelector: selector — поиск категории, которой принадлежит заданный аргументом метод в словаре методов получателя

сохранение в файлах

fileOutOn: aFileStream — запись описания получателя в файл, доступный аргументу
fileOutCategory: categoryName — создание файла с именем получателя и стандартным расширением .st и запись в него описаний сообщений, категоризированных под именем, заданным аргументом
fileOutChangedMessages: setOfChanges on: aFileStream — первый аргумент является группой пар класс — сообщение, которые были изменены. Запись описаний каждой пары в файл, доступный второму аргументу.

8.3. Класс Metaclass

Основная роль данного класса, являющегося собственным подклассом класса ClassDescription, заключается в обеспечении протокола инициализации переменных класса и создания одного инициализированного экземпляра метаклассов.

Metaclass class protocol

создание экземпляров

subclassOf: superMeta — ответ — экземпляр класса Metaclass, который является подклассом метакласса, заданного аргументом
name: newName environment: aSystemDictionary
subclassOf: superClass
instanceVariableNames: stringOfInstVarNames
variable: variableBoolean words: wordBoolean
pointers: pointerBoolean
classVariableNames: stringOfClassVarNames
poolDictionaries: stringOfPoolNames category: categoryName
comment: commentString changed: changed — создание полностью инициализированного класса

Некоторые особенности подобного описания продиктованы технологией графического интерфейса для пользователя, создающего новые классы.

8.4. Класс Class

Этот класс также является собственным подклассом класса ClassDescription, и его экземпляры описывают представление и поведение объектов.

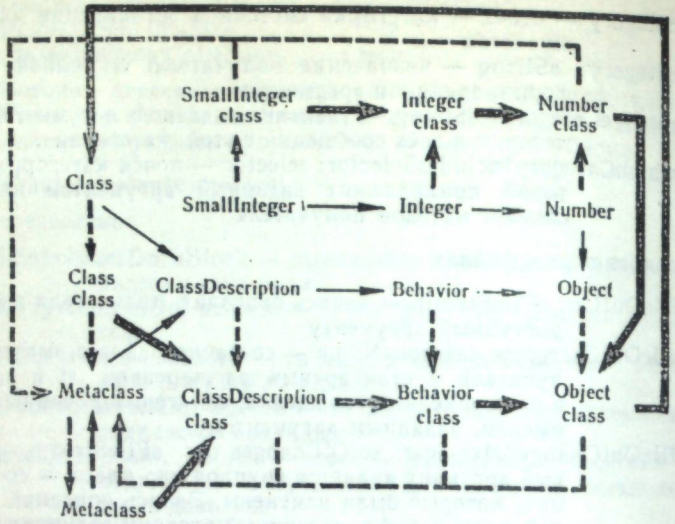


Рис. 20. Взаимосвязь классов и их иерархий.

Class instance protocol

доступ к экземплярам и переменным

addClassVarName: aString — добавление аргумента в качестве переменной класса получателя
removeClassVarName: aString — удаление переменной класса, заданной аргументом
addSharedPool: aDictionary — добавление аргумента в качестве пула разделяемых переменных
removeSharedPool: aDictionary — удаление пула, заданного аргументом
classPool — ответ — словарь переменных класса получателя
initialize — инициализация переменных класса

создание экземпляров

subclass: classNameString
instanceVariableNames: stringInstVarNames
classVariableNames: stringOfClassVarNames
poolDictionaries: stringOfPoolNames
category: categoryNameString — создание нового класса, подкласса получателя, имеющего соответствующие имена элементов, заданных аргументами

В заключение определим взаимосвязь всех рассмотренных классов. К первым трем положениям, указанным в начале главы, добавим следующие:

все метаклассы являются подклассами класса Class;

каждый метакласс является экземпляром класса Metaclass;

методы класса Class и его суперклассов поддерживают поведение, общее для объектов-классов;

методы экземпляров класса Metaclass добавляют поддержку поведения, специфичного для конкретных классов.

На рис. 20 показана взаимосвязь классов и их иерархий класс—суперкласс и класс — метакласс. Тонкими линиями показано отношение класс — суперкласс для классов, жирными линиями — отношение класс — суперкласс для метаклассов, а прерывистой линией — отношение класс — объект, экземпляр класса.

Глава 9

МЕТОДЫ РЕАЛИЗАЦИИ

9.1. Интерфейс пользователя

Как уже указывалось ранее, в ST-машине были применены новые принципы многооконного диалогового интерфейса пользователя. Здесь рассмотрим практическую реализацию подобного интерфейса в системе, известной под названием Smalltalk-2 или Methods, для персональных ЭВМ типа IPC XT [17].

Непосредственно после загрузки системы на экране дисплея появляются три окна: окно системной консоли (фирменная этикетка), рабочее окно и окно системного браузера. Рабочее окно служит для формирования сообщений к объектам реализованных классов. Можно создать несколько рабочих окон. Окно системного браузера служит для создания новых или модификации существующих классов. Окна могут иметь внутренние разбиения на подокна, которые назовем фреймами (в оригинале pane). Например, окно системного браузера первоначально содержит три фрейма.

