КАЗАХСКО - АМЕРИКАНСКИЙ УНИВЕРСИТЕТ

С.Б. Дубовиченко

WEB ПРОГРАММИРОВАНИЕ

Часть I. Основы языка PHP 4

Алматы
2004

Казахско - Американский Университет

С.Б. Дубовиченко

WEB ПРОГРАММИРОВАНИЕ

Часть I. Основы языка PHP 4

Учебник по информатике для ВУЗов

Алматы
2004

УДК 378(073.8): 681.14
ББК
Д 79

Печатается по решению Научно - методического
совета Казахско - Американского Университета

Рецензенты

заведующий кафедрой естественных наук КАУ, академик международной академии информатизации, доктор физико - математических наук, профессор **Чечин Л.М.**,
доцент факультета прикладных наук КАУ, кандидат физико - математических наук **Байзылдаева У.Б.**

Дубовиченко С.Б.

Д 79 Web - программирование. Часть 1. Основы языка PHP 4: Учебник по информатике для ВУЗов. - Алматы: Изд. КАУ, 2004г. -240с.

ISBN

Настоящий учебник предназначен для студентов ВУЗов, изучающих информатику и посвящен некоторым вопросам Web - программирования. Имеющиеся, в данной области издания, которых явно не достаточно, не всегда ориентированы на начинающих пользователей и бывают сравнительно трудны для начального изучения основ Web - программирования.

Настоящая книга преследует цель несколько упростить и систематизировать изложение материала по одному из основных языков Web - программирования PHP версии 4, который постоянно используется в Интернете для обработки данных, получаемых с HTML форм, парольного доступа, сохранения данных пользователей и т.д.

Книга может быть использована любыми желающими, изучить возможности Интернета, языка Web - программирования PHP и создание HTML страниц с его использованием.

Д $\frac{060700000}{00(05)-03}$ ББК

ISBN

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ 4
СКРИПТЫ....7
История 7
Возможности....8
Структура скрипта....9
Выполнение PHP скриптов....10
СИНТАКСИС 13
Форма записи....13
Комментарии 13
Переменные 14
Операторы равенства 16
Кавычки....17
ПЕРЕМЕННЫЕ 19
Типы переменных....20
Примеры переменных 23
Присвоение значения 24
Логические переменные 26
Преобразование типов 28
Область действия переменных....34
Изменяемые переменные....38
Константы 39
ВЫРАЖЕНИЯ 43
Простые выражения 43
Инкремент и декремент 45
Выражения сравнения....46
Логические выражения 47
Совмещенные выражения....48
Строковые выражения 49
МАССИВЫ 53
Простые массивы....54
Ассоциированные массивы 56
Многомерные массивы 57
Работа с массивами 57
ОПЕРАТОРЫ....65
Операторы присваивания 65
Строковые операции 66
Арифметические операторы....67
Операторы сравнения....68
Инкремент и декремент 70

Логические операторы 71
Битовые операции 73
Операции эквивалентности 73
КОНСТРУКЦИИ 76
Условная конструкция if 76
Условная инструкция switch 81
Цикл while 84
Цикл do... while 86
Цикл for 87
Дополнительные операторы 89
ДАТА И ВРЕМЯ 94
СТРОКИ 97
Функции trim() 97
Функции обрезки строк 97
Функции замены и перекодировка 99
Функция перекодировки 101
Функция поиска в строке 101
ФУНКЦИИ 105
Пользовательские функции 105
Область переменных в функции 108
Аргументы по умолчанию 111
Аргументы и ссылки 112
Встроенные функции 113
ФОРМЫ 121
Элементы формы 122
Одностраничная форма 128
Многостраничные формы 135
Пример формы: Калькулятор 138
ГРАФИКА 144
Структура рисунка 144
Рисование линий 146
Рисование дуг 147
Рисование прямоугольников 148
Рисование многоугольников 149
Задание прозрачности цвета 151
ФАЙЛЫ И ДИРЕКТОРИИ 155
Включение файлов в документ 155
Проверка существования файла 157
Открытие и закрытие файла 159
Чтение и запись в файл 161
Работа с директориями 167

Работа с WEB документами 172
СЕССИИ, ЗАГОЛОВКИ И COOKIES 173
Сессии 173
Пример сессии 176
Заголовки 182
PHP и Cookie 184
ОБЪЕКТНО - ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ 189
Принципы ООП 189
Классы 190
Наследование классов 191
ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ 194
Переменные сервера 194
Предопределенные PHP переменные 196
Передача параметров 200
Опция register_globals 202
Новый метод передачи переменных 205
ЛИТЕРАТУРА 213
ПРИЛОЖЕНИЕ 214
Регулярные выражения 214
Регистрация сайта в поисковых системах 223
Аутентификация 224
Система голосования 237

СКРИПТЫ

Язык Web - программирования PHP (препроцессор гипертекста) - это скрипт - язык (Scripting Language), встраиваемый в HTML страницу, интерпретируемый и выполняемый непосредственно на Web - сервере перед выдачей этой страницы браузеру. Код программы, помещенный в теги <? и ?>, не передается браузеру, а выполняется непосредственно на стороне сервера. Браузеру выдается только то, что выводят команды выдачи на экран, например, echo.

Основное отличие от CGI скриптов, написанных на других языках, типа Perl или C, состоит в том, что в CGI программах вы сами пишете выводимый на экран HTML код, а, используя язык PHP, вы встраиваете свою PHP программу в готовую HTML страницу, используя, открывающие и закрывающие теги.

Язык PHP очень похож на ASP от фирмы Microsoft (MS Active Server Page), но приспособлен к Unix подобным системам и чаще всего употребляется с Web - сервером Apache, хотя может работать и с MS IIS (Internet information Server), и, в принципе, с любым другим Web - сервером.

Основное преимущество PHP - это простота, гибкость и скорость его выполнения. Еще одним преимуществом PHP, является то, что он создан специально для того, чтобы работать вместе с базами данных типа Apache + PHP + SQL, в частности, с базой данных MySQL, которая существует и для системы Linux и для Win9x/NT/2000/XP.

История

Язык PHP был задуман примерно в конце 1994 года Расмусом Р. Ледорфом (Rasmus Lerdorf). Ранние, не выпущенные версии PHP, использовались на его домашней странице для того чтобы следить, кто просматривал его интерактивное резюме. Первая используемая версия PHP стала доступна в начале 1995 и была известна, как Personal Home Page Tools. Она состояла из очень упрощенного синтаксического анализатора, который понимал только несколько специальных макрокоманд и ряд утилит.

Довольно трудно дать какую - либо подробную статистику применения PHP, но отмечено, что к 1996 PHP/FI был использован, по крайней мере, на 15 000 Web - сайтах во всем мире. В середине 1997 эта цифра выросла до 50 000 и даже более. В середи-

не 1997 произошли некоторые изменение в структуре и разработке PHP. Из частного проекта Р. Ледорфа, которому способствовала горстка людей, он превратился в организованную рабочую группу. Синтаксический анализатор был заново переписан З. Сураски (Zeev Suraski) и А. Гутмансом (Andi Gutmans), и этот новый анализатор стал основой для PHP третьей версии.

Сегодня, PHP поставляется с рядом коммерческих продуктов типа C2 Strong Hold Web Server и Red Hat Linux. Некоторой оценкой, основанной на экстраполяции чисел, предоставленных фирмой Net Craft, можно считать то, что PHP сегодня используется примерно на 150 000 сайтах во всем мире.

Возможности

Сегодня на языке PHP можно делать все, что можно было сделать с помощью CGI программ. Обрабатывать данные из форм, генерировать динамические страницы, получать и посылать Cookies (Кукис), загружать файлы, создавать файлы и папки, рисовать динамические картинки, отсылать электронную почту и многое другое.

Кроме того, в PHP включена поддержка многих баз данных (Databases), что серьезно расширяет возможности написания по - настоящему динамических Web - приложений. Язык PHP понимает протоколы IMAP, SNMP, NNTP, POP3 и HTTP, а также имеет возможность работать с Сокетами (Sockets) и общаться по другим протоколам.

Все Web - разработчики знают, что Web - страницы - это не только текст и картинки. Достойный внимания сайт должен поддерживать определенный уровень интерактивности с пользователем - всевозможные типы голосования, поиск информации, продажа продуктов, конференции, форум и т.п. Традиционно все это реализовывалось CGI - скриптами, написанными на языке Perl. Но CGI - скрипты очень плохо масштабируемы, т.е. каждый новый вызов CGI, требует от ядра сервера порождения нового процесса, а это занимает процессорное время и существенно тратит оперативную память.

Большое количество хакерских атак на сервера, основываются именно на многократных вызовах CGI, что приводит к загрузке ядра процессора и оперативной памяти. PHP предлагает другой вариант - он работает, как часть Web - сервера и тем самым похож на ASP от фирмы Microsoft.

Синтаксис языка PHP имеет легкую читаемость и в целом понятен для восприятия любыми пользователями. Если вам приходилось программировать на других языках, вы очень быстро сможете научиться писать программы на PHP. В этом языке нет строгой типизации данных и нет необходимости в действиях по выделению (или освобождению) памяти.

Программы, написанные на PHP, достаточно просты для понимания, а PHP - код легко прочитать и понять, в отличие от Perl - программ. Большим плюсом ко всему сказанному является достаточно высокая скорость работы PHP процессора, что особенно проявилось при переходе на четвертую версию.

Структура скрипта

PHP скрипт (код или программа на языке PHP) записывается в любом месте HTML документа внутри тегов

```
<?php
текст скрипта
?>
```

или

```
<?
текст скрипта
?>
```

Имеется и другая форма записи

```
<script language="php">
текст скрипта
</script>
```

Таким образом, имеются открывающий и закрывающий теги, ограничивающие действие PHP кода. Все, что находится внутри них, интерпретируется сервером, как программа на PHP и выполняется с использование PHP интерпретатора.

Еще раз подчеркнем, что отличие PHP, например, от JavaScript, состоит в том, что PHP - скрипт выполняется непосредственно на Web - сервере, а клиенту, т.е. вашему браузеру, передается только результат его работы. Программа на JavaScript

полностью передается на клиентскую машину (ваш браузер) и только там выполняется.

Приведем теперь несколько простых примеров использования PHP. Например, следующий код

```
<?php
echo "Hello, World!";
?>
```

выдает на экране браузера следующий текст

```
Hello, World!
```

Сразу скажем, что имена любых переменных в PHP обозначаются знаком доллара $. Тогда, то же самое "Hello, World!", можно получить следующим образом

```
<?php
$message = "Hello, World!";
echo $message;
?>
```

Поскольку PHP скрипт выполняется самим сервером, то просматривать результаты приведенных выше программ нужно при работающем Web - сервере с установленным PHP интерпретатором и непосредственно через Web - сайт.

Выполнение PHP скриптов

Так как PHP скрипт выполняется непосредственно сервером, то для дальнейшей работы нам понадобится Web - сервер с поддержкой PHP. Он необходим не только для изучения теоретического материала, но и для самостоятельной практики. Для написания самих скриптов будет нужен некоторый текстовой редактор, например, Блокнот из системы Windows.

Сервер

Вы можете использовать, как сервера Интернета (например, можно рекомендовать бесплатные хостинги Narod.ru, WebServis.ru или WallSt.ru), так и сервер на локальном компью-

тере или сервер локальной сети. Локальный вариант более предпочтителен, тем более, что установить его не очень сложно. Можно рекомендовать следующие серверы:

1. Сервер OmniHTTPd - может использоваться для системы Windows 95/98/ME и NT/2000/XP. Несомненным плюсом данного сервера является простота установки и эксплуатации. Вместе с инсталляцией данного сервера происходит и установка PHP интерпретатора, что значительно экономит время на установку всей системы. К минусам этого сервера можно отнести то, что данный продукт является коммерческим и время использования его нелицензионной версии ограничено. Кроме того, этот Web - сервер не всегда стабильно работает с переменными окружения сервера (о переменных окружения мы поговорим позднее).
2. Apache - несомненно, лучший сервер для работы с PHP. Существуют версии, как для системы Windows (работает под Windows 95/98/Me и Windows NT/2000/XP), так и для системы Linux. Apache характеризуется стабильной и надежной работой, но для налаживания взаимодействия этого сервера с PHP интерпретатором (его нужно скачивать и устанавливать отдельно от сервера) потребуются дополнительные и, иногда, немалые усилия. Если возникают проблемы при установке и настройке Apache + PHP, можно рекомендовать статью - "Apache + Perl + PHP4 + MySQL для Windows 95/98: руководство по установке", а также Интернет форум "Форум WIN32".
3. Covalent Fast Start Server - построен на базе Apache версии 2.0 и включает в себя автоматическую установку и настройку PHP, MySQL и Perl. Легко устанавливается на Windows 2000/XP, имеет монитор для управления сервером Apache и очень стабильно работает. После установки нужно лишь задать в файле конфигурации httpd.conf рабочую директорию и прописать виртуальные каталоги (плюс некоторые мелочи, описание которых не входит в данную книгу). Интерпретаторы PHP и Perl подключаются автоматически, и все настройки сервера сводятся к минимуму.

Редактор

Кроме сервера, нам понадобится любой текстовой редактор. Конечно, можно писать и в Блокноте, но это не вполне неудобно. Можно рекомендовать, например, редактор Arisesoft Winsyntax. В

нем есть и подсветка тегов, и PHP кода, и нумерация строк, так что найти ошибку в скрипте становится намного легче. Но вы вправе пользоваться любым удобным для вас редактором, например, известным RPad, который заменяет стандартный NotePad для Windows.

Исполняемый PHP код, после написания скрипта, должен храниться в файле с расширением php, phtml или phtm и если ваш редактор сохраняет файлы с другим расширением, не забудьте его поменять. При работе в NotePad для сохранения файла с правильным расширением нужно выбирать режим сохранения "Все файлы" - тогда редактор не будет добавлять в конце вашего имени файла с расширением PHP свое расширение TXT.

Кроме того, можно использовать очень хороший редактор Front Page XP в HTML режиме, который имеет удобную подсветку тегов и текстов скриптов.

СИНТАКСИС

Рассмотрим теперь общие способы написания PHP скриптов (кодов или программ), форму их записи или синтаксис языка PHP.

Форма записи

Как мы уже говорили ранее, все скрипты должны начинаться с тега

```
<?php
```

или

```
<?
```

и заканчиваться

```
?>
```

Это наиболее короткая форма записи, которая обычно и применяется при написании реальных кодов скриптов. Между этими символами будет выполняться каждая строка кода, кроме строк - комментариев.

Комментарии

Комментарии это некие пояснения, которые находятся внутри PHP программы и не выполняются PHP интерпретатором. Они служат для объяснения хода ее выполнения или описания блоков программы, пояснений для пользователей о назначении скрипта и т.д. Комментарии записываются несколькими способами

```
/* многострочный комментарий - это все, что заключено между этими символами */
```

или

```
// однострочный комментарий - все символы от этого знака до конца строки
```

или

```
# комментарием считаются все символы от данного знака до конца строки
```

Переменные

Как мы уже говорили, все переменные в PHP имеют префикс $ (знак доллара), а далее идет само имя переменной. Имя переменной не должно начинаться с цифры, хотя внутри него может содержаться любое количество цифр. Заметим, что имена переменных чувствительны к регистру, т.е.

```
ABC
```

и

```
Abc или ABc или aBC или abC или abc
```

разные переменные и им можно присваивать различные значения. Именно на этот момент нужно обращать пристальное внимание, иначе ваша программа работать не будет.

Значения строковых (символьных) переменных, как обычно, обрамляются двойными (") или одинарными (') кавычками (об их различии мы поговорим несколько позже). Разделение нескольких операторов выполняется знаком ; (точка с запятой).

Познакомимся теперь с одной из самых распространенных функций PHP - функцией вывода на экран echo. Например, скрипт

```
<?
echo "Мы изучаем PHP!";
?>
```

выведет на экран текст

```
Мы изучаем PHP!
```

без какого - либо форматирования, т.е. это будет текст, в котором используются шрифты вашего браузера, установленные "по умолчанию". Внутри кавычек функции echo можно разме-

щать не только любой текст, но любые HTML - теги. В таком случае выводимый текст будет отформатирован в соответствии с общими правилами языка HTML

Конечно, функция echo была бы бесполезна, если бы могла выводить только одно и то же сообщение или только заданный в ней текст. Совершенно естественно, что эта функция позволяет выводить на экран значения переменных, присвоение которых происходит в самой программе.

Теперь нужно сказать, что простейшим оператором любого языка является оператор присваивания значений переменным. Он записывается с помощью знака = (равно) и задает переменной, стоящей слева, некоторое значение, расположенное справа от этого оператора. Причем, пробелы между переменными, их значениями и этим знаком (как, впрочем, и для всех других операторов языка PHP) просто игнорируются, т.е. записи вида

```
$a=1;
$a =1;
$a= 1;
$a = 1;
```

полностью эквивалентны. Рассмотрим теперь пример присвоения значений переменным с помощью строковых выражений и вывод этих значений на экран браузера. Здесь и далее, с помощью комментариев, мы будем пояснять ход выполнения программы

```
<?
$text = 'Привет от PHP!';
/* присваиваем переменной $text значение 'Привет от PHP!' */
echo $text;
// выводим в браузер значение переменной $text
?>
```

В окне браузера увидим

Привет от PHP!

Конечно, этот скрипт должен выводится через сервер с подключенным PHP интерпретатором, иначе вы вообще ничего не

увидите.

Большой плюс PHP, в отличие от других языков программирования, скажем от Си или Perl, заключается в том, что нет необходимости строго придерживаться типизации переменных, т.е. заранее определять тип некоторой переменной. В любой момент программы, в любом ее месте, вы можете числовой переменной присвоить строковое значение, строковой переменной - вещественное или целое значение и т.д. (Более подробно этот вопрос мы рассмотрим в следующей главе).

Операторы равенства

Рассмотрим теперь несколько вариантов записи операторов равенства, которые используются в программах на PHP, в тех или иных случаях

- Присваивание = - знак равенства, присваивает левой части (относительно знака =), значение, стоящее справа от этого знака.
- Равенство == - двойной знак равенства, используется в логических выражениях, когда нужно сравнить значения двух переменных.
- Тождественно === - тройной знак равенства, используется, если переменные не только равны, но и относятся к одному типу переменных, например, строка, целое число, массив и т.д.

Несколько забегая вперед, рассмотрим пример, в котором используются приведенные выше знаки =. Здесь мы используем оператор сравнения двух переменных if, который будет рассмотрен далее более подробно

```
<?
/* присваиваем переменной $text значение ' PHP!' */
$text = ' PHP!';
$stroka = $text;
// присваиваем переменной $stroka значение
// переменной $text
/* оператор if является условным оператором и сравнивает выражения, стоящие в скобках */
if ( $stroka == 'PHP')
// если сравниваемые величины равны,
```

```
/* а они в данном случае не равны, т.к. в переменной $text стоит начальный пробел ' PHP') */
echo "Привет от $stroka";
// выведет в браузер - Привет от PHP!
else
// в противном случае
echo 'Никаких приветов!';
// выведет - Никаких приветов!
?>
```

На экране браузера увидим

```
Никаких приветов!
```

На начальном этапе программирования у вас почти наверняка возникнут проблемы со знаком равенства и присваивания. Так что внимательно следите за тем, как вы пишите код скрипта. Если в одном из выражений вы, вместо == (равно), напишите = (присвоить), PHP интерпретатор не выдаст никакой диагностики, а в программе будет ошибка.

Кавычки

Вернемся теперь к двум типам кавычек, которые вы уже видели в приведенных примерах. Если мы ставим одинарные кавычки (апострофы), то все символы, заключенные в них выводятся на экран в неизменном виде. Если значения переменных обрамляется двойными кавычками (кавычки), то PHP интерпретирует, т.е. подставляет значения во все встречающиеся в строке переменные.

Рассмотрим пример использования таких кавычек

```
<?php
/* присваиваем переменной $text значение 'PHP!' */
$text = 'PHP!';
echo '1 - Привет от $text';
// выведет на экран вариант 1
echo "2 - Привет от $text";
// выведет на экран вариант 2
?>
```

В браузере будем иметь два варианта вывода

1 - Привет от $text
2 - Привет от PHP!

Этот пример наглядно демонстрирует различие одинарных и двойных кавычек.

ПЕРЕМЕННЫЕ

Во всех языках программирования используются переменные величины, которые определяются своим именем. Имя переменной имеет буквенное обозначение, называемое идентификатором переменной и под каждую переменную выделяется определенная компьютерная память. Обычно считается, что сама переменная величина и отводимая под нее память, обозначаются одинаково.