Для управления могут использоваться либо клавиатура, либо манипулятор типа «мышь». В случае клавиатуры определенные клавиши создают так называемое ручное управление. Мигающая черта (подчеркивание) на экране задает курсор, перемещение которого осуществляется клавишами управления курсором (горизонтальные и вертикальные стрелки). Основными методами управления ST-машиной являются группы команд, объединенные в меню. Меню появляются в специальных окнах и определяются текущим местонахождением курсора. Вызов меню производится путем нажатия клавиш Ins (меню окна) или Del (меню фрейма). Если курсор находится вне окон, то нажатие одной (любой) из названных клавиш вызывает системное меню. Внутри меню выбор требуемой команды (функции) производится путем инверсной подсветки соответствующей команды, которую можно сдвигать по вертикали клавишами управления курсором с последующим нажатием клавиши выбора (удлиненная клавиша +). Выход из меню осуществляется сдвигом курсора влево. На рис. 21 приведено назначение основных управляющих клавиш, а на рис. 22 — первоначальный (после загрузки системной копии) вид экрана дисплея.

Перемещение курсора в область окна вызывает активизацию окна. При этом метка окна меняет подсветку, все окно выдвигается на передний план, и его содержимое становится доступным.

Для быстрого перемещения курсора по окнам либо фреймам внутри окна можно использовать функциональные клавиши F9, F10. Если текст, визуализируемый в окнах либо фреймах, больше, чем размер окна, то его видимость достигается путем перемещения (скролинга) содержимого в окнах как по горизонтали, так и по вертикали.

Рассмотрим основные меню системы. Системное меню вызывается клавишей Ins или Del, когда курсор находится вне окон.

Рис. 21. Управляющие клавиши

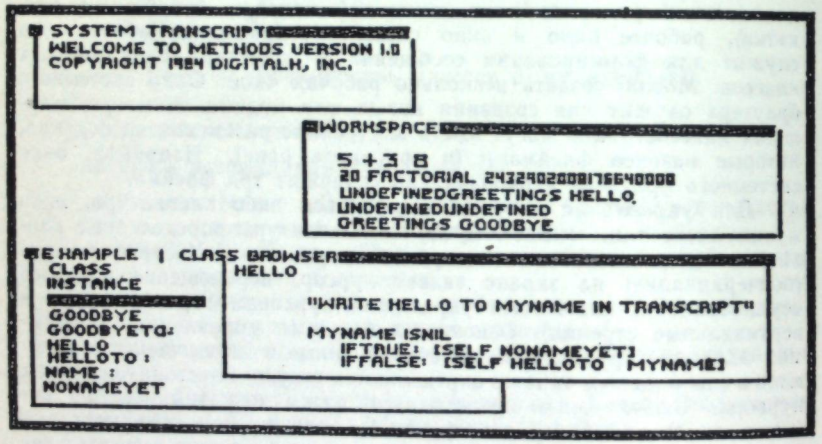


Рис. 22. Начальный вид экрана.

Оно содержит следующие команды: color screen (изменение цвета экрана); exit Methods (выход из системы); browse disk (просмотр диска); open Workspace (открыть рабочее окно); browse classes (просмотр классов); redraw screen (перепись экрана); save image (сохранение имиджа).

Назначение команд в целом понятно из их названия. Следует отметить, что под имиджем здесь понимается состояние ST-машины, которое может меняться при изменении состава и содержания классов (см. п. 9.2). При выполнении команд могут появляться дополнительные окна меню для параметров команд, а также окна, в которых помещается результат выполнения, например справочник файлов на диске.

Меню окна вызывается клавишей Ins, когда курсор находится в каком-либо окне и содержит следующие команды манипуляции с окнами: color (цвет); collapse (разрушение); under (под окном); frame (размер окна); move (переместить); close (закрыть); label (метка).

С помощью этих команд можно изменять цвет окна, убрать окно с сохранением содержимого (collapse), сделать активным окно, лежащее под данным (under), изменить размер окна и место размещения (frame, move), исключить окно (close), а также пометить окно некоторым символьным именем, которое располагается в левом верхнем углу окна (label).

Для большинства окон доступен единый текстовый редактор, с помощью которого вводятся и модифицируются требуемые тексты, например наименование окна (метка), сообщение в рабочем окне или описание класса.

Кроме традиционных методов и средств редактирования текстов, среди которых есть выделение некоторого фрагмента текста клавишей расширенного выбора (клавиша —), с текстовыми окнами (точнее, фреймами) связано также специальное меню редактирования, вызываемое нажатием клавиши Del. Это меню включает следующие команды: copy (копировать); cut (вырезать); paste (склеить); save (сохранить); restore (восстановить).

В этом же меню имеются также две команды show it (показать) и do it (выполнить), которые используются для вычисления заданных выражений.

Общая технология работы с ST-машиной, таким образом, заключается в задании требуемых окон, в которых создаются тексты необходимых выражений. Затем требуемое выражение либо последовательность выражений выделяются как фрагмент, после чего с помощью команд фреймового меню (show it, do it) выражения вычисляются с визуализацией (show it) соответствующих результатов. Наличие общего текстового редактора позволяет легко исправлять ошибки в некорректных выражениях, а также переносить фрагменты в разные окна, формируя некоторую собственную среду вычислений.

Несколько сложнее является изменение ST-машины, связанное с разработкой новых классов, поскольку при аналогичных действиях по редактированию текстов описаний методов требуются тщательно спланированные действия по отладке новых классов. При этом доступ к описанию метода осуществляется через окна браузера по основной иерархии фреймов системного браузера: фрейм типов методов (класс или экземпляр класса), фрейм селекторов методов выбранного типа и, далее, фрейм описания метода.

Для модификации существующих классов удобно открыть отдельное окно системного браузера для данного класса путем посылки унарного сообщения требуемому классу с селектором edit, например

```
Example edit
```

Далее, с помощью команд меню окна можно установить его удобные размеры и расположение и затем, последовательно переводя курсор по трем внутренним фреймам окна с выбором соответствующих наименований методов (селекторов), вызвать текст описания метода, к которому применяются обычные действия по редактированию.

В общем случае для создания и модификации методов следует использовать команду browse classes системного меню, по которой создается окно, показывающее иерархию всех классов. В этом случае в окне браузера будет четыре фрейма: один — для иерархического перечисления всех классов, второй — для типа метода (класса или экземпляра класса), третий — для селекторов методов, выбранных в предыдущих фреймах, и четвертый — для описания выбранного таким образом метода.

Для манипуляций с описаниями методов предусмотрено специальное меню фрейма списка селекторов со следующими командами: remove (удалить); new method (новый метод); implementors (суперклассы, содержащие выбранное имя селектора); senders (классы, ссылающиеся на выбранное имя селектора).

В практической реализации в интерфейс пользователя входят также разнообразные средства диагностики и отладки описания методов, построенные аналогичным образом и обеспечивающие надежную работу с системой.

9.2. Компилятор

С точки зрения реализации ST-машины следует различать понятия виртуальной машины и виртуального имиджа (image).

Виртуальный имидж состоит из всех объектов, существующих в системе. Виртуальная машина включает аппаратуру и программный интерпретатор некоторого промежуточного кода, исполнение которого обеспечивает заданную динамику объектов. Можно сказать, что виртуальный имидж загружается в виртуальную машину, после чего ST-машина становится некоторым единым интерактивным средством, поведение которого было описано выше.

Общая технология получения ST-машины включает описание методов, т. е. собственно программирование. Эти методы транслируются компилятором в последовательность 8-битовых команд, называемых байткодами. Байткоды являются командами программного интерпретатора, работающего с двумя основными моделями памяти: стеком методов и памятью для объектов. Память для объектов фактически образует виртуальный имидж. Нижний уровень ST-машины образует аппаратные компоненты ЭВМ, обеспечивающие конечные эксплуатационные характеристики.

Подобная архитектура ST-машины предусмотрена для традиционных ЭВМ, в частности для ПЭВМ класса IBM PC XT. Имеются сообщения об аппаратной реализации ST-машины, в которых описываемые ниже компоненты могут быть сконструированы иначе.

Рассмотрим основные вопросы реализации компилятора.

Исходные методы, написанные программистом, представлены в ST-машине как экземпляры класса String. Эти экземпляры содержат последовательность символов, удовлетворяющих синтаксису описания метода.

Компилятор транслирует описания в последовательность байткодов интерпретатора. Например, для класса Rectangle унарное сообщение для поиска точки (Point) в центре прямоугольника имеет следующую реализацию:

```
center
    ^origin+corner/2
```

Этот исходный текст будет преобразован компилятором в последовательность байткодов вида

0, 1, 176, 119, 185, 124

которые имеют следующий смысл при интерпретации:

0 — поместить значение первой экземплярной переменной получателя (origin) в стек интерпретатора
1 — поместить значения второй экземплярной переменной получателя (corner) в стек

176 — послать бинарное сообщение с селектором +
119 — поместить 2 (объект класса SmallInteger) в стек
185 — послать бинарное сообщение с селектором /
124 — возвратить объект, находящийся в верхушке стека, в качестве значения сообщения (center)

Упомянутый в комментариях к байткодам стек интерпретатора является основным механизмом упорядоченной безадресной передачи основных компонентов сообщения, т. е. получателя и аргументов, а также возвращаемого значения, и подробнее будет описан в п. 9.3.

Последовательности протранслированных байткодов для ST-машины представляют экземпляры класса CompiledMethod.

Особенность реализации компилятора заключается в том, что пользователь (программист) не взаимодействует с компилятором непосредственно.

Когда к существующему классу добавляется новый (модифицируется существующий) метод, то класс (например, Rectangle) запрашивает у компилятора соответствующий экземпляр класса CompiledMethod. При этом класс обеспечивает компилятор некоторой необходимой информацией, не содержащейся непосредственно во вводимом пользователем описании метода (имена переменных и т. п.). После создания необходимого экземпляра CompiledMethod класс помещает информацию о методе в специальный словарь методов, с помощью которого осуществляется поиск метода при интерпретации сообщения.