Еще раз подчеркнем, что все переменные в PHP чувствительны к регистру, а это значит, что

```
$text
$TEXT
$TeXt
```

представляют собой три совершенно разные переменные. Как уже говорилось, в языке PHP нет строгой типизации переменных. Интерпретатор, в том или ином месте кода, сам определяет, к какому типу относится данная переменная.

Однако, и он может ошибаться, например, в том случае, если в текстовой строке будет находиться десятичное число, в котором присутствует точка, как разделитель целой и дробной части. Поэтому, иногда возникает необходимость указывать явно, какой тип имеет некоторое выражение или переменная.

В отводимой под переменную величину памяти могут храниться значения разных типов - это может быть любой текст или некоторое число. Здесь главное то, что после того, как переменной величине задано имя и присвоено некоторое значение, на нее можно ссылаться и из памяти, отведенной для нее, можно получать интересующую вас информацию.

Для того, чтобы создать переменную, нужно выполнить следующие очень простые шаги

1. Придумать подходящее для нее имя, например, если вы хотите хранить имя пользователя, то подходящим можно считать имя username (что и означает имя пользователя).
2. Поместить символ доллара ($) перед именем - $username.
3. Поставить знак равенства после имени переменной $username= и присвоить ей некоторое значение.
4. Завершить строку точкой с запятой - ; .

Итак, имена переменных в PHP начинаются с символа $ - доллар. После знака $ не может стоять цифра, но внутри или в конце, имя может содержать цифры. Кроме того, в имени может присутствовать символ подчеркивания _ .

Следует заново повторить очень важное положение - имена переменных величин чувствительны к регистру. Это значит, что

```
$username
```

и

```
$USERNAME
```

совершенно разные величины, которые хранятся в разных ячейках памяти.

Рекомендуется всегда давать имена переменным, наполненные смыслом. Это значит, что лучше написать имя $name, чем просто $n или имя $password, чем просто $p. Даже, если вы сами написали свою программу и потом, спустя некоторое время, начнете разбираться в ней, то есть очень большая вероятность запутаться в коротких именах. Ведь переменная $n может обозначать, как name, так и number, а $p может быть и page, и password.

Типы переменных

При программировании на PHP в основном приходится иметь дело с двумя типами переменных - это скалярный тип и массивы. Скалярные величины содержат только одно значение, а массивы состоят из списка значений или даже из нескольких списков. Например, вы создали переменную

```
$username = "String";
```

или

```
$pass = 123;
```

это просто скалярные величины, которым присвоено одно определенное значение.

Отметим, что когда создается переменная величина, то обычно используется один из следующих типов значений

1. Целое без дробной части (integer).
2. Число с плавающей точкой (floating - point, "float" или "double"), т.е. число с дробной частью.
3. Строка или строковая переменная (string).

В последнем случае это будет выражение, состоящее из текста или текста вместе с цифрами, заключенное в кавычки - парные двойные (" ") или простые одинарные (' ').

Кроме того, в PHP явно существует несколько типов переменных, которые можно указывать в тексте программы

1. ***Int*** или ***Integer*** - Целое число, длиной в 32 байта, в диапазоне от - 2147483648 до 2147483647.
2. ***Real, double*** или ***float*** - Вещественное число (число с плавающей точкой и точка является разделителем целой и дробной части числа). При работе с вещественными числами не следует доверять последнему десятичному разряду, поскольку он содержит ошибку округления. Это связано с тем, что простые десятичные дроби, например, 0,1 или 0,7 не могут преобразовываться во внутреннее двоичное представление без некоторой потери точности. В результате использования, например, выражения (0,1+0,7)*10, мы получим не целое число 8, как могли бы ожидать, а число 7 с дробной частью, поскольку внутреннее представление результата будет 7,999999.. или 8,0000...1.
4. ***String*** - Строка или символы - последовательность алфавитно - цифровых символов. Длина строки ограничена только размером свободной памяти компьютера. Вполне реально поместить в одну строку целый файл и размер его может быть, например, в 200 или 300 Кб. Для работы со строками предназначены стандартные, встроенные функции (о некоторых из них мы поговорим позднее). Кроме того, можно обратиться к любому символу строки, как к индексу массива, т.е. запись вида

```
$a = "abcdegf";
$b = $a[5];
```

приведет к присвоению величине $b значения "e".

5. ***Array*** - Массивы - в общем случае, это набор данных, представляющий собой последовательность вида "ключ => значение" (ассоциативный или хэш массив). Ключом могут служить

не только цифры, но и строки. Это некая таблица, содержащая пары "имя переменной" => "ее значение". Здесь символом => обозначается соответствие определенному ключу какого - то определенного значения. Доступ к отдельным элементам массива осуществляется указанием их ключа. Например, вполне возможно существование таких команд

```
$a=array(0=>"zzzz", "a"=>"aaa", "b"=>"bbb", "c"=>"ccc");
//здесь мы создаем массив с ключами "0", "a", "b" и "c"
echo $a["b"];
// выведет на экран "bbb"
$a["1"]="qq";
/* создаст новый элемент в массиве и присвоит ему значение "qq" */
$a["a"]="new_aaa";
/* присвоит существующему элементу новое значение "new_aaa" */
```

6. ***List*** - Список - обычно это массив с целыми ключами, пронумерованными, начиная с нуля. Список является некоторой разновидностью ассоциативного массива или хэша. Поэтому вместо типа list всегда можно использовать и тип array. При этом программа ничего не заметит и будет работать с этим массивом, как со списком. Список представляет собой набор значений и его можно сортировать в порядке возрастания или убывания. В то же время, ассоциативный массив - это упорядоченный набор пар значений и нет никакого смысла разъединять эти пары.

7. ***Object*** - Объект какой - то структуры. Обычно структура объекта уточняется в самой программе. Объект реализует несколько простейших принципов объектно - ориентированного программирования. Внутренняя структура объекта похожа на хэш - массив. Для доступа к отдельным элементам и функциям объекта используется оператор -> (стрелка), а не квадратные скобки.

8. ***Bool*** - Логический тип, который содержит только одно из двух возможных значений true (истина) или false (ложь).

9. ***Void*** - Это самый простой тип, который применяется для определения возвращаемого функцией значения. Этот тип не возвращает ничего. Но с учетом того, что в языке PHP функция не может ничего не возвращать, все void - функции возвращают false, т.е. пустую строку.

10. ***Mixed*** - Это может быть или целое, или дробное число. Например, тип mixed имеет стандартная функция gettype() или settype(). Если указано, что функция возвращает mixed, то это значит, что тип результата зависит от операндов и уточняется при описании функции.

Для того, чтобы проверить, существует ли в данный момент и в данном месте программы некоторая переменная (т.е. была ли она использована и было ли ей присвоено некоторое значение), можно использовать функцию IsSet(), а в скобках нужно указать имя этой переменной. Если переменной в данный момент не существует, либо она была ранее удалена, то функция IsSet() возвращает значение false (ложь), в противном случае - true (истина).

Для того, чтобы удалить переменную, необходимо вызвать функцию UnSet() и в скобках указать ее имя. Некоторые типы стандартных функций и их использование мы подробно рассмотрим в следующих главах.

Все переменные в приведенных выше примерах фиксированы, т.е. задаются в самой программе. Для их изменения нам необходимо самим менять их значения в тексте скрипта. Во многих случаях этого бывает вполне достаточно, но не редки ситуации, когда нужно вводит значения таких переменных, как бы извне, т.е., не меняя текста программы, присваивать ее переменным новые значения.

Такая возможность осуществляется благодаря стандартным формам языка HTML. Каждому полю формы с заданным именем, например, name = abc, сопоставляется PHP переменная с таким же именем, т.е. $abc. Присвоение значений переменным PHP происходит после нажатия кнопки OK на HTML форме (эти вопросы мы рассмотрим в следующих главах).

Примеры переменных

Воспользуемся редактором типа Блокнот, например, RPad и запишем в некотором файле следующий HTML текст

```
<HTML>
<HEAD>
<TITLE>
Печать значений переменных величин
</TITLE>
```

```
<BODY>
```

Далее добавим сюда PHP блок и создадим переменные разных типов - целого, с плавающей точкой, строчную и добавим для каждой из них команду echo, чтобы вывести их значения на экран браузера

```
<?
$int = 52164;
$float = 732.23;
$string = "Строка";
// или так
$string = 'Строка';
echo "integer (целая) : $int";
echo "<br>float (дробная) : $float";
echo "<br>string (строчная) : $string";
?>
</BODY>
</HTML>
```

Сохраните эту программу в файле, например, с именем print.php в папке с документами для PHP программ. Обычно это папки - internet\home\www. Запустите Web - браузер, наберите в строке адреса http://localhost/print.php и нажмите клавишу Enter - на экране вы получите значения этих переменных.

Присвоение значения

Еще раз напомним, что для присвоения значений переменным в PHP программах используется обычный оператор присваивания, т.е. в нем присутствует обыкновенный знак равенства "="

```
$имя _переменной=значение;
```

В отличие от этого знака, который применяется в операторе присваивания, в PHP применяется и двойной знак равно "==". Он используется, как и в языке Си, в условных операторах, например

```
if (a==b)...
```

Если в языке Си в операторе if пропустить один знак равно, то компилятор выдаст сообщение об ошибке. В PHP это не так, например, если выполняется программа

```
$a=0;
$b=1;
if ($a=$b)
echo "a и b одинаковы";
else
echo "a и b различны";
```

на экран будет выдано

```
a и b одинаковы
```

так как соотношение

```
if ($a=$b)
```

истинно.

Как обычно, присвоение предполагает, что старое значение переменной и ее тип полностью теряются, а новое значение переменной становится полной копией той величины, которая записывается в эту переменную.

Напомним, что всегда можно проверить, инициализирована ли некая переменная, как переменная с определенным именем. Выполняется такой контроль с помощью оператора IsSet(). Например

```
If (IsSet($MyVar)) echo "Такая переменная есть и ее значение $MyVar";
else
echo "Переменная $MyVar не определена";
```

Если переменной $MyVar в данный момент и в данном месте программы нет, т.е. ей нигде ранее в программе не присваивалось никакое значение, либо она была удалена ранее при помощи специального оператора UnSet(), то значение IsSet(), при проверке, устанавливается в состояние "ложь". В противном случае вы будете иметь истинное значение функции IsSet().

Вы не можете использовать в программе неинициализиро-

ванную ранее переменную - это приведет к предупреждению со стороны интерпретатора PHP.

Рассмотрим теперь более подробно новое действие над переменными - уничтожение. Уничтожение переменной реализуется оператором UnSet(). После этого действия из внутренних таблиц интерпретатора, т.е. из памяти компьютера удаляется всякое воспоминание об этой переменной, и она снова может быть установлена и использована, как в первый раз.

Например, рассмотрим следующий текст

```
// Переменная $a еще не существует
$a="Hello there!";
// Теперь переменная $a инициализирована
//... здесь идут какие-то команды сценария и так далее
echo $a;
// Значение переменной будет выведено на экран
// Теперь удалим переменную $a
Unset($a);
// Теперь переменной $a не существует
echo $a;
// Это ошибка - нет такой переменой
```

Команда или функция UnSet() часто применяется при работе с массивами. Например, нужно удалить из массива $A элемент с ключом del. Это можно сделать так

```
UnSet($A["del"]);
```

В этом случае элемент del не просто станет пустым, он полностью удалится. Проверка или поиск этого элемента, такой элемент не обнаружит.

Логические переменные

Логические переменные могут принимать одно из двух возможных значений - истина (true) или ложь (false). В языке PHP имеется некоторая особенность - любое ненулевое число в ячейке памяти или не пустая строка, а также ключевое слово true или число 1 рассматриваются, как истина. Напротив, нулевое значение в ячейке, то есть число 0, пустая строка и false - это ложь с точки зрения логики.

Например, в следующем фрагменте кода

```
echo false;
//выводится пустая строка, т.е. false - это пустая строка
echo true;
//выводится число 1
```

Разберем теперь небольшую программу (здесь мы несколько забегаем вперед и используем операторы или конструкции, которые будут рассмотрены в следующих главах) и проверим работу команды if с логической переменной

```
<?
$a=100;
if ($a==1) echo "Переменная равна 1!<BR>";
if ($a==true) echo "Переменная истинна! ";
?>
```

Занесите текст этого примера, например, в файл

```
c:/internet/home/localhost/www/ex1.php
```

Здесь снова нужно напомнить, что PHP скрипты выполняются самим сервером и для проверки всех приведенных выше примеров нужно иметь на своем компьютере установленный и запущенный Web - сервер, например, Apache с подключенным PHP интерпретатором. Обычно основная директория сервера называется WWW и именно в нее нужно записать файл с нашим примером. По умолчанию имя сервера задается - localhost.

Далее, воспользуйтесь каким - нибудь браузером, например, Internet Explorer и задайте в окне адреса команду

```
http://localhost/ex1.php
```

Можно убедиться, что при выполнении программы будет исполнена только вторая, из двух команд if, и на экран будет выведено сообщение

```
Переменная истинна!
```

В первой команде записана проверка логического условия

при сравнения двух целых чисел

```
if ($a==1) echo "Переменная равна 1!<BR>";
```

поскольку $a действительно не равно 1, то условие $a==1 не выполняется и сообщение на экран не выводится.

Во второй команде величина true - логическая и потому содержимое ячейки памяти $a рассматривается, как логическая переменная. А поскольку логической величине true соответствует любое ненулевое значение переменной, то это условие и будет выполняться.

Рассмотрим еще один пример ex2.php

```
<?
$a=100;
$b=true;
echo "a = $a<br>";
echo "b = $b<br>";
if ($a==$b) echo 'a равно b!';
?>
```

Такая программа выдаст на экран

```
a=100
b=1
a равно b!
```

Так и должно быть, потому что переменная $b имеет логический тип true и переменные $a и $b сравниваются, как логические. А вот при выполнении арифметических операций логические переменные преобразуются в целые числа 1 - true и 0 - false. Именно поэтому при выдаче на экран величина $b имеет значение 1.

Преобразование типов

Если некоторой переменной приравнивается строка (т.е. текст в кавычках), то эта переменная становится строковой. Если с такой переменной выполняется одна из математических функций или она приравнивается численной переменой, то она становится численной, причем, если она приравнивается не целому

значению, то переменная будет иметь тип double.

В любом месте программы с помощью функций

```
SetType();
```

можно задать или изменить тип некоторой переменной явным образом. Для использования этой функции нужно написать

```
$a = SetType(var, type);
```

тогда тип переменной var изменяется на type, а возможными значениями параметра type могут быть

```
integer
double
string
array
object
```

Сама функция возвращает (задает значение переменной $a) true при успехе операции или false в противном случае.

Определить, какой тип имеет переменная в данном месте программы можно с помощью функции

```
GetType();
```

Для ее использования нужно написать

```
$a = GetType(var);
```

Переменная $a будет типа string с возможными значениями

```
integer
double
string
array
object
unknown type
```

Если мы напишем программу

```
$a = SetType($v, string);
echo $a."<br>";
$a = GetType($v);
echo $a;
```

то на экране получим

```
1
string
```

Как мы уже говорили, в некоторых случаях строковая переменная может быть оценена интерпретатором PHP, как числовая (если в ее значении присутствуют числа). Тогда результирующее значение и тип этой переменной определяются следующим образом

1. Если в начале строки стоят числа, она будет оценена, как числовая, т.е. если строка начинается с допустимых цифровых данных, то это значение и будет использоваться в дальнейшем.
2. Допустимые цифровые данные - это конструкция из любого буквенного символа, следующего за одной или несколькими цифрами.
3. Переменная типа string будет оценена, как double (float - с плавающей точкой), если она содержит любой из символов '.' , 'e' или 'E'. Иначе ее тип будет оценен, как integer - целый. Экспонента может обозначаться символом 'e' или 'E', который может следовать за одной или несколькими цифрами.

Рассмотрим несколько примеров

```
$f = "0";
echo "$f<br>";
// $f это строка "0" с ASCII кодом 48
$f="1";
echo "$f<br>";
// $f тоже строка "1" с ASCII кодом 49
$f= 2;
echo "$f<br>";
// $f становится теперь типа integer со значением 2
$f = $f + 1.3;
echo "$f<br>";
```

```
/* $f становится переменной типа double со значением
3.3, поскольку предыдущее значение $f было равно 2 */
$f = 5 + "10 строк";
echo "$f<br>";
// $f теперь снова становится типа integer
// со значением 15
$f = 5 + "15 операторов";
echo "$f<br>";
// $f по - прежнему integer со значением 20
$f = 1 + "10.5";
echo "$f<br>";
// $ тип double, значение 11.5
$f = 1 + "-1.3e3";
echo "$f<br>";
// $f тип double, значение -1299.0
$f = 1 + "bob - 1.3e3";
echo "$f<br>";
// $f тип integer, значение 1
$f = 1 + "bob3";
echo "$f<br>";
// $f тип integer, значение 1
$f = 1 + "10 Symbols";
echo "$f<br>";
// $f тип integer, значение 11
```

Такая программа выдаст на экран следующую информацию

```
0
1
2
3.3
15
20
11.5
-1299
1
1
11
```

в полном соответствии с комментариями в самой программе. Рассмотрим теперь другие способы определения типа перемен-

ной. Определить ее тип можно и с помощью следующих функций

```
is_integer($a);
```

Возвращает true, если $a - целое число.

```
is_double($a);
```

Возвращает true, если $a - действительное число.

```
is_string($a);
```

Возвращает true, если $a - является строкой.

```
is_array($a);
```

Возвращает true, если $a - является массивом.

```
is_object($a);
```

Возвращает true, если $a - объявлена, как объект.

```
is_boolean($a);
```

Возвращает true, если $a - определена, как логическая переменная. Результат работы любой из этих функций может выводиться на экран следующим образом

```
echo is_integer($a);
```

Кроме того, для преобразования или приведения типа можно использовать перед переменной, которая должна быть приведена к данному типу, запись типа в круглых скобках (префиксная операторная форма), например

```
$f = 10;
// задана переменная $f типа integer
$b = (double) $f;
// а теперь переменная $b стала типа double
```

Для демонстрации изменения типа переменной можно напи-

сать простой скрипт

```
<?
$f = 10;
$a = GetType($f);
echo "$a<br>";
// задана переменная $f типа integer
$b = (double) $f;
$a = GetType($b);
echo $a;
// а теперь переменная $b стала типа double
?>
```

На экране мы увидим

```
integer
double
```

Допускается следующее приведение типов

- (int), (integer) - приведение к целому.
- (real), (double), (float) - приведение к типу double.
- (string) - приведение к строке.
- (array) - приведение к массиву.
- (object) - приведение к объектной переменной.
- (bool) - приведение к логическому типу.

Заметим, что внутри круглых скобок допускаются табуляция и пробелы, поэтому следующие записи полностью эквивалентны

```
$f = (int) $bar;
$f = ( int ) $bar;
```

Кроме префиксной формы возможна и функциональная запись приведения типа. Например, следующие две инструкции полностью эквивалентны

```
$a=intval($b);
$a=(int)$b;
```

Тем же способом можно пользоваться для преобразования

типов целых выражений

```
$b=intval(выражение)
```

или

```
$b=(int)(выражение)
```

Здесь значение выражения переводится в целое число, которое присваивается переменной $b. Следующий пример

```
$b=doubleval(выражение)
```

или

```
$b=(double)(выражение)
```

переводит значение выражения в действительное число и присваивает его переменной $b.

Для строковых выражений можно использовать запись вида

```
$b=strval(выражение)
```

или

```
$b=(string)(выражение)
```

которая переводит значение выражения в строку. И наконец

```
$b=(bool)(выражение)
```

преобразует значение выражения в логический тип. После выполнения этого оператора в переменной $b будет находиться значение true или false.

Область действия переменных

Областью действия переменной является блок текста скрипта, внутри которого она определена и имеет некоторое значение. В основном, все переменные PHP имеют одну общую область действия. Однако, внутри функций, определенных пользователем

(описание функций смотрите в следующих главах), возможна локальная область действия переменной. Любая переменная, определенная внутри функции, "по умолчанию" ограничена областью этой функции.