При компиляции метода (т. е. создании экземпляра класса CompiledMethod) не все элементы исходного описания, включая и дополнительную информацию для компилятора, передаваемую соответствующим классом, могут иметь прямые эквиваленты в байткодах. Поэтому экземпляр CompiledMethod может иметь также некоторую дополнительную информацию, называемую литеральным фреймом, которую можно сравнить с понятием внешних ссылок в компиляторах традиционных языков программирования.

Категориями объектов, которые могут транслироваться прямо в байткоды, являются:

получатель и аргументы вызываемого сообщения;
значения экземплярных переменных получателя;
значения временных переменных метода;
7 специальных констант (true, false, nil, —1, 0, 1, 2);
32 специальных селектора сообщений +, —, <, >, <=, >=, =, ~=, *, /, \, @, bitShift:, \\, bitAnd:, bitOr:, at:, at:put:, size, next, nextPut:, atEnd, ==, class, blockCopy:, value, value:, do:, new, new:, x, y.

Перечисленные специальные селекторы соответствуют встроенным методам базовой ST-машины.

Все объекты, содержащиеся в CompiledMethod, не относящиеся к указанным выше категориям, представляются в экземпляре CompiledMethod в литеральных фреймах, в частности:

разделяемые переменные (глобальные, класса или пула); большинство литеральных констант (числа, строки, массивы и т. д.); большинство селекторов сообщений.

Байткоды, имеющие объекты литерального фрейма, ссылаются на них; при этом, если компилятор обнаруживает два идентичных объекта, относящихся к литеральному фрейму, то в последний помещается только одна копия, а ссылки соответствующих байткодов корректируются на этот один экземпляр.

Рассмотрим пример компиляции еще одного метода для экземпляров класса Rectangle, иллюстрирующего сказанное.

```
intersects: aRectangle
    ^(origin max: aRectangle origin)<
    (corner min: aRectangle corner)
```

Соответствующий экземпляр CompiledMethod будет иметь следующую структуру.

Байткоды:

0 — поместить значения первой экземплярной переменной (origin) в стек
16 — поместить в стек аргумент (aRectangle)
209 — послать унарное сообщение с селектором, находящимся во второй позиции литерального фрейма (origin)
224 — послать сообщение с селектором, находящимся в первой позиции литерального фрейма (max:) с одним аргументом
1 — поместить значение второй экземплярной переменной (corner) в стек
16 — поместить в стек аргумент (aRectangle)
211 — послать унарное сообщение с селектором, находящимся в четвертой позиции литерального фрейма (corner)
226 — послать сообщение с селектором, находящимся в третьей позиции литерального фрейма (min:) с одним аргументом
178 — послать бинарное сообщение с селектором <
124 — возвратить объект, находящийся в верхушке стека, в качестве значения сообщения (intersects:)

Литеральный фрейм
max:
origin
min:
corner

Система команд для интерпретатора, используемая компилятором, включает 256 байткодов, разделенных на следующие категории: размещение в стеке; связь с переменными; посылка сообщения; возврат; переходы.

Байткоды размещения помещают в верхушку стека следующие объекты:

получатель сообщения, который вызывается CompiledMethod,
экземплярные переменные получателя,
временные переменные и формальные аргументы метода, литеральный фрейм соответствующего экземпляра CompiledMethod,
верхушка стека (т. е. коды, дублирующие содержимое верхушки стека).

Байткоды связи с переменными генерируются компилятором, если сообщение компилируемого метода имеет присваивание. Байткоды могут ссылаться на экземплярные переменные, временные переменные метода или разделяемые переменные. При этом байткоды могут сохранять объекты в верхушке стека либо удалять их.

Байткоды посылки сообщений специфицируют селекторы сообщений и количество аргументов к ним. Структура стека формируется таким образом, что в нем получатель находится под аргументами.

Байткоды возврата индицируют завершение выполнения метода, при этом возвращаемое значение, как правило, представляет объект, находящийся в верхушке стека.

Байткоды перехода представляют собой средства традиционных безусловного и условного переходов внутри последовательности байткодов, интерпретируемых в общем случае последовательно. Они, в частности, используются при компиляции конструкций типа

```
ifTrue: [    ]
ifFalse: [    ]
```

9.3. Интерпретатор

Интерпретатор выполняет байткоды для текущего экземпляра CompiledMethod. Интерпретатор как вычислительная машина характеризуется следующим состоянием:

экземпляр CompiledMethod, байткоды которого интерпретируются,
указатель следующей команды (байткода),
получатель и аргументы сообщения, вызванные CompiledMethod,
временные переменные, необходимые CompiledMethod, стек.

Общий цикл работы интерпретатора состоит из трех фаз:

выборка байткода (по значению указателя),
установка нового значения указателя байткода,
выполнение функции, определенной выбранным байткодом.

Команды размещения в стеке, связи с переменными и переходов фактически оперируют только со стеком и указателем, в то время как команды посылки сообщений и возврата изменяют состояние, поскольку требуют перехода к интерпретации другого экземпляра CompiledMethod. Сохранение состояния интерпретатора предусмотрено в объектах, названных контекстом (context). Организация стековой памяти для контекстов обеспечивает правильную интерпретацию цепочек сообщений, включая рекурсии.

Когда встречается байткод посылки сообщений, интерпретатор выполняет следующие действия:

находит получатель сообщения (под аргументами в стеке),
получает доступ к словарю методов класса получателя,
находит требуемый элемент словаря по заданному селектору метода,
если селектор найден, переходит к интерпретации ассоциированного метода,
если селектор не найден, то путем возврата к третьему шагу проводит поиск в словарях соответствующих суперклассов; при этом отсутствие селектора в суперклассах приводит к сообщению об ошибке.

Приведенный алгоритм учитывает также наличие некоторого множества встроенных первоначальных классов, методы которых относят к примитивам системы. При реализации множество примитивов должно обладать минимальным составом, обеспечивающим создание новых классов.

9.4. Память объектов

Память объектов представляет собой самостоятельный компонент реализации и обеспечивает взаимодействие интерпретатора с объектами. Каждый объект идентифицируется уникальным кодом, называемым указателем объекта. Для традиционных 16-битовых слов мини- и микроЭВМ количество представимых объектов

не должно превышать 65536. В объектной памяти указатель объектов ассоциирован с некоторым множеством обязательных и необязательных указателей на другие объекты. Каждый указатель объекта должен быть ассоциирован, по крайней мере, с указателем объекта класса, к которому принадлежит данный объект.

Память объектов обеспечивает следующие пять основных функций для интерпретатора:

доступ к значению экземплярной переменной объекта;
изменение значения экземплярной переменной объекта;
доступ к классу объекта;
создание нового объекта;
вычисление количества экземплярных переменных объекта.

Функция удаления объекта явно не задается, поскольку объект, ссылки на который не существует, автоматически удаляется.

В качестве базовой среды встроенных объектов принято представление целых чисел — объектов класса SmallInteger, для которых отсутствует указатель класса, а сами указатели объектов (коды чисел от —16384 до 16384) являются значениями.

Объектная память построена по принципу heap-памяти, в которой практически не возникает проблема «сборки мусора», характерная для отображения динамически организуемых данных на линейные структуры памяти.

9.5. Аппаратное обеспечение

В общем случае принципы построения ST-машины допускают ее реализацию на машинах традиционных архитектур, в частности на современных ПЭВМ, имеющих основной 16-разрядный процессор и математический сопроцессор, оперативную память объемом 640 Кбайт, стандартную клавиатуру и фиксированный диск (типа Винчестер) объемом 10—30 Мбайт.

Для полного использования классов графики удобно иметь цветной графический монитор и устройство точечной печати. Основные манипуляции удобно проводить с помощью манипуляторов типа «мышь» с тремя инициирующими прерывания кнопками. Для работы с классами времени и даты необходимы также миллисекундный таймер и системные часы с автономным питанием.

Весьма интересные идеи развития ST связаны с реализацией распределенной обработки данных, включая почти фантастические возможности распределенно «живущих» объектов, мастерски описанных Тцикридзисом в [15].

Ввиду того что посылка автономных объектов в другие вычислительные установки требует адекватной среды поддержки их интерпретации, в ближайшее время аппаратной поддержкой подобных распределенных систем, по-видимому, могут являться только однородные локальные вычислительные сети.

Перспектива погружения виртуальной ST-машины в аппаратуру также является, по-видимому, делом недалекого будущего, хотя основным тормозом может оказаться несовместимость с существующими стандартами на виды и типы реализации программного обеспечения ЭВМ.

ПРИЛОЖЕНИЕ

Словарь основных терминов dBASE III Plus (Clipper)

Алиас (alias) — имя рабочей зоны как альтернативное имя (псевдоним) файла данных (отношения), размещенного в ней. По умолчанию, алиасы обозначаются литерами от A до J для рабочих зон 1—10 соответственно.

ASCII — акроним для американского стандартного кода обмена информацией в ЭВМ (American Standard Code for Information Interchange).

Атрибут (attribute) — характеристика объекта. Для файла данных в реляционной модели соответствует полю записи (описывающего некоторую характеристику объекта, заданного файлом—отношением). Употребляется также для характеристик состояния физических устройств ПЭВМ (например, интенсивность отображения на экране дисплея).

База данных (database) — один или несколько файлов данных (.dbf), содержащих записи взаимосвязанных данных. Может включать также такие дополнительные файлы, как индексные (.ndx), форматные файлы отчетов (.frm), запросы (.qry) и др., используемые одним или несколькими приложениями.

Выражение (expression) — последовательность констант, имен переменных памяти, полей записей файлов данных и функций одинаковых типов (кроме типа даты), связанных знаками соответствующих операций, используемых для получения некоторых вычисляемых значений.

Виртуальное поле (или запись) — поле или запись, физически не существующие, но доступные для обработки. Эти поля (записи) физически формируются в момент запроса со стороны прикладной программы на основании некоторых заданных выражений над реальными данными.