Например, если переменная определена внутри самого скрипта, т.е. в глобальной области

```
$a = 1;
/* глобальная область */
```

то она действует во всем скрипте, кроме внутренней области функций. Таким образом, если определена функция

```
function test()
{
/* локальная область */
$a = 2;
echo $a;
/* ссылка на переменную локальной области */
}
```

то ее при вызове

```
test();
```

эта функция выдаст на экране цифру "2", поскольку инструкция echo относится к локальной версии переменной $a, определяемой внутри самой функции. Если мы не определим переменную $a внутри функции, то оператор echo ничего на экран не выдаст.

В языке PHP глобальные переменные, определенные в самом скрипте должны быть продекларированы внутри каждой функции, если предполагается их использование в данной функции.

Например

```
<?
$a = 1;
$b = 2;
/* глобальная область */

function sum()
```

```
{
global $a, $b;
$c = $a + $b;
echo $c;
}
?>
```

При вызове функции

```
sum();
```

описанный выше скрипт выдаст на экране значение "3". Поскольку $a и $b декларируются глобально внутри функции, ссылки на данные переменные трактуются, как ссылки на их глобальные версии. Если внутри функции переопределить эти переменные, то и вне нее, т.е. в глобальной области они поменяют свои значения

```
<?
$a = 1;
$b = 2;
/* глобальная область */
sum();

function sum()
{
global $a, $b;
$c = $a + $b;
echo $c;
$a=3;
$b=5;
}

echo "$a, $b";
?>
```

На экране получим

```
3
3, 5
```

Вторым способом доступа к переменным из глобальной области внутри функции является использование специально определяемого PHP массива $GLOBALS (описание массивов смотрите в следующих главах). При этом предыдущий пример может быть записан в виде

```
$a = 1;
$b = 2;

function sum ()
{
$GLOBALS["c"] = $GLOBALS["a"] + $GLOBALS["b"];
echo $c;
}
```

При вызове этой функции

```
sum();
```

мы также получим цифру 3. Массив $GLOBALS является ассоциативным массивом, в котором имя глобальной переменной является ключом, а значение этой переменной является значением элемента массива.

Другой важной характеристикой области определения переменной является статическая переменная. Статическая переменная существует только в локальной области функции, но она не теряет своего значения, когда программа, при исполнении, покидает эту область.

Рассмотрим следующий пример

```
function test()
{
$a = 0;
echo $a;
$a++;
}
```

Эта функция совершенно бесполезна практически, поскольку каждый раз при ее вызове она устанавливает $a в 0 и выводит на экран "0". Выражение $a++, которое увеличивает значение переменной, так же бесполезно, поскольку при выходе из функ-

ции переменная $a теряется (выражения типа $a++ будут подробно рассмотрены в следующих главах).

Для придания дееспособности функции подсчета, которая не теряла бы значения текущего значения, переменная $a декларируется, как статическая

```
function test()
{
static $a = 0;
echo $a;
$a++;
}
```

Теперь, каждый раз при вызове функции test() она будет выводить значение $a и увеличивать его на единицу.

Статические переменные также весьма полезны, когда функции вызываются рекурсивно. Рекурсивные функции - это те функции, которые вызывают сами себя. Составлять рекурсивную функцию нужно внимательно, т.к. при неправильном ее написании можно сделать рекурсию неопределенной. Вы должны быть уверены в адекватности способа прекращения рекурсии.

Следующая простая функция рекурсивно считает до 10

```
function test()
{
static $count = 0;
$count++;
echo $count;
if ($count < 10) {test();}
}
```

Однократный вызов такой функции, она выдаст на экран последовательность чисел от 1 до 10.

Изменяемые переменные

Иногда бывает удобно давать переменным изменяемые имена, которые могут изменяться динамически. Обычная переменная устанавливается в виде

```
$a = "hello";
```

Изменяемая переменная берет некое значение и обрабатывает его, как имя переменной. В приведенном выше примере значение hello может быть использовано, как имя переменной, посредством применения двух записанных подряд знаков доллара

```
$$a = "world";
```

С этой точки зрения, две переменные определены и сохранены в символьном дереве PHP - $a с содержимым "hello" и $hello с содержимым "world". Таким образом, инструкция вида

```
echo "$a ${$a}";
```

осуществляет то же самое, что и инструкция

```
echo "$a $hello";
```

а именно, обе они выведут на экран

```
hello world
```

Константы

Константы или постоянные величины - это часто встречающиеся величины, значения которых не меняются в течение работы программы. Это могут быть математические константы, например, число π = 3.14159. Это может быть путь к файлам или пароли пользователя и т.д. Еще раз подчеркнем, что константы нельзя переопределять в процессе работы программы.

Рассмотрим, например, уже упоминавшуюся выше константу π в языке PHP. Так скрипт

```
// Использование математической константы pi=3.14159
$a=2.34*sin(3*pi/8)+5;
echo "Это число pi";
```

выводит на экран браузера следующий текст

```
Это число pi
```

Отметим одну важную особенность использования констант

- не требуется писать знак "доллар" перед именем константы, как это делается при использовании переменной величины.

При выводе имени константы в кавычках, например, "Это константа pi", само значение этой константы на экран не выводится, как это делается для любой переменной.

Всего имеется два вида констант

1. Предопределенные, устанавливаемые самим интерпретатором PHP.
2. Устанавливаемые программистом в процессе работы над программой.

Существует всего несколько констант первого типа

- **_FILE_** - хранит имя файла программы, которая выполняется в данный момент.
- **_LINE_** - хранит текущий номер строки, которую обрабатывает интерпретатор в данный момент.
- **PHP_VERSION** - хранит версию интерпретатора PHP.
- **PHP_OS** - хранит имя операционной системы, с которой работает PHP.
- **TRUE или true** - хранит значение "истина".
- **FALSE или false** - хранит значение "ложь".

Для начинающих программистов на PHP, эти константы могут очень существенно помощь, так как с их помощью можно отслеживать большое количество ошибок.

Теперь поговорим о втором типе констант - определяемых программистом. Для определения своей собственной константы используется оператор define(), который имеет простую форму записи

```
define(name, value [, необязательный параметр $case_sen]);
```

где name - имя константы, value - значение константы, а $case_sen - может принимать два значения - false (регистр символов константы не учитывается) или true ("по умолчанию", регистр символов константы учитывается). При определении собственной константы используются двойные кавычки, которые обрамляют имя константы и ее значение.

Например

```
<?
define("text", "Привет от PHP!");
define("num", 10);
$result = num*5;
echo text;
```

если в операторе echo поместить text в кавычки, то на выходе мы получим просто слово text вместо значения константы text, равное

```
Привет от PHP!
```

Знак звездочки в четвертой строке этого примера обозначает операцию умножения, в частности, здесь константа num, равная 10, умножается на число 5.

Продолжаем нашу программу и дополнительно сделаем вывод переменной $result

```
echo "<br>$result";
?>
```

тогда в окне браузера мы получим следующий результат

```
Привет от PHP!
50
```

Еще раз подчеркнем, что нельзя несколько раз определить константу с одним и тем же именем и разными значениями.

Кроме того, в языке PHP существует функция, которая проверяет, существует ли (т.е. была ли она ранее определена) константа с указанным именем

```
$a = defined(name);
```

Если константа с названным именем уже была определена, то данная функция возвращает true, т.е. значение переменной $a будет равно true. Приведем пример

```
<?
```

```
define("text", "Привет от PHP!");
echo text."<br>";
$a = defined("text");
echo $a;
?>
```

на экране увидим

```
Привет от PHP!
1
```

это говорит о том, что заданная константа существует, т.е. $a = true.

ВЫРАЖЕНИЯ

Выражения - это основа любого языка программирования и, в том числе, языка PHP. В скриптах PHP почти все, что они содержат, является выражениями. Простейший и, по - видимому, наиболее точный способ определения выражения - это "нечто, что имеет определенное значение". Или наоборот, если "что - то имеет определенное значение - это и есть выражение". В каждой программе пишутся некоторые формулы, определенные соотношения и все это делается именно с помощью выражений.

Простые выражения

Простейший пример выражений это константы и переменные языка PHP. Когда вы пишете

```
$a = 5;
```

вы присваиваете значение 5 переменной $a - это и является выражением, определяющим величину переменной (в данном случае число 5 это целочисленная константа).

Кроме того, выражением будет и следующее использование оператора присваивания

```
$c=$a;
$d=$a+$b;
```

Приведем следующий пример

```
$a=$b=$c=10;
```

в языке PHP возможна и такая запись, когда нескольким переменным в одном выражении присваивается одно и тоже число. Еще один пример

```
$a=3*sin($b=$c+10)+$d;
```

После выполнения этих команд вы получите то же самое, что и после выполнения такого фрагмента программы

```
$b=$c+10;
```

```
$a=3*sin($b)+$d;
```

или

```
$a=3*sin($c+10)+$d;
```

Отсюда видно, что в PHP, при вычислении сложного выражения, можно задавать значения переменным прямо в самом операторе присваивания.

Несколько более сложные примеры выражений - это функции (более подробно о функциях будет написано в следующих главах). Например, подумайте над функцией

```
function f()
{
return 5;
}
```

В самом тексте скрипта мы можем обратиться к ней следующим образом

```
$c = f();
```

и это практически то же самое, что написать

```
$c = 5;
```

Функция - это выражение с тем значением, которое она возвращает, и в данном случае это целое число 5. Возвращаемое значение это результат действия функции, т.е. это значение, которое является результатом вычислений, выполняемых функцией.

Конечно, значения в PHP не обязаны быть только целыми и зачастую они такими не являются. Как мы уже говорили, язык PHP поддерживает 3 скалярных типа значений или переменных - целое число, число с плавающей точкой и символьные строки. Скалярные выражения нельзя "разбить" на более маленькие части, как, например, массивы.

Кроме того, PHP поддерживает 2 составных (не скалярных) типа переменных - это массивы и объекты. Каждое из таких значений может быть присвоено переменной или возвращено функ-

цией.

Инкремент и декремент

Другой хороший пример выражения это предварительное и последующее увеличение (инкремент) и уменьшение (декремент) значения переменной. Пользователи PHP и многих других языков могут быть знакомы с записями типа variable++ и variable-- - это операторы увеличения и уменьшения или инкремента и декремента.

В PHP, подобно языку C, есть 2 типа инкремента (увеличения значения переменной) - предварительный и последующий. И предыдущий, предварительный и последующий инкремент увеличивает значение переменной на единицу. Разница в значении выражения инкремента заключается в том, что предыдущее увеличение (пре - инкремент), которое записывается, как ++$variable, приравнивается увеличенной переменной - PHP увеличивает переменную до того, как "прочитать" ее значение. Для пост - инкремента $variable++ увеличение переменной происходит уже после ее "прочитывания" интерпретатором PHP.

Следующий пример должен помочь вам лучше понять предварительное и последующее увеличение значений переменных и вообще выражения

```
function double($i)
/* функция увеличения переменной на 2 */
{
return $i*2;
}

$b = $a = 5;
/* присваиваем значения переменным $a и $b - такое выражение вполне допустимо */
$c = $a++;
/* последующее увеличение, т.е. присваиваем переменной $c начальное значение $a, равное 5 */
$e = $d = ++$b;
/* предварительное увеличение, присваиваем $d и $e увеличенное значение $b, которое равно 6 */
$f = double($d++);
/* присвоить переменной $f удвоенное значение $d до
```

```
его увеличения, то есть 2*6 = 12 */
    $g = double(++$e);
    /* присвоить переменной $g удвоенное значение $e после его увеличения, то есть 2*7 = 14 */
    $h = $g+= 10;
    /* сначала увеличить значение $g на 10, что дает в результате 24, а затем присвоить это значение переменной $h, что также дает 24 */
```

Приведем другой пример

Выражения	***Эквивалент***
$a+= 2 или $a-= 3	$a = $a + 2 или $a = $a - 3
$a*= 10 или $a/= 5	$a = $a * 10 или $a = $a / 5
$a++ или $a--	$a = $a + 1 или $a= $a - 1
++$a или --$a	$a = $a + 1 или $a= $a – 1

Еще раз подчеркнем, что отличие последних двух выражений заключается в том, что при использовании выражения типа $a++ PHP сначала берет оригинальное значение, а затем увеличивает его на единицу. А при использовании выражения вида ++$a значение переменной сначала увеличивается, а затем уже используется это увеличенное значение. Заметим, что все знаки операций "+", "-" и "=" должны писаться слитно, без пробелов.

Операции декремента (уменьшения) определяются точно также - имеется пре - декремент --$a и пост - декремент $a-- . В первом случает значение переменной сначала уменьшается на единицу, а затем используется в программе скрипта. Во втором случае значение переменной сначала используется в программе, а только потом уменьшается на единицу.

Выражения сравнения

Другой очень распространенный тип выражений это выражения сравнения. Эти выражения имеют значение 0 или 1, которые обозначают "Ложь" - false или "Истину" - true, соответственно. Язык PHP поддерживает следующие выражения сравнения

> - больше, чем.
>= - больше или равно.
== - равно.

< - меньше, чем.
<= - меньше или равно.

Эти выражения, в основном, используются внутри условий, например, условного оператора IF (понятие оператора будет изложено в следующей главе).

Имеется еще одно логическое выражение, которое может показаться незнакомым, если вы не встречались с ним в других языках - это условный оператор с тремя операндами

```
$first ? $second : $third;
```

Если значение первого (first) выражения истинно (не равно 0), то исполняется второе (second) выражение и это является результатом действия данного условного оператора. В противном случае исполняется третье (thrid) выражение.

Логические выражения

Поясним еще одно понятие - это логические значения выражений. Во многих случаях, в основном в условных операторах и операторах циклов (смотрите следующую главу), вас не интересуют конкретные значения выражений, вас интересует только одно, являются ли их значения TRUE или FALSE. В PHP считается, что любое не нулевое целое значение - это TRUE, а ноль - это FALSE. Обратите внимание на то, что отрицательные значения - это не ноль и поэтому они считаются равными TRUE. Пустая строка и строка '0' это FALSE, а все остальные строки - TRUE.

Для составных типов (массивы и объекты) переменных считается, что если значение такого типа не содержит элементов, то оно считается равным FALSE, иначе подразумевается значение TRUE.

В логических выражениях чаще всего используются операторы сравнения - >, <, ==, или логические операторы - || (логическое ИЛИ), && (логическое И), ! (логическое НЕ) и так далее. Приведем пример

```
$a=10<5;
echo "First a = $a<br>";
/* $a = false, поскольку логическое условие не выполня-
```

```
ется */

    $b=1;
    $a=$b==1;
    echo "$a<br>";
    /* $a = true, если $b = 1 - в данном случае это выплня-
ется */

    $a=$b>=1&&$b<=10;
    echo "$a<br>";
    /* $a = true, если  $b находится в пределах от 1 до 10 - в
данном случае выполняется */
```

Проверка истинности любой логической переменной выполняется по обычным для логических выражений правилам. Например

```
    $a=3;
    $b=$a>=1&&$a<=10;
    // присваиваем $b значение логического выражения;
    if ($b) echo "переменная $a находится в заданном диа-
пазоне значений";
    // в данном случае условие выполняется
```

Совмещенные выражения

Вы уже знаете, что для увеличения значения переменной $a на единицу можно написать

```
$a++
```

или

```
++$a
```

Но если следует увеличить значение больше, чем на единицу, например, на 3? Конечно, можно написать $a++ несколько раз, но это не очень удобно и эффективно. Намного больше распространена запись вида

```
$a = $a + 3;
```

Здесь $a + 3 вычисляется, как значение $a плюс 3, а затем присваивается переменной $a - в результате этих действий значение $a увеличивается на 3.

В языке PHP, как и в ряде других языков типа C, вы можете записать это выражение короче, что часто бывает проще и быстрее. Добавление числа 3 к текущему значению $a может быть записано, как

```
$a+= 3;
```

Это значит следующее - возьми значение $a, добавь к нему 3 и присвой новое значение прежней переменной $a. Кроме того, что это быстрее и понятнее, такой тип выражений быстрее исполняется на компьютере. Заметим, что знак операции "+" и знак равенства "=" должны писаться слитно, без пробелов.

Любой бинарный (имеющий два операнда) оператор может быть записан таким методом, например

```
$a-= 5;
```

что означает - вычесть 5 из текущего значения $a и присвоить результат исходной переменной $a. Другой пример

```
$b*= 7;
```

умножить значение $b на 7 и присвоить новое значение прежней переменной $b.

Строковые выражения

Строки, строковые или символьные выражения в PHP - это одни из самых основных элементов языка. Они могут содержать обычный текст или некоторые другие данные. Строковое выражение в кавычках или апострофах может начинаться на одной строке и заканчиваться на другой. Например

```
$a="Этот текст начинается на одной строке
и продолжается на другой, третей и так далее, пока не закрыты кавычки";
```

Для слияния, сложения строк в PHP используется операция,

которая записывается с помощью точки ".". Это объясняется тем, что в PHP знак плюс "+" применяется только для сложения чисел. Приведем небольшой фрагмент программы

```
$a="100";
$b="200";
echo $a+$b;
// выведет на экран "300"
echo $a.$b;
//выведет на экран "100200"
```

Этим мы показали, что плюс используется только, как числовой оператор, а точка, как оператор строковый. Приведем еще один пример использования строковых переменных

```
$path="c:/windows";
$name="win";
$ext="com";
$FullPath="$path/$name.$ext";
```

Это выглядит более красиво, чем такой фрагмент

```
$path="c:/windows";
$name="win";
$ext="com";
$FullPath=$path."/".$name.".".$ext;
```

Этот пример наглядно демонстрирует, что в выражениях, записанных в двойных кавычках, вместо имен переменных, подставляются их значения.

Рассмотрим теперь более подробно строковые константы, заключенные в апострофы и в кавычки.

Строка в апострофах

Отметим, что если строка заключена в апострофы (например, 'строка'), то она трактуется так, как написана. Правда есть небольшое исключение

1. Последовательность символов \' трактуется в языке PHP, как апостроф и предназначена для вставки апострофа в строку,

заключенную в апострофы.

2. Последовательность символов \\ трактуется, как один обратный слеш и позволяет вставлять его в эту строку.

Все остальные символы обозначают сами себя, т.е. выводятся на экран так, как они написаны в программе.

Строка в кавычках

Набор специальных метасимволов в языке PHP, которые, будучи помещены в кавычки, определяют тот или иной символ, шире и богаче, чем рассмотренные только что исключения в использовании апострофов

- \n обозначает символ новой строки.
- \r обозначает символ возврата каретки.
- \t обозначает символ табуляции.
- \$ обозначает символ $, чтобы следующий за ним текст не был случайно интерпретирован, как переменная величина.
- \" обозначает кавычку.
- \xNN обозначает символ с шестнадцатеричным кодом NN.

Приведем некоторые примеры

```
$a="Hello";
echo "$a world!";
```

Указанный фрагмент программы выведет

```
Hello world!
```

Вы видите, что переменная $a, записанная в кавычках заменена, на значение этой переменной. Этому помог знак доллара, который предваряет любую переменную величину. Рассмотрим еще один пример

```
$a="Hell";
// Здесь слово Hello дано без буквы "o"
echo "$ao world!";
```

Этот пример работать правильно не будет, потому что здесь знак доллара стоит перед обозначением "ao" и интерпретатор PHP воспримет это, как новую переменную $ao. Если вы попробуете выполнить этот фрагмент программы на языке PHP, то получите сообщение о том, что переменная $ao не определена.

Правильная запись для вывода на экран будет выглядеть следующим образом

```
echo $a."o world!";
// это один способ
echo "${a}o world!";
// это другой способ
```

Таким образом, существует два способа для правильной записи этого выражения.

МАССИВЫ

Массив (array) - это многомерная переменная или переменная с индексом. Он дает возможность хранить сколько угодно значений под одним и тем же именем переменной. К каждому конкретному значению массива, необходимо обращаться с помощью числового индекса, или строкового ключа. Конечно, если вам необходимо использовать 3, 5 или 10 переменных, ничто не мешает создать их и использовать, как разные переменные.

Но, если для вашей программы понадобится 50, 100 и больше переменных, обычными переменными здесь не обойтись и придется вводить массивы. Кроме того, массив достаточно гибкая структура и позволяет эффективно и быстро обрабатывать находящиеся в нем данные.

В простейшем случае, массив это список некоторых значений одной переменной, расположенных в порядке возрастания числового индекса, т.е. массив можно представить в виде простой таблицы.

0	1	2	3	4
Вася	Маша	Дима	Лена	Андрей

Вся эта таблица - и есть массив значений переменной, которую можно назвать, например, names. Таблица состоит из ячеек с номерами от 0 до 4 и в каждой ячейке имеется свое значение. Доступ к некоторому значению или элементу массива достигается следующим образом

```
$names[индекс];
```

где индекс - это число, которое в нашем примере принимает значение от 0 до 4. Задаваться массив может несколькими способами

```
$names[0] = "Вася";
$names[1] = "Маша";
// и так все другие элементы
// или
$names = array(0=>"Вася", 1=>"Маша", "Дима", "Лена", "Андрей");
/* в этом способе необязательно писать символы 0=> ...
```

```
и 1=> ..., так как им автоматически присваиваются эти индексы в порядке возрастания */
    // имеется и еще один способ
    $names[] = "Вася";
    // нулевой элемент
    $names[] = "Маша";
    // первый элемент и т.д.
    /* если индекс не указан, то присваивается индекс, на единицу больший максимального индекса до данной операции */
```

Простые массивы

Остановимся на понятиях массива более подробно, поскольку массивы очень часто используются в программах скриптов на PHP. Массив можно создавать с помощью функции array(), как уже приводилось в предыдущем примере

```
<?
$color = array('red', 'black', 'white', 'blue', 'green');
echo "$color[0]<BR>";
// первый элемент
echo "$color[1]<BR>";
// второй элемент
echo "$color[2]<BR>";
// третий элемент
echo "$color[3]<BR>";
// четвертый элемент
echo "$color[4]<BR>";
// пятый элемент
/* теперь добавим еще один элемент - в данном случае индекс элемента явно не указан */
$color[] = 'yellow';
echo "$color[5]";
// шестой элемент
?>
```

В браузере получим

```
red
black
```

```
white
blue
green
yellow
```

В этом примере, мы открываем скобки и перечисляем список значений, при этом индекс каждого следующего элемента увеличивается на 1. Для того, чтобы добавить еще один элемент в массив используется обычная операция присваивания. При этом элемент массива можно указывать с пустыми квадратными скобками [] - тогда он автоматически получает следующий порядковый индекс, относительно последнего уже присвоенного индекса. Следует обратить внимание, что индекс первого элемента [0], а не [1].

Всегда можно явно указывать номер индекса, но совсем не обязательно, чтобы он шел по - порядку. Однако, обычно не рекомендуется использовать этот прием, так как это может привести к ошибкам при обращении к массиву.

Рассмотрим еще один пример для создания массива

```
<?
$color[] = 'red';
// первый элемент
$color[] = 'black';
// второй элемент
$color[] = 'white';
// третий элемент
$color[] = 'blue';
// четвертый элемент
$color[] = 'green';
// пятый элемент
$color[] = 'yellow';
// шестой элемент
echo "$color[0]<BR>$color[1]<BR>$color[2]<BR>";
echo "$color[3]<BR>$color[4]<BR>$color[5]";
?>
```

Этот скрипт выведет на экран браузера те же элементы

```
red
black
```

```
white
blue
green
yellow
```

Ассоциированные массивы

Ассоциированный массив (хэш) - это массив, к элементу которого можно обратиться не по числовому индексу, а по символьному имени этого индекса или ключу. Каждому ключу соответствует (=>) некоторый элемент массива, который имеет заданное вами значение. Такой массив также создается при помощи функции array()

```
<?
$style = array('color' => 'Red', 'size' => '10px');
echo $style['size'].'<BR>';
echo $style['color'];
?>
```

В браузере мы получим две строки

```
10px
Red
```

Таким образом, видно, что каждому ключу, например, color сопоставляется его значение (значение элемента массива), в данном случае это - Red. Можно добавить новый элемент, если использовать имя массива и указать новый ключ и его значение

```
<?php
$style['font family'] = 'Arial, sans - serif';
$style['bgcolor'] = 'Silver';
echo $style['font family'];
echo '<BR>'.$style['bgcolor'];
?>
```

На экране браузера будем иметь

```
Arial, sans - serif
Silver
```

Многомерные массивы

Многомерный массив - это массив, каждый элемент которого сам является массивом. Многомерный массив легко позволяет организовывать сложные структуры данных в более простую форму. Рассмотрим простой пример

```
<?php
$book = array (
/* ниже задан элемент под индексом $book[0] */
array('name' => 'Vasiy', 'family' => 'Petrov', 'phone' => '123 - 44 - 55'),
/* далее задается элемент под индексом $book[1] */
array('name' => 'Petiy', 'family' => 'Sidorov', 'phone' => '123 - 66 - 77'),
/* ниже идет элемент под индексом $book[2] */
array('name' => 'Andrey', 'family' => 'Ivanov', 'phone' => '123 - 88 - 99'));
echo $book[1]['name'].' - тлф.'.$book[1]['phone'];
?>
```

В браузере будем иметь

```
Petiy - тлф. 123 - 66 - 77
```

В следующем параграфе мы продолжим разговор о массивах, точнее о встроенных в PHP функциях, предназначенных для обработки данных, содержащихся в них.

Работа с массивами

Встроенная функция

```
count();
```

возвращает (дает в качестве результата) количество элементов, содержащихся в указанном в скобках массиве. При этом необходимо помнить, что отсчет начинается с 0, поэтому индекс последнего элемента массива всегда равен

```
count($array) - 1;
```

Если эту функцию применить к обычной переменной, будет возвращено значение 1. Еще одна функция

```
sizeof($array);
```

также возвращает число элементов массива. Встроенная функция

```
sort($array);
```

сортирует массив - все элементы, по окончании ее работы, будут расположены в порядке возрастания

```
$fruits = array("lemon","orange","banana","apple");
sort($fruits);
for($key=0; $key <= 3; $key++)
{
echo "$fruits[$key]<br>";
}
```

Этот пример выведет на экран

```
apple
banana
lemon
orange
```

Фрукты будут отсортированы по возрастанию в алфавитном порядке.

Следующая функция

```
reset($array);
```

устанавливает внутренний указатель на первом элементе массива и возвращает его значение. Для предыдущего примера она выведет на экран

```
apple
```

Функция

```
end($array);
```

устанавливает внутренний указатель массива на последнем его элементе и для предыдущего примера выведет на экран

```
orange
```

Функция

```
key($array);
```

возвращает индекс (число) элемента в текущей позиции массива, а функция

```
next($array);
```

возвращает следующий элемент массива, от текущей позиции внутреннего указателя, или false, если элементов больше нет. Для приведенного выше примера она возвратит значение

```
banana
```

Если массив содержит пустые элементы, то эта функция возвратит false и для этих элементов. Чтобы правильно просмотреть массив, который может содержать пустые элементы, нужно использовать функцию each().

Функция next() передвигает внутренний указатель массива на один элемент вперед, прежде чем возвратит значение этого элемента. Если при обращении к следующему элементу обнаружен конец массива - next() возвращает false.

Функция

```
prev($array);
```

возвращает значение предыдущего элемента массива, или false, если перед текущим, элементов больше нет. Для приведенного примера, после действия функции next(), она возвратит значение первого элемента

```
apple
```

Если массив содержит пустые элементы, то функция возвратит "Ложь" и на этих элементах. Чтобы правильно просмотреть массив, который может содержать пустые элементы, нужно использовать функцию each().

Функция prev() ведет себя подобно next(), за исключением того, что она переводит внутренний указатель массива на одну позицию назад, а не вперед.

Следующая функция

```
each($array);
```

возвращает из массива текущую пару "ключ => значение" и переводит указатель массива на следующий элемент. Эта пара возвращается в четырех - элементный массив, с ключами 0, 1 и т.д.

```
$f = array( "bob", "fred", "jussi", "jouni" );
$b = each($f);
for($k=0; $k <= 3; $k++) {echo $b[$k]." ";}
echo "<br>";
$b = each($f);
for($k=0; $k <= 3; $k++) {echo $b[$k]." ";}
echo "<br>";
$b = each($f);
for($k=0; $k <= 3; $k++) {echo $b[$k]." ";}
echo "<br>";
$b = each($f);
for($k=0; $k <= 3; $k++) {echo $b[$k]." ";}
```

Массив $b последовательно содержит следующие пары "ключ => значение"

```
0 bob
1 fred
2 jussi
3 jouni
```

Для сортировки массива в обратном порядке используется функция

```
rsort($array);
```

она сортирует массив по убыванию.

```
$fruits = array("lemon","orange","banana","apple");
rsort($fruits);
for($key=0; $key <= 3; $key++)
{
echo "fruits[$key]<br>";
}
```

Этот пример выведет

```
orange
lemon
banana
apple
```

Теперь фрукты отсортированы в обратном алфавитном порядке.

Сортировка массива в обратном порядке может также выполняться функцией

```
array_reverse($array);
```

которая меняет порядок следования записей массива на обратный, от исходного. Подобные функции очень полезны для скриптов досок объявлений, гостевых книг и т.д. - там, где необходимо первой выводить запись, которая добавлена последней.

Сортировка массива по значениям выполняется функциями

```
asort();
arsort();
```

Функция asort() сортирует значения, переданного ей в качестве параметра массива в алфавитном (если значения массива - строки) или возрастающем (если значения - числа) порядке. При этом, все связи вида "ключ => значение" сохраняются. Функция arsort() выполняет сортировку значений массива по убыванию, т.е. в обратном порядке.

Сортировка массива по ключам выполняется функциями

```
ksort();
```

krsort();

которые сортируют ключи в алфавитном порядке по возрастанию и убыванию. Приведем пример

```
<?
$echo = array("color" => "red", "font" => "Arial", "bgcolor" => "navy");
$reverse = array_reverse($echo);
/* результат работы функции array_reverse() - bgcolor => navy, font => Arial, color => red */
Output($reverse);
echo "<BR>";

asort($echo);
/* результат работы функции asort() - font => Arial, bgcolor => navy, color => red */
Output($echo);
echo "<BR>";

arsort($echo);
/* результат работы функции arsort() - color => red, bgcolor => navy, font => Arial */
Output($echo);
echo "<BR>";

ksort($echo);
/* результат работы функции ksort() - bgcolor => navy, color => red, font => Arial */
Output($echo);
echo "<BR>";

krsort($echo);
/* результат работы функции krsort() - font => Arial, color => red, bgcolor => navy */
Output($echo);

function Output($echo)
{foreach($echo as $key => $value) echo " $key => $value,";}
?>
```

На экране браузера получим

```
bgcolor => navy, font => Arial, color => red,
font => Arial, bgcolor => navy, color => red,
color => red, bgcolor => navy, font => Arial,
bgcolor => navy, color => red, font => Arial,
font => Arial, color => red, bgcolor => navy,
```

в полном соответствии с комментариями в самой программе. Обратите внимание, что при использовании функции array_reverse(), мы присваиваем результат ее работы обычной переменной. Это связано с тем, что функция, обрабатывая заданный массив, возвращает результат в виде нового массива.

Получение списка всех значений в ассоциативном массиве выполняется функцией

```
array_values($array);
```

Подсчет количества повторов каждого значения в списке

```
array_count_values($list);
```

Используя эту функцию, мы можем подсчитать, сколько раз встречается, в переданном списке, каждое из значений. Функция возвращает ассоциативный массив с ключами (элемент списка) и значениями (количество повторов). Приведем пример скрипта, использующего эти функции

```
<?
$echo = array('q'=>'a', 'w'=>'b', 'e'=>'a', 'r'=>'s', 't'=>'s');
$list = array_values($echo);
/* результат работы функции array_values() - 0 => a, 1
=> b, 2 => a, 3 => s, 4 => s */
Output($list);
echo '<BR>';

$list = array_count_values($echo);
/* результат работы функции array_count_values() - a =>
2, b => 1, s => 2 */
Output($list);
```

```
function Output($list)
{foreach($list as $key =>$value) echo " $key => $value,";}
?>
```

На экране будем иметь

```
0 => a, 1 => b, 2 => a, 3 => s, 4 => s,
a => 2, b => 1, s => 2,
```

как и было приведено в комментариях скрипта.

ОПЕРАТОРЫ

Любой PHP скрипт состоит из последовательности операторов или операций над переменными. Оператором может быть присваивание значения переменной "=", вызов функции "f()", цикл "for()", условное выражение "if()" или пустое выражение, которое ничего не делает. Операторы могут быть объединены в группу, если заключить их в фигурные скобки. Группа операторов также является оператором, который можно назвать комбинированным.

В предыдущей главе мы вплотную подошли к понятию оператора - многие выражения могут рассматриваться, как операторы. В общем случае, оператор можно представить в форме

```
выражение;
```

т.е. это некоторое выражение, за которым следует точка с запятой. Можно сказать и так, оператор - это некоторое действие над выражением, в результате которого получается новый результат.

Операторы присваивания

Одним из основных операторов является оператор присваивания, который обозначает "равенство" правой и левой частей выражения. Смысл этого действия состоит в том, что результат, полученный справа от знака "равно", записывается в память, отведенную под переменную величину, имя которой указано слева от этого знака. Например

```
$a=($b=7)+7;
```

В результате выполнения действий, указанных справа от знака "равно", будет получено число 14. Оно запишется в память, отведенную под величину $a, имя которой стоит слева от знака "=". Или

```
$a=10;
```

Простое присваивание переменной $a числа 10. Кроме оператора присвоения (присваивания), существует еще множество

комбинированных операторов

```
$a+= 4;
/* прибавить к переменной $a число 4, а результат снова присвоить переменной $a */
$a++;
/* прибавить к переменной $a число 1, а результат снова присвоить переменной $a */
```

Более подробно они будут рассмотрены в следующих параграфах.

Строковые операции

Строковых операций или операторов всего два

$a.$b.$c - слияние, сложение символьных строк или оператор конкатенации, который соединяет в одну две или более строки.
$a[n] - n - й символ символьной строки $a.

Приведем пример

```
$s="Hello";
$s=$s." world!";
```

теперь в памяти, отведенной под переменную $s, будет находиться

Hello world!

или

```
$s="Hello";
$s.=" world!";
```

теперь в памяти, отведенной под переменную $s, будет находиться

Hello world!

Другой пример

```
$a="Hello Word";
$b=$a[4];
echo "$b";
```

выведет на экран символ "o". Все остальное, что можно выполнять в языке PHP со строками, выполняют строковые функции, которые мы частично рассмотрели в предыдущей главе.

Арифметические операторы

Арифметические операторы или операции мы уже рассматривали и приведем теперь их сводную таблицу.

Пример	***Название***	***Результат***
$x + $y	Сложение	Сумма $x и $y
$x - $y	Вычитание	Разность $x и $y
$x * $y	Умножение	Произведение $x и $y
$x / $y	Деление	Частное от деления $x на $y
$x % $y	Деление по модулю	Остаток от целочисленного деления $x на $y
=	Равно	Оператор присваивания

Кроме того, для всех арифметических и строковых операций в PHP есть "комбинированные операторы". Они представляют собой сокращенный вариант записи вида

```
$x = $x + $y;
$x = $x - $y;
$x = $x / $y;
$x = $x * $y;
$x = $x % $y;
$x = $x.$text;
```

Для такой записи пробелы между именами переменных и знаками операций полностью игнорируются. При использовании

комбинированных операторов эти выражение будут выглядеть следующим образом

```
$x += $y;
$x -= $y;
$x /= $y;
$x *= $y;
$x %= $y;
$x .= $text;
```

Почти все они также были рассмотрены нами в предыдущей главе, а остальные имеют тот же самый смысл. Здесь нужно еще раз отметить, что знаки операций ("+", "-", "*" и т.д.) и знак равенства "=" должны писаться слитно, т.е. именно так, как приведено выше в тексте.

Приведем еще один пример использования таких операторов

```
<?
$text='Привет от ';
$version = 3;
$version+= 1;
// комбинированный оператор, который увеличивает значение переменной $version, равное 3, на единицу
$text.= 'PHP'.$version.'!';
/* комбинированный оператор и оператор конкатенации - сложения строк */
echo $text;
?>
```

Этот скрипт выведет в браузер

Привет от PHP4!

Операторы сравнения

Эти операции всегда имеют своим результатом либо true, либо false. Если в результате сравнения величин условие выполняется, то получается true. В противном случае имеет место false.

Частично мы уже рассматривали эти операторы в предыдущей главе. Теперь приведем сводную таблицу всех операторов сравнения.

Пример	***Название***	***Результат***
$x == $y	Равно	True, если x и y равны
$x === $y	Тождественно	True, если x и y равны и их типы совпадают
$x != $y	Не равно	True, если x и y не равны
$x !== $y	Не тождественно	True, если x и y не равны или их типы не совпадают
$x < $y	Меньше чем	True, если $x меньше чем $y
$x > $y	Больше чем	True, если $x больше чем $y
$x <= $y	Меньше или равно	True, если $x меньше или равно $y
$x >= $y	Больше или равно	True, если $x больше или равно $y

Есть еще один оператор сравнения - тернарный оператор, состоящий из двух символов "?" и ":" - вопроса и двоеточия, его мы также рассматривали в предыдущих главах, но поскольку он бывает очень удобен в некоторых выражения, остановимся на нем еще раз. Этот оператор имеет следующий синтаксис

```
(выражение_1) ? (выражение_2) : (выражение_3);
```

Если выражение_1 есть true, результатом действия этого оператора будет выражение 2, иначе выражение 3. Например

```
$text = 'Привет от ';
$v = 3;
$e = 4;
($v>$e) ? ($text='PHP') : ($text.='PHP');
echo $text;
```

Здесь, при выполнении первого условия, а оно выполняется, на экран будет выведено сообщение

```
Привет от PHP
```

Если записать оператор в несколько другом виде, поменяв первое условие

```
($v<$e) ? ($text='PHP') : ($text.='PHP');
```

то на экран будет выведено другое сообщение

```
PHP
```

Отметим, что в PHP можно сравнивать только строки или числа. Для массивов и объектов эти операции неприменимы.

Инкремент и декремент

Как мы уже говорили, в языке PHP могут использоваться следующие операторы

```
$a+=1;
$b-=1;
```

которые увеличивают или уменьшают значения переменных на единицу. Напомним, что в PHP, по аналогии с языком Си, для этих операторов введены специальные обозначения

```
$a++ - увеличение переменной $a на 1;
$a-- - уменьшение переменной $a на 1.
```

Как и в языке Си, эти операторы увеличивают или уменьшают значение переменной $a после ее использования интерпретатором PHP. Рассмотрим следующий пример

```
$a=10;
$b=$a++;
echo "a = $a, b = $b";
```

который выведет на экран

```
a = 11, b = 10
```

Как видно, вначале выполнено присвоение величине $b значения величины $a, то есть в результате этой операции величина

$b будет равна числу 10. Далее значение величины $a увеличивается на 1, и в итоге переменная $a = 11. Таким образом, вначале переменной $b присвоено текущее значение переменной $a, а затем последняя была инкрементирована (увеличена). При такой записи, инкремент производится только после выполнения выражения присваивания. То же самое будет и для оператора декремента $a--. Напомним, что такие операторы называются пост - инкремент $a++ и пост - декремент $a--.

Кроме того, существуют парные операторы для указанных выше операторов $a++ и $a--. Это операторы ++$a и --$a - пре - инкремента и пре - декремента. Например

```
$a=10;
$b=--$a;
echo "a = $a, b = $b";
```

выведет на экран

```
a = 9, b = 9
```

На практике операторы инкремента и декремента применяются очень часто, особенно, в циклах типа for. Приведем теперь сводную таблицу всех этих операторов.

Пример	***Название***	***Результат***
++$x	Пре - инкремент	Прибавляет к $x число 1, а затем возвращает новое значение $x
--$x	Пре - декремент	Вычитает из $x число 1, а затем возвращает новое значение $x
$x++	Пост - инкремент	Возвращает значение $x, а затем прибавляет к нему число 1
$x--	Пост - декремент	Возвращает $x, а затем вычитает из него число 1

Логические операторы

Здесь очень важно понимать, что эти операторы работают только с логическими значениями, которых всего два - true и false. Ниже приведена сводная таблица таких операторов, которые мы уже частично рассматривали в предыдущих главах.

Пример	***Название***	***Результат***
$x and $y	И	True, если оба выражения имеют значения true
$x or $y	ИЛИ	True, если хотя - бы одно из выражений имеет значение true
$x xor $y	Исключающее ИЛИ	True, если ТОЛЬКО ОДНО из выражений имеет значение true
! $x	НЕ	True, если $x имеет значение false
$x && $y	И	True, если оба выражения имеют значения true
$x || $y	ИЛИ	True, если хотя - бы одно из выражений имеет значение true

Следует обратить внимание, что операторы И и ИЛИ имеют два вида записи. Дело в том, что каждый из них имеет различный приоритет при построении сложных условных выражений, что в свою очередь, позволяет обойтись без лишних скобок.