Виртуальный (virtual) — описатель некоторого объекта, характеризующий логическую доступность его в операциях обработки при отсутствии прямого физического эквивалента в схеме хранения.

Длина поля (field width) — максимальное количество символов (байт), размещаемых в поле.

Замаскированные переменные (hidden variable) —переменные памяти с описателем PUBLIC, которые становятся временно недоступными в подпрограммах при использовании команды PRIVATE.

Запись (record) — группа взаимосвязанных полей информации, рассматриваемых прикладной программой как нечто целое. Соответствует понятию кортежа отношения реляционной модели данных.

Индексный файл (index file) — специальный файл (.ndx файл), содержащий указатели на записи файла данных (.dbf файл) в по-

рядке возрастания некоторого ключа. Ключ задается как выражение над полями файла данных.

История (history) — сохранение до 20 последних введенных диалоговых команд в буферной памяти с возможностью их вызова и повторного исполнения (возможно, с редактированием).

Каталог (catalog) — специальный класс файлов, используемых для регистрации совокупности файлов, образующих пользовательскую базу данных, за исключением командных файлов(.prg), текстовых файлов (.txt) и файлов — копий переменных памяти (.mem).

Команда (command) — основная конструкция входного языка, специфицирующая законченное действие над данными или устанавливающая состояние исполнительной среды.

Координата (coordinate) — число, указывающее на позицию строки или столбца экрана, используемое для указания точки начала визуализации информации.

Курсор (cursor) — видимый маркер, обозначающий текущую строку экрана.

Макро (macro) — слово или символ, заменяющие некоторый текст на языке dBASe III Plus. В качестве макропеременной, хранящей текст подстановки (замены), используется переменная памяти символьного типа. Подстановка осуществляется функцией &. Для Clipper текст подстановки не должен включать ключевых слов, используемых для обозначения команд и опций.

Обновление (update) — модификация базы данных операциями добавления, удаления или изменения данных.

Опция (option) — языковая конструкция, входящая в состав команды и указывающая на определенный выбор режима исполнения команды.

Переменная памяти (memory variable) — поименованный элемент памяти, в котором можно хранить значения данных (промежуточных результатов). Допускаются такие же типы значений, как и для полей записей, за исключением текстовых данных.

Поле (field) — минимальный поименованный элемент записи, соответствующий понятию атрибут реляционной модели данных.

Приложение (application) — программа, написанная на входном языке dBASE (Clipper) для выполнения некоторой задачи пользователя.

Рабочая область (work area) — области (до десяти) оперативной памяти, в которых размещаются доступные записи файла данных. Пронумерованы числами от 1 до 0, именуются символами от A до J, являющимися альтернативными именами связанных файлов данных (см. Алиас).

Расширение (extension) — дополнение к основному имени файла в стандартах операционной системы ПЭВМ, записываемое с помощью точки и следующих за ней символов (до трех). Используется для спецификации типа файла в различных прикладных программах (например, ⟨имя файла⟩. dbf).

Список полей (field list) — последовательность имен полей, разделенных запятой.

Тип поля (field type) — характеристика типа данных, сохраняемых в поле (кодируется при формировании файла данных: C — символьные строки, N — числа, D — даты, L — логические значения, M — тексты).

Тип файла (file type) — спецификация назначения файла, кодируемая с помощью расширения (см.).

Условие (condition) — логическое выражение, используемое в качестве дополнительной конструкции в ряде команд (FOR ⟨condition⟩), применение которого заключается в выполнении команды только тогда, когда значение логического выражения истинно.

Файл (file) — совокупность связанной информации, сохраняемой в качестве поименованной единицы хранения на устройствах внешней памяти.

Файл дампа памяти (memory file) — файл (.mem файл), используемый для сохранения текущего состояния переменных памяти.

Файл текстовых типов данных (data base text file) — вспомогательный файл (.dbt файл), используемый для хранения значений текстовых полей файла данных. В последнем в mem-полях хранятся указатели на соответствующие элементы .dbt файла.

Файл данных (data base file) — файл (.dbf файл), используемый в качестве основной структуры хранения данных, организованных в виде совокупности однородных записей. Соответствует понятию отношения в реляционной модели данных.

Файлы экранных форм (screen form files) — служебные файлы (.scr, .fmt файлы), содержащие информацию о размещении на экране дисплея зон ввода-вывода информации для конкретного приложения.

Файл формата отчета (format file) — служебный файл (.frm файл), содержащий информацию о формате вывода информации в виде отчета, имеющего заголовок — наименование, шапку наименований колонок, итоговые (подытоговые) строки, разбиение на страницы и т. п.

Файл формы запроса (query file) — служебный файл (.qry файл), содержащий информацию о горизонтальном фильтре (см. Фильтр).

Файл виртуальной базы данных (view file) — специальный файл (.vue файл), содержащий информацию о совокупности используемых в данном приложении взаимосвязанных файлов данных и индексных файлов, горизонтальных и вертикальных фильтров над ними, экранных форм и форм отчетов.