Операторы or и and имеют приоритет ниже, чем && и ||. Это значит, что если вам встретилась строка, содержащая и тот и другой оператор, то сначала сравнивается условие && и ||, а только затем or и and.

Вычисление логических выражений выполняется всегда слева направо. Но, если в выражении присутствует false или "0" и &&, то это всегда дает только false, поэтому дальнейшие вычисления обрываются. Например

```
$logics = 0 && (time() > 100);
```

При такой записи выражения стандартная функция time()

вызываться никогда не будет и при выводе на экран функцией echo величины $logics мы получим просто "0".

Битовые операции

Эти операции или операторы позволяют устанавливать, снимать или проверять некоторые группы битов в целой переменной. Биты целого числа - это не что иное, как отдельные разряды числа, записанные в двоичной системе счисления. Например, в двоичной системе число 12 будет выглядеть, как 1100, а число 2, как 10.

Если переменная нецелая, то вначале к ней применяется округление, а затем выполняются битовые операции. Имеются следующие битовые операции

- a & b - операция И - результат операции это число, у которого установлены только те биты, которые установлены у числа "a" и у числа "b" одновременно.
- a | b - операция ИЛИ - результат операции это число, у которого установлены только те биты, которые установлены или в числе "a", или в числе "b".
- -a - результат операции это число, у которого на месте 1 стоит 0, а на месте 0 стоит 1.
- a << b - операция "сдвиг влево" - результат операции это число, полученное поразрядным сдвигом "a" на "b" битов влево.
- a >> b - операция "сдвиг вправо" - аналогично предыдущему случаю, только сдвиг не влево, а вправо.

Например, выражение 12 | 2 вернет на экран число 14 (1110 в двоичной системе счисления)

```
$a=12 | 2;
echo "$a";
```

Операции эквивалентности

В языке PHP4 есть и тройной знак сравнения ===. Это оператор проверки переменных на эквивалентность, т.е. совпадать должны не только их значения, но и типы этих переменных. Рассмотрим для примера следующий фрагмент программы

```
$a = 10;
```

```
$b = "10";
if ($a==$b)
{echo "a и b равны";}
else
{echo "a и b не равны";}
```

Обратите внимание на то, что первая переменная $a представляет собой число, а вторая переменная строку (текст). При таком сравнении строка преобразуется в число, и в результате работы скрипта получим

```
a и b равны
```

Рассмотрим другой фрагмент

```
$a=0;
// ноль
$b="";
// пустая строка
if ($a==$b)
{echo "a и b равны";}
else
{echo "a и b не равны";}
```

В результате его работы также получим

```
a и b равны
```

хотя видно, что $a и $b явно не равны и даже имеют разные типы. Но при таком сравнении пустая строка преобразуется в ноль, который затем сравнивается с числовым нулем.

Подобной ситуации можно избежать благодаря оператору эквивалентности "===". Он не только сравнивает величину двух выражений, но и сравнивает их типы. Покажем тот же фрагмент программы, но уже с использованием этого оператора

```
$a = 0;
// ноль
$b ="";
// пустая строка
if ($a===$b)
```

```
{echo "a и b равны";}
else
{echo "a и b не равны";}
```

Вот теперь мы получим правильный результат

```
a и b не равны
```

Для первого, приведенного выше примера, результат будет тем же самым.

Возможности этого оператора далеко выходят за рамки простого сравнения. С его помощью можно сравнивать массивы, объекты и некоторые другие величины.

КОНСТРУКЦИИ

В этой главе мы познакомимся со средствами, позволяющими управлять ходом выполнения программы в зависимости от тех или иных обстоятельств или условий.

Условная конструкция if

Конструкция или оператор IF это одна из важнейших возможностей многих языков программирования, включая и язык PHP. Она позволяет организовать выполнение фрагмента кода по некоторому условию. Если это условие истинно, то выполняется некоторая группа выражений, если нет, это группа пропускается, и выполняются выражения, стоящие сразу за этой группой

```
if ( условие )
{
/* если значение "условия" равно true (истина) выполняем код программы, находящийся в фигурных скобках. Иначе, блок в скобках пропускается, и выполняется код, стоящий за этими скобками */
}
```

Условием может быть любое выражение, способное возвращать значения true (Истина) или false (Ложь). Чаще всего в условии используются операторы сравнения, например, $a>0. В таблице представлены другие возможные операторы сравнения.

Операторы	***Значение***
==	Проверка на равенство
!= либо < >	Не равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

В операторе if может находиться несколько условий сравнения, которые отделяются логическими операторами. Ниже приведена таблица возможных логических операторов.

Операторы	***Значение***
&& или AND	Условие верно, если два выражения верны
XOR	Условие верно, если только одно выражение верно
|| или OR	Условие верно, если хотя бы одно выражение верно
! *(например, !$a)*	Условие верно, если выражение неверно

Заметим, что условием может быть любая функция, возвращающая true в случае ее успешного выполнения.

Очень часто требуется выполнить одно выражение, если соблюдается какое - либо условие и другое выражение в противном случае. Для этого применяется блок операторов if - else, который расширяет возможности if по части обработки вариантов выражений

```
if (условие )
{
// этот код будет выполняться, если "условие" равно true - истинно
}
else
{
// этот код будет выполняться, если "условие" равно false - ложно
}
```

Если необходимо проверить несколько условий, можно воспользоваться блоком elseif. Elseif, как и else, позволяет выполнить выражение, если значение if равно false, но в отличие от else оно выполнится, только если выражение elseif равно true

```
if ( условие 1 )
{
// выполняется, если "условие 1" является равно true - истинно
}
elseif ( условие 2 )
{
```

```
// выполняется, если "условие 2" равно true - истинно
}
else
{
// выполняется, если "условие 1" и "условие 2" равно false - ложно
}
```

Внутри одного выражения if может быть несколько elseif, но только первое выражение elseif, которое окажется равно true, будет выполнено. Таким образом, выражение elseif будет выполнено только в том случае, если выражение if и все предыдущие elseif равны false, а данный elseif равен true.

Приведем простой пример

```
<?
$x = 5;
$y = 10;
if ( $x > $y )
{
echo '$x больше $y';
}
elseif ( $x == 0 )
{
echo '$x равно нулю';
}
else
{
echo '$x меньше $y';
}
?>
```

Этот код выведет в окно браузера

```
$x меньше $y
```

Приведем еще один пример

```
$a=7;
$b=2;
if ( $a > 3 && $a <= 20 && $a != 5)
```

```
{
$result = $a*$b ;
echo "Умножаем a на b = $result";
}
elseif ( ($a > 20 || $a < 3 ) && $b != 0)
{
$result = $a / $b;
echo "Делим a на b = $result";
}
else
echo "Invalid number";
```

Скрипт выдаст на экран

```
Умножаем a на b = $result
```

При начальных условиях

```
$a=1;
$b=0;
```

Получим

```
Invalid number
```

Рассмотрим еще одну интересную конструкцию языка PHP, которая предлагает иной путь для группировки некоторых операторов вместе с оператором if. Такая конструкция наиболее часто используется, когда нужно внедрить блоки HTML внутрь оператора if, но, конечно, может использоваться и в любом месте PHP программы.

Вместо использования фигурных скобок вслед за оператором if, можно поставить двоеточие, одно или несколько выражений, и завершающий оператор endif

```
if():
...
...
...
endif;
```

Смысл ее в том, что если условие, записанное в круглых скобах оператора if, оказалось истинным, то будет выполняться весь код, начиная от двоеточия до команды endif;

Приведем пример использования такой конструкции

```
<?
$a=5;
if ($a==5):
?>
<font size=3 color=red>Пример использования операторов If - endif
<?
endif;
?>
```

На экране получим сообщение, выведенное красным цветом

Пример использования операторов If - endif

В этом примере блок

```
<font size=3 color=red>
```

внедрен внутрь выражения if, используемого альтернативным способом. Этот HTML блок будет виден в HTML документе, только в том случае, если переменная $a равна 5. Усложним этот пример

```
$a=6;
if ($a==5):
?>
<font size=3 color=red>Пример использования операторов If - endif
<?
else:
?>
<font size=3 color=blue>Пример использования операторов If - endif
<?
endif;
```

В этом случае на экран будет выведен тот же текст, только цвет его будет синим.

Этот альтернативный синтаксис записи оператора if применим и к операторам else и elseif. Приведем пример с использованием всех условных операторов if, else, elseif и endif

```
$a = 7;
if ($a == 5):
print "a is 5";
print "...";
elseif ($a == 6):
print "a is 6";
print "!!!";
else:
print "a is not 5 or 6";
endif;
```

На экране будем иметь

```
a is not 5 or 6
```

Условная инструкция switch

Во многих случаях бывает нужно сравнить переменную (или выражение) со многими различными значениями и выполнить различные фрагменты кода в зависимости от того, чему будет равно значение выражения. Это как раз то, для чего предназначается оператор switch

```
switch ( выражение )
{
case значение_1:
// выполняется, если выражение = значение_1
break;
case значение_2:
// выполняется, если выражение = значение_2
break;
case значение_3:
// выполняется, если выражение = значение_3
break;
default:
```

```
/* необязательный параметр - выполняется, если выражение не равно ни одному из значений */
 break;
}
```

Выражение внутри скобок должно быть простого типа - целое, строка или true и false. В каждом из блоков case выражение сравнивается с текущим значением этого оператора case. Если они совпадают, то выполняется блок операторов стоящих между данным case и закрывающим оператором break.

Приведем пример использования такого оператора

```
<?
$text = 'php';
switch ($text)
{
case 'javascript':
echo 'Это javascript';
break;
case 'perl':
echo 'Это perl';
break;
case 'php':
echo 'Это php';
break;
default:
echo 'Нет совпадений';
break;
}
?>
```

В результате его работы на экране браузера получим сообщение

```
Это php
```

Инструкция break прерывает выполнение всего блока switch, поэтому выполняется только первый из операторов case, в котором выражение совпало с его значением. Это хорошо демонстрирует следующий пример

```
$text = 'php';
switch ($text)
{
case 'php':
echo 'Это php-1';
break;
case 'perl':
echo 'Это perl';
break;
case 'php':
echo 'Это php-2';
break;
default:
echo 'Нет совпадений';
break;
}
```

На экран будет выведено

```
Это php-1
```

Следующие два примера это два различных способа достижения одной цели, но один использует серию операторов if, а другой, оператор switch.
Пример 1.

```
if ($i == 0)
{
 print "i is 0";
}
if ($i == 1)
{
 print "i is 1";
}
if ($i == 2)
{
 print "i is 2";
}
```

Пример 2.

```
switch ($i)
{
 case 0:
 print "i is 0";
 break;
 case 1:
 print "i is 1";
 break;
 case 2:
 print "i is 2";
 break;
}
```

Цикл while

Для многократного выполнения фрагмента программы служат циклы. Обычно цикл выполняется до наступления некоторого заранее определенного условия.

Имеется несколько операторов цикла и первым из них мы рассмотрим оператор while, смысл которого наиболее прост - он предписывает интерпретатору PHP выполнять вложенные в цикл операторы, до тех пор, пока некоторое "выражение" равно true. Значение "выражения" проверяется каждый раз в начале цикла, так что если значение выражения изменится внутри цикла и станет false, он не прервется до конца текущей итерации (выполнение всего блока вложенных операторов - это одна итерация). Но, если значение "выражения" равно false в самом начала цикла, он выполняться не будет

```
while (выражение)
{
// тело цикла, в котором команды выполняются пока "выражение" равно true
}
```

Рассмотрим простейший пример использования такого оператора

```
<?php
$count = 1;
while ($count < 4)
```

```
{
echo "$count умножить на 3 будет ".($count*3).'<BR>';
$count++;
}
?>
```

который выведет в браузер

```
1 умножить на 3 будет 3
2 умножить на 3 будет 6
3 умножить на 3 будет 9
```

Здесь в начале цикла проверяется истинность условия и если оно true, выполняется все тело цикла.

Как и в операторе if, вы можете сгруппировать несколько операторов внутри фигурных скобок или использовать альтернативный синтаксис

```
while (выражение):
// тело цикла
endwhile;
```

Следующие два примера выводят на экран цифры с 1 по 10. Пример 1.

```
$i = 1;
while ($i <= 10)
{
print $i++;
}
```

Пример 2

```
$i = 1;
while ($i <= 10):
print $i;
$i++;
endwhile;
```

Цифры будут выводиться без пробелов в одну строку.

Цикл do... while

Цикл do..while очень похож на while за исключением того, что значение логического выражения проверяется не до, а после окончания итерации - тела цикла. Основное отличие в том, что do..while гарантировано выполняется хотя бы один раз, что в случае while не обязательно.

Для циклов do..while существует только один вид синтаксиса

```
do
{
// тело цикла
}
while (выражение)
```

Приведем простой пример

```
$i = 0;
do
{
print $i;
}
while ($i>0);
```

Этот цикл выполнится только один раз, так как после окончания итерации будет проверено значение логического выражения, а оно равно false ($i не больше 0), и выполнение цикла завершится.

Еще один пример

```
<?php
$count = 5;
do
{
echo "$count умножить на 3 будет ".($count*3).'<BR>';
$count++;
}
while ($count < 2);
?>
```

Выведет на экран браузера

```
5 умножить на 3 будет 15
```

И здесь вначале выполняется тело цикла, а затем уже проверяется истинность условие. Поэтому, даже если у вас изначально неверное условие (как в данном случае), цикл все равно выполнится один раз.

Цикл for

В общем, все, что можно сделать с помощью этого цикла, можно сделать и с помощью while. Но он имеет несколько другой синтаксис, который записывается в одной строке, и в итоге мы имеем более простой код и не забудем инициализировать начальное значение счетчика цикла

```
for (выражение 1 или инициализация; выражение 2 или условие; выражение 3 или шаг цикла)
{
// тело цикла
}
```

Первое выражение или "инициализация" обязательно вычисляется в начале цикла. В начале каждой итерации вычисляется "условие" и если оно равно true, цикл продолжается, и выполняются все вложенные операторы или тело цикла. Если "условие" равно false, цикл заканчивается, и тело цикла не выполняется. В конце каждой итерации вычисляется "шаг", с которым изменяется счетчик цикла.

Каждое из этих трех выражений может быть пустым. Например, если "условие" пусто, то цикл продолжается бесконечно. Это не так бесполезно, как могло бы показаться, так как зачастую требуется закончить выполнение цикла, используя оператор break, в сочетании с логическим условием, вместо применения логического выражения в операторе for.

В таком цикле часто используются выражения следующих типов.

Выражения	***Эквивалент***
$a += 2 или $a -= 3	$a = $a + 2 или $a = $a - 3
$a *= 10 или $a /=5	$a = $a * 10 или $a = $a / 5
$a++ или $a--	$a = $a + 1 или $a= $a - 1

++$a или --$a	$a = $a + 1 или $a= $a - 1

Отличие последних двух выражений заключается в том, что при использовании выражения типа $a++ PHP сначала берет оригинальное значение, а затем увеличивает его на единицу. А при использовании выражения вида ++$a значение переменной сначала увеличивается, а затем уже берется это увеличенное значение.

Приведем пример использования такого оператора цикла

```
<?
for ($count = 5; $count > 2; $count--)
{
echo "$count умножить на 2 будет ".($count*2).'<BR>';
}
?>
```

который выведет в браузер

```
5 умножить на 2 будет 10
4 умножить на 2 будет 8
3 умножить на 2 будет 6
```

Здесь начальное значение счетчика равно 5 и цикл будет выполняться пока значение счетчика больше 2, а сам счетчик уменьшается при каждой итерации (один проход цикла) на единицу.

Язык PHP поддерживает также альтернативный синтаксис оператора for

```
for (выражение 1; выражение 2; выражение 1):
// тело цикла
endfor;
```

И в заключение, приведем еще один оператор цикла, который был добавлен только в четвертой версии PHP, и служит для последовательного перебора элементов массива

```
foreach ($array as $key => $value)
{
//тело цикла
```

```
}
```

Например, его можно использовать для вывода всех элементов массива

```
$a=array('1'=>'a', '2'=>'b', '3'=>'c');
foreach ($a as $key => $value)
{
print "$a[$key]<br>";
}
```

Тогда на экране в столбик получим

```
a
b
c
```

Дополнительные операторы

Теперь поговорим о том, как можно прервать цикл до конца его завершения, и как пропустить итерацию цикла. Иногда возникает ситуация, в которой нужно прервать выполнение цикла и продолжить работу основной программы, не дожидаясь завершения цикла.

Для этого служит инструкция (оператор, команда или встроенная функция)

```
break;
```

которая может прервать любой цикл и отдать управление оператору, следующему за оператором окончания конструкции цикла. Мы уже использовали этот оператор, но полностью не объясняли его действие.

Приведем несколько примеров

```
for ($i = 1;;$i++)
{
if ($i > 10)
{
break;
}
```

```
print $i;
}
```

и

```
$i = 1;
for (;;)
{
if ($i > 10)
{
break;
}
print $i;
$i++;
}
```

В обоих случаях на экран будет выдана последовательность чисел от 1 до 10

```
12345678910
```

Имеется еще одна полезная возможность записи оператора цикла

```
for ($i = 1; $i <= 10; print $i, $i++);
```

которая приведет к такому же результату. Возможна ситуация, когда мы используем несколько вложенных друг в друга циклов и хотим выйти сразу из нескольких. Тогда нам необходимо использовать эту же инструкцию, только с числовым параметром, который будет указывать количество циклов, из которых нам необходимо выйти

```
break(2);
```

Кроме принудительного выхода из цикла нам может понадобиться пропустить одну или несколько итераций. В этом случае нам поможет инструкция (оператор)

```
continue;
```

которая передает управление на начало следующей итерации цикла. Данную функцию очень удобно использовать в циклах - фильтрах, в которых из множества данных, отбираются и обрабатываются только те, которые удовлетворяют некоторому условию.

Приведем пример

```
while (выражение)
{
if (выражение 2)
continue;
действие;
};
```

Такая программа аналогична следующей, но без использования оператора continue

```
while (выражение)
{
if (! выражение 2)
действие;
};
```

Имеется еще один способ принудительного выхода из цикла, однако, его можно применять только в функциях. Делается это с помощью инструкции

```
return();
```

Например, цикл, расположенный внутри некоторой функции, находит определенное значение из перечня числовых или символьных данных и эта функция должна вернуть найденное значение в вызывающую, основную программу. Тогда в цикле можно записать проверку некоторых условий, и когда найденное выражение совпадает с заданным, функция return() возвращает это значение в вызывающую программу. При этом цикл завершается, вместе с завершением выполнения самой функции.

Рассмотрим еще два оператора, первый из которых

```
require 'file name';
```

Этот оператор позволяет включить в данный скрипт некоторый файл, имя которого указано в кавычках. В файле, включаемом оператором require, обычно хранятся какие - то данные, необходимые для работы многих скриптов и включать его можно в любой из них.

Таким образом, оператор require заменяет себя содержимым указанного файла. Это значит, что вы не можете поместить require внутрь цикла и ждать, что в процессе каждой итерации он включит содержимое другого файла несколько раз.

Следующий оператор

```
include 'file name';
```

позволяет включать и выполнять код, содержащийся в указанном файле и выполнять его столько раз, сколько программа встречает этот оператор.

Например, в файле func.inc у нас хранится программа вывода на экран определенных параметров. Каждый раз, когда нам нужно будет выводить эти параметры, мы будем вставлять в текст нашей основной программы

```
include 'func.inc'
```

т.е. это в принципе то же самое, как если - бы мы везде в таких случаях вставили - бы текст, содержащийся в файле func.inc

Для включения в основную программу нескольких файлов можно включить этот оператор внутрь цикла

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++)
{
 include($files[$i]);
}
```

Оператор include отличается от require тем, что include выполняется каждый раз при его включении в основную программу, а require заменяется на содержимое указанного файла независимо от того, будет ли выполнено его содержимое или нет

Так как include это специальный оператор, требуется заключать его в фигурные скобки при использовании внутри условного оператора. Приведем пример неправильной записи этого опера-

тора

```
if (выражение)
include($file);
else
include($other);
```

А эта запись правильная и все будет работать, как нужно

```
if (выражение)
{include($file);}
else
{include($other);}
```

Имеется очень полезный оператор (функция), который в любом месте скрипта позволяет закончить его выполнение

```
exit;
```

Эта функция, завершая текущий скрипт, не возвращает никаких значений, но может выводить сообщение

```
exit("текст сообщения");
```

И в заключение приведем еще одну функцию (оператор), которая выполняется при невыполнении некоторых действий, т.е. в том случае, когда эти действия привели к результату false. Рассмотрим пример, в котором, по некоторым причинам, невозможно выполнение некоторой функции f()

```
f() or die("текст сообщения");
```

Если функция не может быть выполнена и способна возвратить false, то будет выполняться оператор die() с заданным сообщением.

ДАТА И ВРЕМЯ

В языке PHP достаточно хорошо реализованы инструментальные средства для работы с датами и временем. Встроенная функция time() - предоставляет всю необходимую информацию о текущей дате. Она возвращает целочисленное значение, которое, на первый взгляд никак не похоже на дату. Однако, это именно текущая дата

```
<?
echo time();
?>
```

Результат будет примерно таким

```
1083471052
```

Дело в том, что данное число представляет собой количество секунд, прошедших от, так называемой, "точки UNIX" (12 часов ночи 1 января 1970г.). Может показаться, что подобное представление не очень удобно, но, на самом деле, можно получить достаточно большой объем информации, всего лишь, из одного этого значения.

Для того, чтобы представить некоторую дату в привычном формате, используется функция getdate(), которая в результате своей работы возвращает ассоциированный массив со следующими характеристиками.

Параметр	***Описание***	***Пример***
seconds	Секунды после целой минуты (0 - 59)	46
minutes	Минуты после целого часа (0 - 59)	34
hours	Часы с начала суток (0 - 23)	13
mday	День месяца (1 - 31)	18
wday	День недели (0 - 6)	4
mon	Месяц (1 - 12)	2
year	Год (4 разряда)	2003
yday	День года (0 - 365)	157
weekday	День недели (название)	Thursday

month	Месяц в году (название)	May
0	Абсолютное время	1044299626

Таким образом, данная функция удобна, если необходимо получить некоторую информацию о неопределенной дате.

Если нам нужно просто вывести текущую дату в приемлемом для восприятия виде, целесообразнее воспользоваться функцией date(), которая возвращает отформатированную строку. Форматом вывода можно управлять при помощи следующих аргументов, которые необходимо задать этой функции.

Параметр	***Описание***	***Пример***
a	am/pm, нижний регистр	am
A	AM/PM, верхний регистр	PM
d	День месяца	20
D	День недели (первые три буквы)	Thu
F	Название месяца	December
h	Час, 12 - часовой формат, дополнение нулями	03
H	Час, 24 - часовой формат, дополнение нулями	09
g	Час, 12 - часовой формат, без дополнения нулями	6
G	Час, 24 - часовой формат, без дополнения нулями	9
i	Минуты	47
j	День месяца	18
l	День недели (название)	Thursday
L	Високосный год (1 - да, 0 - нет)	0
m	Месяц года, число, дополненное нулями	02
M	Месяц года, три первые буквы	Jan
n	Месяц года, число без дополнения нулями	3

s	Секунды	29
U	Абсолютное время	1044299626
y	Год, два разряда	03
Y	Год, четыре разряда	2003
z	День года (0 - 365)	237
Z	Время в сек. от	GMT 0

Рассмотрим пример

```
<?
echo date("Сегодня: m/d/y").'<BR>';
echo date("Сегодня: d.m.Y, сейчас: H:i:s").'<BR>';
echo date("Это: z день с начала Y года");
?>
```

который выведет в браузер

```
Сегодня: 05/02/04
Сегодня: 02.05.2004, сейчас: 11:15:31
Это: 122 день с начала 2004 года
```

Это уже полная информация о дате и времени в привычном для нас формате.

СТРОКИ

Одной из наиболее часто встречающихся задач является обработка символьных последовательностей или строк. Очень часто возникает необходимость в обработке поступающих от HTML формы или взятых из файла данных и при этом практически всегда заранее неизвестно, что именно там находится и в каком виде. Поэтому прежде, чем производить какие - либо действия с полученными данными, необходимо их проверить.

Функции trim()

Первое, что бывает нужно сделать, это удалить лишние пробелы. Для этого в PHP есть специальная функция trim(), которая обрезает пробелы в начале и в конце строки и возвращает обрезанную строку.

Пример записи

```
$str = trim($str);
```

В результате, обработанное значение строки $str не будет содержать повторяющихся начальных и конечных пробелов. Эта функция, кроме пробелов удаляет еще некоторые символы - табуляция, вертикальная табуляция, символ перевода строки, символ возврата каретки и нулевой байт.

Имеются еще и функции rtrim() и ltrim() - первая удаляет повторяющиеся пробелы в конце строки, а вторая в начале.

Функции обрезки строк

Имеется еще одна возможность при работе со строками - их обрезка. Она часто применяется при обработке HTML - форм для ввода данных. Представьте себе, что кто - нибудь из ваших "гостей" введет в вашу гостевую книгу текст всей этой книги. Вот для этого и нужно ограничить количество вводимых символов в любом поле формы, тем более, что делается это очень просто. Для начала нужно прописать ограничение для поля в самой форме, например

```
<input type="text" maxlength="50" name="user">
```

Теперь поле с именем user ограничено в количестве вводимых символов - 50. Но, дело в том, что обойти такое ограничение не слишком сложно, и нужно оно скорее для того, чтобы показать посетителю предел числа символов. Дальше нужно воспользоваться функцией PHP substr()

```
$user = substr($user,0,30);
```

Этим вы просто отрезаете часть полученной строки, превышающую 30 символов (начиная с нулевого символа, вырезается тридцать символов). Теперь все попытки завалить вас ненужной информацией будут блокированы, так как этот скрипт никогда не пропустит больше заданного числа символов.

Собственно говоря, у функции

```
substr(string, start, length)
```

более общее назначение. Она возвращает часть строки string, определяемую параметрами start (начало) и length (длина). Если параметр start положительный, то возвращаемая строка будет начинаться со start - ного символа строки string длинной length.

Приведем пример использования этой функции

```
<?
$user = substr('Ivanov', 1);
echo $user."<br>";
/* вернет vanov - т.е. от первого символа и до конца строки */
$user = substr('Ivanov', 1, 3);
echo $user."<br>";
// вернет van - т.е. 1,2,3 символы
$user = substr('Ivanov', -1);
echo $user."<br>";
// вернет v - т.е. один символ от конца строки
$user = substr('Ivanov', -2);
echo $user."<br>";
// вернет ov - т.е. два символа от конца строки
$user = substr('Ivanov', -3, 1);
echo $user."<br>";
/* вернет n - т.е. один символ, который стоит третьим от конца строки */
```

```
?>
```

На экране будем иметь

```
vanov
van
v
ov
n
```

Как видите, если параметр start отрицательный, то возвращаемая строка будет начинаться со start - ного символа от конца строки string.

Функции замены и перекодировка

При обработке данных формы очень важно уметь вырезать из полученной строки лишние или просто недопустимые символы. В этом случае можно применить специальную функцию, которая заменяет все вхождения строки needle в строке haystack на указанную строку str.

Эта функция записывается в виде

```
str_replace(needle, str, haystack)
```

Если не требуется производить какие - либо сложные замены, то всегда используйте эту функцию вместо ereg_replace() (которая работает несколько медленнее - о ней мы поговорим позже).

Приведем пример

```
<?
$name = 'Ivanov Ivan Ivanovich';
$new = str_replace('Ivan','Petr',$name);
echo $new."<br>";
// вернет Petrov Petr Petrovich

$name = 'Иванов Иван Иванович';
$new = str_replace('Иван','Петр',$name);
echo $new."<br>";
// вернет Петров Петр Петрович
```

```
$url = 'http://www.yandex.ru';
$new = str_replace('http://','',$url);
echo $new."<br>";
// вернет www.yandex.ru

$font = '<FONT COLOR="RED">RED</FONT>';
$new = str_replace('RED','BLUE',$font);
echo $new."<br>";
/* вернет <FONT COLOR="BLUE">BLUE</FONT> и на экране шрифт будет синим */
?>
```

На экране будем иметь

```
Petrov Petr Petrovich
Петров Петр Петрович
www.yandex.ru
BLUE - синего цвета
```

Начиная с версии PHP 4.0.5 в качестве любого из параметров функции str_replace() можно использовать массив, что значительно расширяет возможности данной функции.

Еще одна важная и полезная функция strlen() - нахождение длины строки. Приведем простой пример

```
<?
$str = strlen('В этом выражении около ста символов и было бы интересно узнать, сколько – же их на самом деле?');
echo "$str";
?>
```

На экране получим

```
95
```

В переменной $str будет содержаться число 95, так как длина строки именно 95 символов.

Еще одна интересная возможность языка PHP - перевод текста из одной кодировки в другую. Это очень полезно, если нужно

согласовать кодировки, например, сайта и почтовой программы. Причем поддерживаются самые распространенные русские кодировки

```
convert_cyr_string(string, from, to)
```

Аргументы from и to являются одним символом, который определяет исходную и конечную кодовую таблицу. Поддерживаются следующие типы кодировок

- k-koi8-r
- w-windows-1251
- i-iso8859-5
- a-x-cp866
- d-x-cp866
- m-x-mac-cyrillic

Функция перекодировки

Для полной обработки строки применяют функцию

```
htmlentities();
```

Она переводит все возможные символы в коды языка HTML. Эта функция заменяет все символы, которые имеют соответствующий HTML код на этот HTML код. В настоящее время применяется только кодовая таблица ISO - 8859 - 1.

Функция поиска в строке

Функция strstr находит первое появление заданного символа

```
strstr(haystack, needle);
```

При ее работе возвращается часть строки haystack от первого вхождения needle до конца haystack. Если строка needle не найдена, возвращает false. Если параметр needle не является строкой, то он переводится в целое число и рассматривается, как числовое значение символа.

Эта функция чувствительна к регистру, поэтому если необходимо найти символ без учета регистра используйте функцию

stristr(), синтаксис которой полностью идентичен strstr().

Приведем пример

```
<?
$url = 'HTTP://WWW.YANDEX.RU';
// задаем домен
$domain = stristr($url,'www');
echo $domain;
?>
```

На экране получим

```
WWW.YANDEX.RU
```

Среди стандартных функций имеется и функция разбивки строки на подстроки

```
explode(separator, string [,limit]);
```

которая разбивает строку на массив, используя разделитель separator. В результате возвращается массив строк, каждая из которых является подстрокой строки string, и сформирована путем разделения строки по границам, образованным сепаратором строки separator.

Если limit установлен, возвращаемый массив будет содержать максимум элементов limit с последним элементом, содержащим остаток string. Если возникла ошибка, explode(), как и большинство функций, возвращает false.

Если разделитель separator не найден в искомой строке, функция возвратит массив со строкой, в качестве единственного элемента.

Рассмотрим пример

```
<?
$string = 'У нас есть некоторая строка, которая содержит нужную информацию';
$array = explode(" ",$string);
/* разбиваем строку на отдельные слова и заполняем ими массив */
foreach($array as $value)
echo $value.'<BR>';
```

```
// выведет в столбик все слова нашей строки
?>
```

На экране увидим

```
У
нас
есть
некоторая
строка,
которая
содержит
нужную
информацию
```

Если вам необходимо наоборот, собрать массив в строку, используйте функцию

```
implode(separator, array).
```

Синтаксис и параметры этой функции подобны предыдущей. Пример

```
<?
$array = array('Это', 'то', 'же', 'полезная', 'функция', '!');
$string = implode(" ",$array);
// отдельные слова из массива собираем в строку с помощью пробела
echo $string;
?>
```

выведен на экран

```
Это то же полезная функция !
```

Еще одна интересная функция

```
substr_replace (string, replace, start [, length]);
```

заменяет текст части строки. Эта функция замещает копию строки string, ограниченную параметрами start и length строкой, заданной в параметре replace. Если start положительный, замеще-

ние начинается со start - вого смещения с начала строки string. Если start отрицательный, замещение начинается со смещения start - вого символа от конца строки string.

Если задан положительный length, он представляет длину замещаемой части строки string. Если он отрицательный, он представляет количество символов от конца строки string, с которых замещение останавливается. Если он не задан, то по умолчанию будет strlen, т.е. конец замещения - в конце string.

Рассмотрим пример

```
<?
$string = "http:--www.rambler.ru";
// замена с отсчетом от начала
echo substr_replace($string, '//', 5, 2);
// получим http://www.rambler.ru
// замена с отсчетом от конца
echo substr_replace($string, '//', - 16, 2);
// также получим http://www.rambler.ru
?>
```

На экране увидим

```
http://www.rambler.ru
http://www.rambler.ru
```

В этой главе мы рассмотрели далеко не полный перечень строковых функций. Были приведены примеры только самых основных и наиболее употребляемых при реальном программировании возможностей. Далее, по мере изложения материала, мы будем приводить и другие подобные функции.

ФУНКЦИИ

Язык PHP, как и практически любой современный язык программирования, поддерживает конструкции, которые называются функциями. Функции отличаются от других конструкций языка скобками (), между которыми могут присутствовать некоторые аргументы.

Таким образом, функция - это заранее определенный блок инструкций или кодов, который не выполняется немедленно, но может быть вызван основной программой на исполнение в любое время. Функции позволяют автоматизировать многие процессы, и, пожалуй, именно в этом заключается их основное достоинство.

В PHP существует два типа функций:

- Зарезервированные, внутренние, встроенные в интерпретатор PHP функции - стандартные функции.
- Функции, определяемые пользователем в ходе написания программы - пользовательские функции.

С первым типом функций мы уже сталкивались, например, когда упоминали функции print() или echo(). Эти функции уже заранее разработаны производителем, и мы можем их использовать. Они имеют самое разнообразное применение - это и математические функции, и функции работы со строками, массивами, файлами, временем и т.д.

Пользовательские функции

Больше возможностей, а значит и проблем, предоставляют функции, определяемые пользователем. Эти функции разрабатываются самим программистом в процессе написания кода скрипта. Они могут быть самыми разнообразными по сложности и разноплановыми по применению. Особенность этих функций состоит в том, что их нужно самостоятельно определять, в отличие от зарезервированных функций.

Приведем теперь синтаксис определения таких функций в общем виде

```
function имя_функции(арг1, арг2, арг3, и так далее)
{
// тело функции - некоторые операции, выполняемые
```

при ее вызове
}

Имя функции является оригинальным идентификатором и может содержать все буквы латинского алфавита, числа и знак подчеркивания. Следите, чтобы имя вашей функции не совпадало с зарезервированной функцией PHP, иначе программа выведет ошибку. Заметим, что имя функции, в отличие от имени переменной, нечувствительно к регистру.

Аргументами функций могут являться переменные и/или константы. Это входные значения, которые будут использоваться командами в теле функции. Внутри функции может быть любой PHP код, включая другие функции обоих типов. Использование функций, определенных пользователем, полностью идентично использованию зарезервированных функций.

Допустим, у нас есть некая программа, которая многократно открывает файл, читает его содержимое, производит какие - либо действия с этим содержимым, закрывает его и так много раз подряд, причем, все это должно осуществляться на различных страницах нашего сайта. Тогда нужно создать функцию для автоматизации этого процесса, которая может иметь следующий вид

```
<?
function Open()
{
// блок операторов работы с файлами
}
?>
```

Общая схема обращения к любой функции следующая - пишем некоторую функцию

```
<?
function PrintText($tex, $te)
{
echo "$tex, $te";
}
?>
```

в определенный файл, например

Print.php

затем подключаем его на тех страницах сайта, где это необходимо

```
Include "Print.php";
```

и для вызова функции просто пишем ее имя в основной программе, которая может находиться, например, в файле Text.php

```
<?
// здесь участок какого - то кода основной программы
...
// задание аргументов функции

$text='Результат работы функции';
$tex='которая задана в отдельном файле';

// подключаем файл функции

Include "Print.php";

// вызываем функцию

PrintText($text, $tex);
?>
```

В браузере будем иметь

Результат работы функции, которая задана в отдельном файле

Как мы видели, в скобках функции можно указывать аргументы, которые передаются функции из основной программы.

Кроме того, можно вернуть полученное в теле функции значение в вызывающую, основную программу. Рассмотрим следующий пример

Файл с функцией, например, sum.php

```
<?
```

```
function sum($a,$b)
{
$res = $a+$b;
return($res);
/* возвращаем полученное значение. Можно было написать и так return $res; */
}
?>
```

Файл основной программы, например, result.php

```
<?
...
// некоторый код основной программы
...
include "sum.php";
$a = sum(3,9);
/* можно было написать и так $x=3; $y=9; $a = sum($x,$y); */
echo $a;
?>
```

В результате своей работа программа выведет в браузер число

```
12
```

При описании функции мы использовали два аргумента $a и $b, которые передают функции любые два числа (или две переменные), функция складывает их, а затем передает нам результат вычислений. Кстати, при использовании конструкции return() есть одно важное ограничение - мы можем возвратить только один аргумент, хотя это может быть и массив, и таким образом, можно передать несколько значений.

Область переменных в функции

Теперь поговорим об области видимости переменных внутри функции и программы. Переменная созданная внутри функции становится "локальной" по отношению к данной функции. А это значит, что вы не можете обратиться к такой переменной из дру-

гой функции или любого места кода, находящегося вне функции, создавшей эту переменную. Кроме того, переменная, созданная вне функции, также не будет доступна внутри функции, если не передана ей в качестве параметра. Именно поэтому, в приведенном выше примере, можно было использовать переменную $a, как внутри функции, так и вне нее, так как эти переменные различны.

Для передачи параметров можно использовать следующие возможности

1. Если необходимо использовать внутри функции переменные, ранее определенные вне этой функции нужно

- Передать эти переменные в качестве аргументов функции.
- Зарегистрировать эти переменные внутри функции, как глобальные, т.е. доступные в самой функции.
- Обратиться к этим переменным через массив $GLOBALS[].

2. Если необходимо использовать в основной программе переменные, ранее определенные внутри некоторой функции нужно

- Вернуть эти переменные конструкцией return().
- Передать аргументы по ссылке (об этом мы поговорим немного позже).

Рассмотрим пример

```
<?
// Основная программа
$a = 3;
$b = 2;
// Определили переменные a и b

function Test()
{
global $a, $b;
/* декларируем переменные, как глобальные и теперь их можно использовать внутри функции */
$c = $a+$b;
```

```
return ($c);
}

echo Test();
// Выводим на экран результат действия функции
?>
```

который выведет в браузер

```
5
```

Однако, использование инструкции global считается плохим стилем написания программы и рекомендуется использовать обращение к массиву $GLOBALS[]. В данном случае это будет выглядеть так

```
$c = $GLOBALS["a"]+$GLOBALS["b"];
```

Иногда, возникает необходимость, чтобы функция помнила некоторую информацию о своих предыдущих вызовах. Есть два способа решения подобной задачи - использование инструкции global (или массива $GLOBALS[]) и определение статической переменной static. О первом варианте мы уже говорили и теперь рассмотрим второй способ. Для его демонстрации приведем пример

```
<?
function Counter()
{
static $count = 0;
$count++;
return $count;
}

for ($x = 0; $x < 5; $x++)
echo Counter();
?>
```

который выведет в браузер

```
12345
```

Здесь мы создаем внутри функции переменную $count и декларируем ее, как статическую. Это значит, что при первом вызове функции переменная получает присвоенное ей значение (в данном случае 0). Далее это значение увеличивается на 1 и возвращается в основную программу. Если бы отсутствовала инструкция static, то при каждом обращении к функции значение нашей переменной просто обнулялось, и в основную программу каждый раз возвращалась 1. А благодаря инструкции static, функция запоминает внутреннее значение переменной $count и таким образом операция присваивания ($count = 0) выполняется только однажды - при первом обращении к нашей функции.

Аргументы по умолчанию

Мы уже говорили о том, что можно передавать функции некоторые аргументы. В языке PHP, при работе с функциями, имеется возможность задавать значения аргументов "по умолчанию". Рассмотрим простой пример

```
<?
function Schet($x,$y=7)
{
$a = $x*$y;
return $a;
}

$rez = Schet(5);
echo $rez;
?>
```

Который выведет в браузер

```
35
```

Смысл такой записи заключается в том, что мы определяем функцию с двумя аргументами $x и $y, причем для второго аргумента в самом определении функции указываем значение "по умолчанию" - в данном случае это число 7.

Обработка функции происходит следующим образом

- Если при вызове функции задан только один аргумент

(здесь это 5), то второй подставляется "по умолчанию" (здесь это 7). В итоге получаем $a = 5*7.

- Если переданы оба аргумента (например, Schet(6,3)), то вычисляются именно переданные аргументы. Тогда получим $a = 6*3.
- Если не передан ни один из аргументов, функция вернет 0.

Если передано аргументов больше, чем определено в функции, лишние аргументы игнорируются - отсчет аргументов идет слева - направо, т.е. вызов функции в виде Schet(6,3,4,5) результат работы функции не изменит.

Значения аргументов "по умолчанию" могут находиться в любом месте в списке аргументов функции, т.е. записи вида