Фильтр (filter) — горизонтальный фильтр представляет логическое выражение, используемое для манипуляций только с теми записями, которые удовлетворяют условию, заданному выражением. Вертикальный фильтр представляет (неполный) список полей файла данных, к которым разрешается доступ во время действия фильтра (другое название — селектор).

Словарь основных терминов Knowledgeman

Вариация (variance) — среднее значение квадратичного отклонения от математического ожидания в частотном распределении.

График (graph) — визуальное представление данных на экране или печатающем устройстве.

Запись (record) — строка таблицы СУБД, содержащая по одному значению для каждого поля.

Импорт (import) — перемещение данных из внешних файлов в таблицы СУБД.

Индекс (index) — упорядоченный список, содержащий по одному входу для каждой записи таблицы СУБД и предназначенный для быстрого поиска данных в БД.

Максимум (maximum) — наибольшее значение некоторого поля.

Меню (menu) — список режимов работы пакета, выводимый на экран.

Минимум (minimum) — наименьшее значение некоторого поля.

Область команд (command area) — область в нижней части экрана, предназначенная для отображения команд, построенных с помощью выбора режимов меню.

Область меню (menu area) — область в левой части экрана, предназначенная для отображения режимов текущего меню.

Область сообщений (message area) — двухстрочная область экрана предназначенная для отображения сообщений пакета.

Операнд (operand) — число, переменная памяти, числовая функция или поле.

Определение ячейки (cell definition) — значение данных или числовое выражение (формула), определяющее значение ячейки.

Подсказка (prompt) — сообщение, отображаемое в области сообщений, на которое в строке ввода нужно ввести ответ.

Поле (field) — имя категории данных, хранящихся в таблице СУБД.

Свертка слов (wordwrap) — процесс автоматического переноса слов в следующую строку текста для выравнивания правого поля.

Сессия (session) — период работы с пакетом.

Словарь (vocabulary) — набор слов пакета, определенных пользователем в качестве группы полей, числового выражения, команды и т. д.

Служебная область (utility area) — область в правой части экрана, отображающая помощь, описание управляющих клавиш, специальные меню.

Среднее (average) — результат деления суммы значений некоторого поля на количество значений этого поля.

Стандартное отклонение (standard deviation) — корень квадратный средних квадратичных отклонений от математического ожидания в частотном распределении.

Страница помощи (help page) — информация, описывающая текущий режим, выбранный с помощью последовательности меню.

Строка ввода (entry line) — строка на экране, предназначенная для ввода ответа пользователя на сообщения пакета.

Строковая константа (string constant) — константа, состоящая из букв, цифр или их комбинации.

Счетчик (count) — количество наблюдений данных.

Файл ASCII (ASCII ile) — файл, в котором представленные в стандартном ASCII формате данные разделяются запятыми или пробелами.

Файл DIF (DIF file) — файл, в котором данные представлены в стандартном формате обмена данными (Data Interchange Format)

Установка (setting) — параметр, значение которого может меняться во время сессии.

Числовая константа (numeric constant) — целое или десятичное число.

Экспорт (export) — перемещение данных из таблиц СУБД во внешние файлы.

Электронная таблица (spreadsheet) — таблица в памяти ЭВМ, состоящая из ячеек, значения которых могут задаваться и/или программироваться пользователем.

Ячейка (cell) — пересечение строки и столбца электронной таблицы.

Абстрактный класс (abstract class) — класс, который специфицирует протокол, но полностью его не реализует. По соглашению экземпляры данного класса не создаются.

Аргумент блока (block argument) — параметр, который должен быть заменен значением при вычислении блока (посылка сообщения value:).

Бинарное сообщение (binary message) — сообщение с одним аргументом, селектор которого составлен из одного или двух специальных символов.

Блок (block) — описание отложенной последовательности действий.

Временная переменная (temporary variable) — переменная, создаваемая для специальных действий и доступная только в течение выполнения этих действий.

Выражение (expression) — последовательность символов, описывающая объект.

Глобальная переменная (global variable) — переменная, разделяемая всеми экземплярами всех классов.

Имя аргумента (argument name) — имя псевдопеременной, доступной методу только на время его исполнения; значение имени аргумента — фактический аргумент сообщения, вызывающего исполнение данного метода.

Имя переменной (variable name) — идентификатор, ссылающийся на текущее значение переменной.

Имя псевдопеременной (psevdo-variable name) — выражение, подобное имени переменной, однако в отличие от последнего значение его не может быть изменено присваиванием.

Каскадирование (cascading) — описание нескольких сообщений к объекту в одном выражении.

Категория сообщения (message category) — группа методов в описании класса.

Класс (class) — описание группы подобных объектов.

Ключевое слово (keyword) — идентификатор, заканчивающийся двоеточием.

Литерал (literal) — выражение, описывающее объект-константу: число,строку, массив, . . .

Массив (array) — группа объектов, элементы которой ассоциированы с целочисленными индексами.

Метакласс (metaclass) — класс, порождающий объект — другой класс.

Метод (method) — описание выполнения одной операции объекта.

Образец сообщения (message pattern) — селектор сообщения и множество имен аргументов, каждое для каждого из аргументов, которые должны быть в сообщении, задаваемом этим селектором.