```
function Schet($x=4,$y);
function Schet($x,$y=6,$z);
```

так же допустима, но при вызове функции нужно задавать все определенный аргументы. Поэтому, будет невозможно использовать значение аргумента "по умолчанию". Для его использования нужно, чтобы такой аргумент стоял последним в списке аргументов функции.

Аргументы и ссылки

Допустим, у нас есть некоторая программа, которая в ходе своего выполнения вызывает функцию. При этом происходит следующее - значение переменной, передаваемой в качестве аргумента функции, копируется функцией в "локальную область" и именно с этой копией функция работает. Результат работы функции также находится в "локальной области". Именно поэтому, нам необходимо пользоваться инструкцией return, чтобы вернуть значение из "локальной области" в глобальную (область переменных основной программы) и присвоить его в качестве некоторого значения новой, глобальной переменной.

В то же время, имеется и другая возможность, чтобы функция имела доступ не к копии, а непосредственно к самой переменной. Такой механизм передачи параметров получил название "передача параметров по ссылке". При этом нет необходимости использовать return, так как в этом случае функция работает

именно с реальной переменной, находящейся в глобальной области, а не с локальной копией. Рассмотрим следующий пример

```
<?
// обратите внимание на знак & - амперсанд
function GetLink(&$link)
{
$link= $link*2;
$link++;
}

$test = 10;
GetLink($test);
echo $test;
?>
```

который выведет число

```
21
```

При определении функции, передаем параметр по ссылке - для этого перед именем аргумента $link ставим знак & - амперсант. Далее умножаем переменную на 2 и прибавляем 1. Итого 10*2+1=21. При такой передаче аргументов, инструкция return уже не требуется.

Можно указывать знак амперсанта непосредственно при передаче переменной в функцию

```
GetLink(&$test);
```

но все же рекомендуется ставить амперсант именно при определении функции, как это сделано в приведенном выше примере.

При использовании механизма "передача параметров по ссылке", в качестве параметра функции можно указывать только переменные, а не конкретные значения, например, цифру 10 передавать нельзя.

Встроенные функции

В этом параграфе мы рассмотрим несколько полезных

встроенных функций, которые часто встречаются при написании программ скриптов на PHP. Вначале приведем несколько функций общего назначения.

Первая из них, производит выполнение строки содержащей PHP код

```
eval(string);
```

где строка string должна содержать некоторый PHP код. Эта возможность может быть полезной для сохранения кода в текстовом поле базы данных для более позднего выполнения. При изменении значений переменных в eval(), эти переменные будут изменены и в основных данных

Рассмотрим пример, который выполняет простое объединение текста

```
<?
$string = 'cup';
$name = 'coffee';
$str = 'This is a $string with my $name.<br>';
echo $str;
eval( "\$str = \"$str\";" );
echo $str;
?>
```

При выполнении этого примера на экран будет выведено

```
This is a $string with my $name.
This is a cup with my coffee.
```

Следующая функция позволяет задавать задержку выполнения некоторого скрипта

```
sleep(seconds);
```

функция производит задержку выполнения программы в секундах (seconds). А другая функция

```
usleep(microseconds);
```

производит задержку выполнения программы в микросекун-

дах (microseconds).

Еще одна очень полезная функция, которую мы уже вкратце рассматривали

```
die(message);
```

выводит на экран сообщение message и завершает выполнение текущего скрипта, не возвращая никаких значений. Например

```
<?
$filename = '/path/to/data-file';
$file = fopen($filename, 'r') or die("Unable to open file $filename)";
?>
```

Теперь рассмотрим некоторые функции работы с символьными или строковыми переменными. Первая из них

```
convert_cyr_string(string, from, to);
```

переводит указанную строку string из одной русской кодовой таблицы from в другую to. Аргументы from и to являются одним символом, который определяет исходную и целевую кодовую таблицу. При перекодировке поддерживаются следующие типы кодовых таблиц

- k - koi8-r
- w - windows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

Например код скрипта

```
$a=convert_cyr_string($str, w, k);
```

Переедет строковую переменную $str из кодировки Windows в кодировку КОИ-8.

Функция шифрования

```
$a=crypt(string, [salt]);
```

зашифрует строку, используя стандартный метод шифрации UNIX DES. Аргументы являются строкой string, которую нужно зашифровать и дополнительной 2-символной строкой salt, на которой будет основываться шифрование. Если аргумент salt отсутствует, то он буде генерирован случайным образом. Зашифрованная строка буде находиться в переменной $a.

Некоторые операционные системы поддерживают больше одного типа шифрования. Иногда метод шифрования DES заменяется другими, основанными на методе MD5, алгоритмами. Во время установки, PHP определяет возможности функций шифрации, и будет поддерживать аргумент salt для других методов шифрации. Если параметр salt не установлен, то PHP автоматически сгенерирует стандартный 2-х символьный ключ DES, а если же в системе "по умолчанию" установлен тип шифрации MD5, то будет сгенерирован MD5 - совместимый ключ. Заметим, что функций дешифрации не существует, а crypt() использует однопроходный алгоритм.

Другая функция шифрования

```
$a=md5(string);
```

вычисляет зашифрованное значение MD5 для строки string, используя алгоритм RSA Data Security Inc. MD5 Message-Digest. Результат шифрования находится в переменной $a.

Кроме функции echo, которую мы уже рассматривали, имеются и другие способы вывода информации на экран. Рассмотрим следующую функцию

```
print($string);
```

которая выводит на экран строку $string. Следующая функция

```
printf(format, [$args]...);
```

осуществляет вывод строки на экран в соответствии с параметром format, который совпадает с описанием функции sprintf(). Эта функция имеет вид

```
sprintf(format, [$args]...);
```

и возвращает строку, обрабатываемую в соответствии с форматирующей строкой format.

Форматирующая строка может содержать ноль или более директив, которыми являются обычные символы (кроме %). Каждое такое описание состоит из следующих элементов

1. Дополнительный описатель заполнения, который указывает, какие символы будут использоваться для заполнения результата до правильного размера строки. "По умолчанию" строка заполняется пробелами, но это может быть и 0 (символ нуля). Альтернативный символ заполнения может быть определен в одинарных кавычках " ' ".
2. Дополнительный описатель выравнивания, который указывает на тип выравнивания результата - возможно выравнивание по левому или правому краю. "По умолчанию" выравнивание происходит по правому краю экрана. Символ тире " - " приведет к выравниванию по левому краю.
3. Дополнительный описатель ширины, который задается цифрой и указывает, с каким количеством символов (минимум) может производиться вывод числа.
4. Дополнительный описатель точности, (также задается числом) который указывает, сколько десятичных знаков после запятой следует отображать для чисел с плавающей точкой. Этот описатель не действует на остальные типы (имеется и другая полезная функция для форматирования чисел number_format()).
5. Описатель типа числа, который указывает на то, как должен трактоваться тип данных аргумента. Здесь возможны следующие типы

% - символ процента, с которого начинается написание формата числа.

b - аргумент трактуется, как integer и представляется, как двоичное число.

c - аргумент трактуется, как integer и представляется, как символ с ASCII значением.

o - аргумент трактуется, как integer и представляется, как восьмеричное число.

x - аргумент трактуется, как integer и представляется, как шестнадцатеричное число (с буквами в нижнем регистре).

X - аргумент трактуется, как integer и представляется, как шестнадцатеричное число (с буквами в верхнем регистре).

d - аргумент трактуется, как integer и представляется, как десятичное число.

f - аргумент трактуется, как double и представляется, как число с плавающей точкой.

s - аргумент трактуется и представляется, как строка.

Рассмотрим пример функции sprintf, в котором перед числами дополняются нули

```
<?
$year =2004;
$month = 5;
$day = 1;
$date = sprintf("%04d-%02d-%02d", $year, $month, $day);
echo "$date";
?>
```

На экране увидим

```
2004-05-01
```

Другой пример sprintf для форматирования денежной единицы

```
<?
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
echo "$money will output 123.1";
$formatted = sprintf ("%03.2f", $money);
echo "<br>$formatted will output 123.10";
?>
```

На экране получим

```
123.1 will output 123.1
123.10 will output 123.10
```

Если написать

```
$formatted = sprintf ("%03.3f", $money);
```

то после запятой будет выводиться два нуля 123.100, а если написать

```
$formatted = sprintf ("%05.2f", $money);
```

то перед самим числом будет стоять два нуля 00123.10. Перейдем теперь к следующей функции

```
$a=rawurlencode(string);
```

которая кодирует исходную строку в соответствии со стандартом RFC1738 и возвращает строку, в которой все не буквенно - цифровые символы, кроме "_" заменяются на знак (%) с последующими двумя шестнадцатеричными цифрами. Это кодирование применяется для защиты символов от интерпретации их, как особых разделителей URL, и для защиты URL от искажения системами передачи данных с переводом символов.

Такая функция используется, например, если вы хотите включить пароль в ftp URL

```
echo '<A HREF="ftp://user:', rawurlencode('foo @serg'), '@ftp.my.com/x.txt">'.'FTP</a>';
```

тогда реальная ссылка будет иметь вид

```
ftp://user:foo%20@serg@ftp.my.com/x.txt
```

Или, если передаете информацию в качестве части URL

```
echo '<A HREF="http://x.com/department_list_script/', rawurlencode('sales and marketing/Miami'), '">'.'URL</a>';
```

Реальная гиперссылка будет иметь вид

```
http://x.com/department_list_script/sales%20and%20marketing/Miami
```

Обратная ей функция

```
$a=rawurldecode(string);
```

декодирует URL - кодированную строку и возвращает строку, в которой последовательность из символа процента (%) и последующих 2-х шестнадцатеричных цифр заменяется соответствующим буквенным символом. Например, строка

```
foo%20bar%40baz
```

будет заменена на

```
foo bar@baz
```

А строка

```
http://x.com/department_list_script/sales%20and%20marketing/Miami
```

заменится на

```
http://x.com/department_list_script/sales and marketing/Miami
```

ФОРМЫ

Основная концепция, положенная в основу обработки форм на PHP, состоит в том, что поле любой формы с именем name="user", например

```
<input type = "text" name = "user" value = "">
```

автоматически преобразуется в переменную $user, а величина такой переменной будет равна величине, заданной в поле value="".

Все сказанное работает только в том случае, если в файле конфигурации PHP (php.ini) включена опция register_globals=On - в последних версиях PHP, начиная с 4.3, она обычно находится в положении Off.

Предположим пока, что register_globals=On и рассмотрим простую форму

```
<form method="post" action="form.php">
<input type="text" name="name" value="">
<input type="text" name="mail" value="">
<input type="submit" name="OK">
</form>
```

Тогда в файле скрипта будут непосредственно доступны следующие переменные

```
$name
$mail
```

со значениями, заданными в полях формы value. С ними можно выполнять любые действия, как с обычными HPH переменными.

В том случае, если register_globals=Off эти переменные в скрипте не существуют и нужно пользоваться элементами некоторых массивов. Например, при выборе метода POST для передачи данных из формы все переменные формы окажутся в массиве с именем $_POST. Таким образом, после заполнения и отправки формы, на странице обработки (т.е. в файле form.php) нам будут доступны следующие переменные

```
$name=$_POST["name"];
$mail=$_POST["mail"];
```

со значениями, введенными пользователем в соответствующие поля.

Существует два самых распространенных метода передачи данных из формы в скрипт - это GET и POST. Пользователь может различить их только по виду адресной строки. Например, если URL выглядит

```
http://www.ser.ru/reg.php?firstnam=Serg&secnam=Ivanov
```

то можно сказать, что в данной форме использован метод GET. Этот метод предполагает присоединение к URL имен и значений форм и делается это следующим образом

```
http://URL/file.php?переменная_1=значение&переменная_2=значение ...
```

Метод POST, в отличие от GET, незаметен для пользователя и ничего не прибавляет к URL.

Элементы формы

Перейдем теперь к рассмотрению различных элементов формы и способам передачи данных в PHP скрипт.

Простейший элемент - это однострочное текстовое поле

```
<input type="text" name="user" value="">
```

которое выглядит на экране следующим образом

В PHP скрипт передается переменная $user со значением, которое введено в это текстовое поле.

Следующий элемент

```
<textarea name="nam" value="">
```

допускает многострочный текстовой ввод и имеет полосы прокрутки. Передача переменных и их значений происходит, как в предыдущем случае.

Еще один элемент формы - это выбор из нескольких вариантов

```
<select name="nam">
```

В языке HTML ему сопоставляется следующий код

```
<form name="name">
<select name="variants">
<option value="1">
вариант1
</option>
<option value="2">
вариант 2
</option>
<option value="etc">
и т.д.
</option>
</select>
</form>
```

В скрипт передается переменная $variants с одним из значений (1, 2 или etc) в зависимости от выбора пользователя. Вместо обычной переменной можно использовать и массивы. Если в <OPTION> величины value не указаны, передается текст между тегами <OPTION> и </OPTION>.

Рассмотрим теперь два управляющих элемента формы

```
<input type="checkbox" name="check" value="">
```

В случае установки флажка на таком элементе, в PHP обработчик передается переменная с именем, соответствующим имени самого checkbox (т.е. $check) и значением On. Если checkbox не установлен, то эта переменная определена не будет.

Следующий элемент

```
<input type="radio" name="radiobox" value="">
```

RadioBox1

самый распространенный способ использования этого элемента - выбор только одного из альтернативных вариантов. Для него существует следующий HTML код

```
<form ...>
<input type="radio" name="var1" value="1">
вариант 1
<input type="radio" name="var2" value="2">
вариант 2
<input type="radio" name="var3" value="etc">
и т.д.
</form>
```

В PHP скрипт передается переменная $var с номером и значением 1, 2 или etc. Вместо простой переменной можно использовать и массивы.

Невидимый элемент формы (это также однострочное текстовое поле, но оно не видимо на экране браузера), но, тем не менее, несущий значения в PHP скрипт

```
<input type="hidden" value="">
```

В PHP программировании его часто используют для передачи данных через несколько страниц. Для этого на каждой странице необходимо расположить скрипт, читающий значение элемента Hidden с предыдущей страницы и присваивающий это значение элементу Hidden текущей страницы.

Рассмотрим еще два элемента форм, которые также являются управляющими

```
<input type="submit" name="sub" value="submit">
```


```
<input type="image" src="url картинки" name="sub">
```

Оба эти элемента выполняют одинаковую функцию - передачу данных из формы в PHP скрипт для их обработки. Первый элемент Submit - это простая кнопки. Второй Image имеет важную особенность - этот элемент может передавать скрипту два значения - координаты X и Y пикселя изображения, на котором был совершен щелчок мышкой.

В случае, если register_globals=On и мы имеем более старую версию PHP (обычно до 4.2), то для вывода всех значений форм можно использовать предопределенные переменные

```
$HTTP_GET_VARS;
$HTTP_POST_VARS;
```

которые зависят от метода передачи данных. О таких переменных мы поговорим в следующих главах, а пока лишь скажем, что это зарезервированные переменные, которые берут свои значения из окружения сервера.

Рассмотрим теперь пример формы для случая register_globals=On, которая содержит управляющие элементы. Создадим файл с формой form.htm

```
<html>
<head>
<body>
<form method="post" action="script.php">
Имя
<input type="text" name="user"><br>
Возраст
<select name="age[]">
<option value=15-20>
15 – 20
<option value=21-30>
21 – 30
<option value=31-40>
```

```
31 – 40
<option value=более_40>
Более 40
</select><br>
<input type="checkbox" name="confirm">
Подписаться на рассылку<br>
<input type="submit">
</form>
</body>
</html>
```

И файл обработчик формы script.php

```
<?
// если человек подписался
if ($confirm == 'on')
{
// просто выведем переменные в браузер
echo "$user<BR>$age[0]";
}
?>
```

Если пользователь выберет на форме checkbox "Подписаться на рассылку", в браузере мы увидим значения, введенные в поля Имя и Возраст.

Для случая register_globals=Off скрипт обработчик будет выглядеть следующим образом

```
<?
// если человек подписался
if ($_POST["confirm"] == 'on')
{
// просто выведем переменные в браузер
echo "$user<BR>$age[0]";
}
?>
```

В приведенной форме, для работы с полем типа SELECT, был использован ассоциированный массив, т.е. было задано имя поля с пустыми квадратными скобками []. Для применения такого метода есть две причины

1. При такой записи пользователь может делать множественный выбор. В этом случае нужно использовать служебное слово MULTIPLE внутри конструкции SELECT, и нам придется проверить весь массив на наличие не нулевых элементов.
2. Переменные одной смысловой группы удобнее хранить в массиве под одним именем, а не использовать множество разных переменных.

Остановимся теперь на элементе формы radio - PHP скрипт вполне может обрабатывать и этот элемент. Создадим файл с формой form.htm

```
<html>
<head>
<body>
<form method="post" action="script.php">
Имя
<input type="text" name="user"><br>
Возраст<br>
<input type="radio" name=age[] value='15 - 20'>
15 - 20<br>
<input type="radio" name=age[] value='21 - 30'>
21 - 30<br>
<input type="radio" name=age[] value='31 - 50'>
31 - 50<br>
<input type="checkbox" name="confirm">
Подписаться на рассылку<br>
<input type="submit">
</form>
</body>
</html>
```

И файл (для случая register_globals=On) обработчик script.php

```
<?
// если человек подписался
if ($confirm == 'on')
{
// просто выведем переменные в браузер
echo $_post["user"].'<br>'.$age[0];
```

```
}
?>
```

Для случая register_globals=Off скрипт обработчик будет выглядеть следующим образом

```
<?
// если человек подписался
if ($_POST["confirm"] == 'on')
{
// просто выведем переменные в браузер
echo $_POST["user"].'<br>'.$age[0];
}
?>
```

В том случае, если пользователь выберет на форме checkbox "Подписаться на рассылку", в браузере мы увидим значения, введенные в поля Имя и Возраст.

Как можно видеть, каждая кнопка radio имеет имя массива age и соответствующее значение. Таким образом, при обработке формы мы получим в элементе массива age с индексом 0, т.е. age[0], выбранное пользователем значение. В массиве всегда будет содержаться только один элемент, так как отсутствует возможность множественного выбора.

Возможен, конечно, и другой вариант, когда для каждого элемента radio мы определяем разные переменные, а затем проверяем, какая из них существует.

Одностраничная форма

Безусловно, уникальной особенность PHP является его способность автоматически передавать значения полей форм в переменные PHP. Например, вы обрабатываете форму, которая имеет поле ввода следующего вида

```
<input type="text" name="name" value="Hello">
```

и когда вы щелкаете по кнопке Submit на форме, переменная $name получает значение Hello.

Тогда вы можете вывести это значение на экран

```
echo " $name!";
```

или создать некоторую проверку с помощью условного оператора

```
if ($name == "Hello")
{echo "Please input your E-mail";}
```

Рассмотрим небольшой пример, в котором попросим посетителя ответить на несколько вопросов (указать имя, электронный адрес и т.д.) и сформируем выводимую страницу с информацией в зависимости от его ответов.

Разделим нашу страницу на две части, но не будем создавать два отдельных файла (раньше использовалась одна страница для формы, другая для PHP - скрипта, который эту форму обрабатывает), а создадим одну PHP - страницу, вид которой изменяется в зависимости от результатов анализа данных, введенных посетителем в имеющуюся на этой странице форму.

Первая функция служит для вывода формы

```
<?
function display_form()
{
global $php_self;
?>

<form target="<?php echo $php_self; ?>" method=get>
Имя:
<input type=text name="name"><br>

Любимый сорт сыра:
<input type=radio name="cheese" value="brie">
Very soft french brie
<input type=radio name="cheese" value="cheddar">
Farmhouse english cheddar
<input type=radio name="cheese" value="mozzarella">
Italian buffalo mozzarella<br>

Когда вы предпочитаете есть сыр:
<input type=checkbox name="times[]" value="m">
На завтрак
```

```
<input type=checkbox name="times[]" value="n">
В обед
<input type=checkbox name="times[]" value="d">
На ужин
<input type=checkbox name="times[]" value="l">
Поздно ночью