Объект (object) — компонент ST-80, представленный некоторой приватной памятью и множеством операций.

Описание протокола (protocol description) — описание класса в терминах протокола его экземплярных сообщений.

Описание реализации (implementation description) — описание класса в терминах его экземплярной приватной памяти и множества методов, описывающих выполнение экземпляров объектов своих операций.

Переменная класса (class variable) — переменная, разделяемая всеми экземплярами простого класса.

Перемещение метода (overriding a method) — спецификация метода в подклассе для тех же сообщений, что и метод в суперклассе.
Подкласс (subclass) — класс, наследующий переменные и методы из существующего класса.
Получатель (receiver) — объект, которому послано сообщение.
Примитив (primitive method) — операция, выполняемая ST-машиной.
Присваивание (assigment) — выражение, описывающее изменение значения переменной.
Селектор сообщения (message selector) — имя типа операции, которую сообщение требует от получателя.
Символ (symbol) — строка, последовательность знаков которой отличается от любого другого символа.
Системные классы (system classes) — множество классов, представляемых ST-80-системой.
Сообщение (message) — запрос к объекту на выполнение одной из его операций.
Сообщение с ключевым словом (keyword message) — сообщение с одним или более аргументами, селектор которого представлен одним или более ключевыми словами.
Сопряжение (interface) — сообщения, на которые объект может ответить.
Суперкласс (superclass) — класс, из которого наследуются переменные и методы.
Унарное сообщение (unary message) — сообщение без аргументов.
Экземпляр (instance) — один из объектов, описанных классом.
Экземплярная переменная (instance variable) — часть приватной памяти объекта.

СПИСОК ЛИТЕРАТУРЫ

1. *Disk* Operating System, Verssion 3. 10// Referense. IBM Personal Computer Software. — 1985. — 400 p.
2. *Microsoft* C Compiler for the MS DOS Operating System User's Guide. — Microsoft Corporation, 1986. — 444 p.
3. *TurboPascal* Version 3.0 Reference Manual. — Borland International, 1985. — 376 p.
4. *Grafix* Partner, Version 1.8 Graphics Processing Software. — Brightbill Roberts, 1985. — 32 p.
5. *PC Paintbrush*. — Zsoft Corporation, 1985. — 72 p.
6. *dBASE* III Plus: Learning and Using. — Ashton Tate, 1986. — 817 p.
7. *Clipper* dBASE III Compiler. — Nantucket Corporation, 1985. — 247 p.
8. *Knowledgeman-2*: User's Guide. — Micro Data Base Systems Inc, 1985. — 710 p.
9. *Framework* II. — Ashton Tate, 1986.
10. *Smalltalk*-80. The Language und its Implementation/ A. Goldberg, D. Robson. — Palo Alto: Xerox Corporation, 1980. — 714 p.
11. *Дал У. И., Мюрхауг Б., Нюгорд К.* Симула 67. Универсальный язык программирования — М.: Мир, 1969 — 99 с.
12. *Liskov B., Snyder A., Atrinson R., Schaffert C.* Abstraction mechanisms in CLU//Comm. ACM. — 1977. — 20, N 8. — P. 564—576.
13. *Ichbia D. et al.* Preliminary ADA Reference Manual/SIGPLAN Notices. — 1979. — 14, N6.
14. *Wirth N.* MODULA: a language for modular multiprogramming, Software: Practice and Exp. — 1977. — 7, N 1.
15. *Office* Automation Concepts and Tools. — Berlin. — 399 p.
16. *Персональные* ЭВМ//ТИИЭР. — 1984. — 72, N 3. — 189 с.
17. *Methods* V1.0 Digitalk Inc. — 1984.

УКАЗАТЕЛЬ ИНДЕКСОВ КОМАНД ВХОДНОГО ЯЗЫКА dBASE III PLUS, CLIPPER

УКАЗАТЕЛЬ ИНДЕКСОВ КОМАНД ВХОДНОГО ЯЗЫКА KNOWLEDGEMAN

Справочное издание

Стогний Анатолий Александрович
Ананьевский Сергей Анатольевич
Барсук Яков Израилевич и др.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПЕРСОНАЛЬНЫХ ЭВМ

Справочное пособие

Оформление художника *Г. М. Балюна*
Художественный редактор *А. В. Косяк*
Технический редактор *И. А. Ратнер*
Корректоры *Л. И. Ноур, В. Н. Божок, Л. Г. Гузиашвили*

ИБ № 9763

Сдано в набор 29.07.88. Подп. в печ. 21.03.89. БФ 00020.
Формат 84 × 108/32. Бум. тип. № 1. Лит. гарн. Выс. печ.
Усл. печ. л. 19,32. Усл. кр.-отт. 19,32. Уч.-изд. л. 26,95.
Тираж 75000 экз. Зак. 8-827. Цена 1 р. 70 к.

Издательство «Наукова думка». 252601 Киев 4, ул. Репина, 3.

Книжная ф-ка им. М. В. Фрунзе. 310057 Харьков,
ул. Донец-Захаржевского, 6/8.

Ярина

«Лелеча думка»