<input type=hidden name="stage" value="results"><br>
<input type="submit" value="OK">
</form>
<?
}
?>
```

Большая часть представленного выше текста - это обычный HTML - код, необходимый для создания формы. Однако, есть несколько необычных моментов, которые требуют особого пояснения

1. Во - первых, это переменная $PHP_SELF, которая представляет удобную форму ссылки на PHP скрипт, а ее значение автоматически равно URL текущей страницы. Мы задаем здесь атрибуту TARGET этой формы значение $PHP_SELF для того, чтобы именно эта страница обрабатывала форму. Используя $PHP_SELF вместо реального пути к файлу скрипта, мы получаем возможность, перемещать его, не заботясь о внесении соответствующих изменений.

2. Строка

```
global $php_self;
```

означает, что мы хотим использовать внутри функции глобальную переменную $PHP_SELF. Внутри функции переменные имеют локальную сферу действия, то есть имеют значения, отличающиеся от значений переменных с теми же самыми именами, но определенных вне этой функции. Если мы не укажем PHP, что хотим использовать глобальное значение $PHP_SELF, то обнаружим, что $PHP_SELF будет пустым.

3. Далее, мы свободно переходим в режим PHP и обратно даже внутри функции. PHP умеет игнорировать HTML - код и выделять только PHP - блоки. Это быстрее и проще, чем оста-

ваться в режиме PHP и организовывать динамический вывод HTML тегов, используя команду PHP echo.

Теперь обратите внимание на атрибут NAME тега INPUT. Видно, что в случае, когда тип поля ввода checkbox, после наименования поля (в данном случае $times[]) стоит пара квадратных скобок, если же тип поля ввода есть radio, то после имени (cheese) таких скобок нет. Это объясняется тем, что поля типа radio служат для выбора одного и только одного из вариантов ответа, поэтому значением переменной $cheese будет только одна строка.

Поля типа checkbox (их несколько) позволяют выбрать (или не выбрать) каждый из возможных вариантов, поэтому для сохранения ответов пользователя, PHP должен запомнить их в виде массива. Добавление [] после имени переменной times и указывает на то, что в данном случае это массив, а не единичная переменная.

И, наконец, в этой части кода имеется скрытая переменная с именем $stage, которая будет использоваться для указания программе - хотим ли мы вывести на экран форму и отобразить результаты обработки ответов.

Теперь рассмотрим другую функцию process_form(), которая используется для обработки формы после ввода ответов пользователем

```
<?
function process_form()
{
global $name;
global $cheese;
global $times;

if ($cheese == 'brie')
{
$cheese_message = 'I love brie';
}

elseif ($cheese == 'cheddar')
{
$cheese_message = 'Cheddar is awesome!';
}
```

```
else
{
$cheese_message = 'Fresh mozzarella is divine';
}

$favorite_times = count($times);

if ($favorite_times <= 1)
{
$times_message = 'You should eat cheese more often';
}

elseif ($favorite_times < 4 && $favorite_times > 1)
{
$times_message = 'Those are good times to eat cheese';
}

else
{
$times_message = 'You are eating too much cheese';
}

echo "Hello $name<br>";
echo "$cheese_message <br> $times_message";
}
?>
```

Здесь так же, как с $PHP_SELF, мы считываем глобальные значения переменных, включенных в форму. Затем мы смотрим, какой сорт сыра выбрал пользователь, и в соответствии с этим формируем начало выводимого на страницу текста.

Далее используется функция count() для того, чтобы подсчитать, сколько раз в день ест сыр человек, ответивший на наши вопросы. Если в предыдущем случае мы должны были сравнить ответ посетителя с каждым из возможных значений переменной $cheese, то при анализе значений переменной $favorite_times можно воспользоваться операциями сравнения "меньше чем" и "больше чем". Знаки && в условном операторе

```
"$favorite_times > 1 && $favorite_times < 4"
```

означают логическую операцию "И" ("and"). Для того, чтобы условие было выполнено $favorite_times должно одновременно быть больше 1 и меньше 4. И в конце, выводится введенное пользователем имя и наше для него сообщение.

Теперь, когда мы имеем эти две функции, остается добавить только маленький кусочек кода для того, чтобы вызвать их в нужной последовательности. Выше записи тела функций display_form() и process_form() добавляем

```
<?
if (empty($stage))
{
display_form();
}
else
{
process_form();
}
?>
```

Здесь мы вначале проверяем, задано ли значение переменной $stage. В PHP переменная считается пустой (empty), если ее значение не задано явно (т.е. в PHP переменным никогда не придаются значения "по - умолчанию") или ей задано пустое значение (переменной присвоено значение пусто, если она приравнена пустой строке - '' или ей присвоено значение 0). Когда посетитель впервые попадает на нашу страницу, переменная $stage - пуста. Поэтому на страницу будет выводиться форма с нашими вопросами, в противном случае мы должны обработать заполненную форму с помощью второй функции.

В языке PHP имеется встроенная функция Empty(), которая проверяет, присвоено ли переменной некоторое значение. Кроме того, можно использовать функцию IsSet(), которая проверяет, существует ли переменная.

Имеется и другой вариант записи такой программы - приведем его полностью, поскольку он мало отличается от описанного выше, и не содержит функции

```
<?
if (empty($stage))
{
```

```
?>

<form action="<? echo $php_self; ?>" method=post>
Имя:
<input type=text name="name"><br>

Любимый сорт сыра:
<input type=radio name="cheese" value="brie">
very soft french brie
<input type=radio name="cheese" value="cheddar">
farmhouse english cheddar
<input type=radio name="cheese" value="mozzarella">
italian buffalo mozzarella

<br>Когда вы предпочитаете есть сыр:
<input type=checkbox name="times[]" value="m">
На завтрак
<input type=checkbox name="times[]" value="n">
В обед
<input type=checkbox name="times[]" value="d">
На ужин
<input type=checkbox name="times[]" value="l">
Поздно ночью

<input type=hidden name="stage" value="results">
<br><input type="submit" value="OK">
</form>
<?
}

else

{
if ($cheese == 'brie')
{
$cheese_message = 'I love brie';
}

elseif ($cheese == 'cheddar')
{
$cheese_message = 'Cheddar is awesome!';
```

```
}

else
{
$cheese_message = 'Fresh mozzarella is divine';
}

$favorite_times = count($times);
if ($favorite_times <= 1)
{
$times_message = 'You should eat cheese more often';
}

elseif ($favorite_times < 4 && $favorite_times > 1)
{
$times_message = 'Those are good times to eat cheese';
}

else
{
$times_message = 'You are eating too much cheese';
}

echo "Hello $name<br>";
echo "$cheese_message <br> $times_message";
}
?>
```

Такой вариант программы проще с точки зрения логики ее построения и заметно быстрее работает.

Многостраничные формы

Такие конструкции могут применяться при создании больших форм анкет или сбора других данных, которые вводит пользователь при регистрации на сервере. К сожалению, протокол HTTP не имеет встроенных механизмов переноса данных с одной страницы на другую, поскольку в нем не определено понятие серии страниц. Однако, с помощью нескольких приемов и ряда удобных встроенных функций языка PHP мы можем обойти это ограничение без больших затруднений.

Основная идея очень проста - мы сохраняем в скрытых переменных все значения, введенные посетителем в поля формы на предыдущей странице, и используем их затем на следующей странице. Таким образом, нужная информация сохраняется при переходе со страницы на страницу.

Для удобства вернемся к примеру формы, который мы рассматривали выше, и разобьем нашу страницу на две страницы. В результате мы будем иметь три функции вместо двух. В дополнение немного изменим их названия, чтобы они соответствовали изменившемуся содержанию. Функция display_form() теперь будет называться display_name()

```
<?
function display_name()
{
global $PHP_SELF;
?>

<form target="<? echo $php_self;?>" method=get>
name:
<input type=text name="name"><br>
<input type=hidden name="stage" value="cheese">
<input type=submit value="thanks!">
</form>
<?
}
?>
```

Она идентична ранее введенной функции, за тем исключением, что мы удалили все вопросы, кроме первого, и изменили название следующего этапа на "cheese", чтобы дать более точное указание на то, что будет делаться на следующем этапе. Далее идет функция display_cheese().

```
<?
function display_cheese()
{
global $php_self;
global $name;
?>
```

```
<form target="<? echo $php_self;?>" method=get>
Ваш любимый сорт сыра:
<input type=radio name="cheese" value="brie">
Very soft french brie
<input type=radio name="cheese" value="cheddar">
Farmhouse english cheddar
<input type=radio name="cheese" value="mozzarella">
Italian buffalo mozzarella

Когда вы предпочитаете есть сыр:
<input type=checkbox name="times[]" value="m">
На завтрак
<input type=checkbox name="times[]" value="n">
На обед
<input type=checkbox name="times[]" value="d">
На ужин
<input type=checkbox name="times[]" value="l">
Поздно ночью
<input type=hidden name="name" value="<? echo
htmlspecialchars($name); ?>">
<input type=hidden name="stage" value="results">
<input type=submit value="OK">
</form>
<?
}
?>
```

И тут нет никаких неожиданных изменений по сравнению с предыдущим вариантом. Просто, перед тем, как изменить значение скрытой переменной, используемой для определения этапа перемещения по нашим страницам, мы еще запоминаем имя и значение переменной $name, т.е. часть информации, которая была получена на предыдущем этапе.

При этом мы не просто напрямую передаем это значение, а пропускаем его через функцию PHP, которая называется htmlspecialchars(). Стандарт HTML определяет четыре символа, которые не могут использоваться в других целях, кроме как в тегах для разметки документов <, >, ", и & . По этой причине, для того чтобы убедиться, что мы не поставим браузер пользователя в затруднительное положение, мы пропускаем $name через функцию htmlspecialchars(), чтобы пользователь, имеющий имя "Bret

& Jeff", так и остался "Brett & Jeff."

Теперь, когда информация, введенная в поля формы, передается на другую страницу, мы не потеряем ни одной ее части. В результате этого, у нас нет необходимости что - либо менять в функции process_form(), но сохраняем ее в другом файле с именем script.php.

В операторе, определяющем логику отображения страниц, добавляем только одну строку и подключаем файл с последней функцией оператором include

```
<?
include "script.php";
if (empty($stage))
{
display_name();
}
elseif ($stage == 'cheese')
{
display_cheese();
}
else
{
process_form();
}
?>
```

Здесь мы добавили elseif между двумя предыдущими операторами. В этом особенность использования переменной $stage. Если мы хотим добавить дополнительные страницы, нам нужно только написать новые функции для отображения того, что мы хотим, и еще по одной строке в операторе управления, в соответствии с тем, когда эти страницы должны быть отображены.

Пример формы: Калькулятор

В данный момент мы уже рассмотрели достаточно много различных возможностей PHP, и может попытаться применить их на практике, написав программу простого калькулятора. Вначале решим, что должен делать такой калькулятор. В первую очередь, он, конечно, должен выполнять арифметические действия - это сложение, вычитание, деление и умножение. Затем добавим еще

несколько функций, которые будут вычислять корень из числа, возводить его в указанную степень и, наконец, выводить процент от числа.

Для начала создадим файл, который назовем calc.php, а использование пользовательских и встроенных функций позволит нам совместить в этом файле, как форму для ввода чисел и других данных, так и вывод результата

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title>Калькулятор</title>
</head>
<body>

<?
// теперь пишем php код
function show()
{global $action;
?>

<form method=Get action="calc.php" target="_blank">
<!--target="_blank" - позволяет выводить результат обработки формы в новом окне -->
Первое число
<input type="text" name="first"><br>
Второе число (Степень, процент)
<input type="text" name="second"><br>

<select size="1" name="action">
<option value="sum">Сложить</option>
<option value="min">Вычесть</option>
<option value="mult">Умножить</option>
<option value="dev">Разделить</option>
<option value="stepen">Возвести в степень</option>
<option value="procent">Процент от числа</option>
<option value="koren">Корень</option>
</select>
<br>
<input type="submit" value="Выполнить">
```

```
</form>

<?
}
//конец функции show()
```

Здесь мы определили функцию show(), которая будет выводить в окне браузера форму для задания чисел и выбора действий.

Обратите внимание на разбивку скрипта - мы снова вставили в него блок HTML - кода, который также относится к функции show. Такая форма подобна написанию HTML - кода с использованием возможностей PHP

```
echo "<form ... и так далее до </html>";
```

Метод передачи данных из формы может быть одним из двух - либо POST, либо GET. Переменные, передаваемые из формы в сценарий PHP, помещаются в глобальный ассоциативный массив $HTTP_POST_VARS или $HTTP_GET_VARS в зависимости от метода, указанного в форме. Они автоматически становятся доступными для вашей программы, если опция register_globals установлена в файле php.ini, как register_globals =On.

Продолжаем писать код файла calc.php дальше, и определим новую функцию

```
function calc()
{global $action, $result, $first, $second;
switch($action)
{
case "sum": $result = $first+$second; break;
case "min": $result = $first-$second; break;
case "mult": $result = $first*$second; break;
case "dev":

if (!$second) {exit("Деление на ноль");}
// если второе число равно "0" или вообще не введено

$result=$first/$second; break;
case "procent": $result = $first*($second/100); break;
```

```
case "stepen": $result = pow($first, $second); break;
case "koren": $result = pow($first,0.5); break;
}
?>

Результат Вашего действия равен <b>

<?
echo $result;
// вывод результата
?>
</b>
</font>
<?
}
// конец функций calc()
```

Здесь мы написали функцию calc, занимающуюся непосредственно вычислением результата и вывода его на экран браузера.

Заканчиваем написание кода программы

```
if ($action) calc(); else show();
?>
```

Если в переменной $action есть какое - то значение (а оно может появиться только в случае нажатия кнопки "Выполнить"), то выполняется функция вычисления calc(), если же $action пуста, то снова выводится форма калькулятора.

Конечно, можно было дополнить наш калькулятор еще тригонометрическими функциями, например, sin(), cos(), tan(), котангенс 1/tan() и т.д. Но это не меняет сути дела, а лишь делает пример более громоздким.

Приведем еще один пример калькулятора, который будет располагаться в двух файлах - HTML с именем calcul.htm и PHP с именем calcul.php.

Файл calcul.htm, который запускается первым

```
<html>
<head><title>Форма Калькулятора</title></head>
<body>
<form method="post" action="calcul.php">
```

```
<P>Значение 1:
<input type="test" name="val1" size=10>
<P>Значение 2:
<input type="test" name="val2" size=10>
<P>Действие:<BR>
<input type="radio" name="calc" value="сложить">
Сложить<BR>
<input type="radio" name="calc" value="вычесть">
Вычесть<BR>
<input type="radio" name="calc" value="умножить">
Умножить<BR>
<input type="radio" name="calc" value="разделить">
Разделить</P>
<P>
<input type="submit" name="submit" value="Вычислить">
</P>
</form>
</body>
</html>
```

Файл calcul.php, который вызывается после нажатия кнопки "Вычислить"

```
<?
if ($calc == "сложить")
{
$result = $val1 + $val2;
}
else if ($calc == "вычесть")
{
$result = $val1 - +$val2;
}
else if ($calc == "умножить")
{
$result = $val1 * $val2;
}
else if ($calc == "разделить")
{
$result = $val1 / $val2;
}
?>
```

```
<html>
<head>
<title>Результат вычисления</title>
</head>
<body>
Результат вычисления:
<? echo "$result"; ?>
</body>
</html>
```

В результате работы такой программы в том же окне будет выведен результат вычислений.

ГРАФИКА

Перейдем теперь к рассмотрению средств языка PHP, которые используются для работы с графикой. Используя эти средства можно формировать по - настоящему динамические изображения, рисовать различные графики, графические счетчики, кнопки, элементы интерфейса, меняющиеся в зависимости от некоторых условий и многое другое.

Для работы с изображениями в PHP необходима дополнительная графическая библиотека GD. И если вы собираетесь дальше работать с динамическими изображениями нужно заранее узнать, есть она на сервере вашего хостинг - провайдера и какие функции она поддерживает.

Структура рисунка

В самом начале, перед выводом генерируемого изображения нам необходимо послать браузеру заголовок, определяющий формат текущего рисунка

```
Header ("Content - type: image/png");
```

Следующий шаг, это вызов одной из основных функций библиотеки GD при работе с изображениями

```
imagecreate($x,$y);
```

которая создает, так называемое "окно изображения", т.е. некоторую ограниченную область, размером $x на $y пикселей, на которой будет формироваться изображение и позволяет присвоить ей идентификатор. Работа с изображениями строится по тому - же принципу, что и работа с файлами, мы работаем не с самим файлом или изображением, а с их идентификатором.

Для работы с цветом существует функция

```
imagecolorallocate(1,2,3,4);
```

аргументами которой будут являться четыре значения

1 - идентификатор изображения.

2,3,4 - три целочисленных значения в диапазоне от 0 до 255,

которые определяют интенсивность цветов в формате RGB.

Для вывода изображения в браузер используется функция

```
imagePng($id);
```

а также

```
imageGif($id);
imageJpeg($id);
```

где $id - это идентификатор готового изображения.

Освобождение памяти, занятой картинкой, производит функция

```
imageDestroy($id);
```

Рассмотрим теперь, как может выглядеть сам рисунок

```
<?
/* создаем окно изображения 100x100 пикселов */
$image = imagecreate(100,100);
// создаем цвет заливки, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
/* посылаем браузеру заголовок о том, что будет выводиться рисунок */
Header ("Content - type: image/png");
// выводим наш квадрат, залитый красной палитрой
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

Этот скрипт выведет в браузер просто красный квадрат

Обратите внимание, мы присвоили переменной $red цвет заливки, но для вывода изображения, по - прежнему, обращаемся к установленному идентификатору $image. Можно не присваивать значение заливки переменной $red, а просто записать

```
imagecolorallocate($image,255,0,0);
```

и результат не изменится.

Рисование линий

Для рисования простых линий служит функция

```
imageline(1,2,3,4,5,6);
```

где 1 - идентификатор изображения.
2 и 3 - координаты начала линии, отчет координат начинается с нуля.
4 и 5 - координаты окончания линии.
6 - цвет линии.

Функция нарисует линию заданного цвета, начиная и заканчивая заданными координатами.
Заливка ограниченной области заданным цветом выполняется с помощью функции

```
imagefill(1,2,3,4);
```

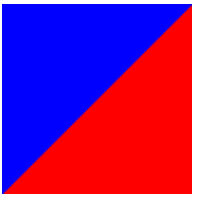
где 1 - идентификатор изображения.
2 и 3 - координаты точки, с которой нужно начинать заливку.
4 - цвет заливки.

Посмотрим, как выглядит все описанное выше на примере

```
<?
/* создаем окно изображения размером 100x100 пикселей */
$image = imagecreate(100,100);
// создаем цвет заливки, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет заливки, в данном случае синий
```

```
$blue = imagecolorallocate($image,0,0,255);
// чертим синюю линию из угла в угол
imageline ($image,0,99,99,0,$blue);
// заливаем одну из частей синим цветом
imagefill ($image,0,90,$blue);
/* посылаем браузеру заголовок о том, что будет выводиться рисунок */
Header ("Content - type: image/png");
/* выводим наш сине - красный квадрат, разбитый по – диагонали */
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

Такой скрипт выведет в браузер рисунок



Рисование дуг

Для рисования дуги служит функция

```
imagearc (1,2,3,4,5,6,7,8);
```

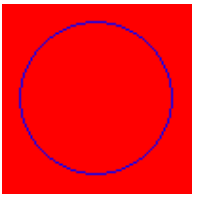
где 1 - идентификатор изображения.
2 и 3 - координаты центра дуги.
4 и 5 - целочисленные значения ширины и высоты.
6 и 7 - начальный и конечный угол дуги (в градусах).
8 - идентификатор цвета.

Рассмотрим пример

```
<?
/* создаем окно изображения размером 100x100 пикселей */
```

```
$image = imagecreate(100,100);
// создаем цвет заливки, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет дуги, в данном случае синий
$blue = imagecolorallocate($image,0,0,255);
// рисуем круг
imagearc($image,49,49,80,80,0,360,$blue);
/* посылаем браузеру заголовок о том, что будет выводиться рисунок */
Header ("Content - type: image/png");
// выводим красный квадрат, с синим кругом по центру
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

который выведет в браузер



Рисование прямоугольников

Для рисования прямоугольников используется функция

```
imagerectangle(1,2,3,4,5,6);
```

где 1 - идентификатор изображения.
2 и 3 - координаты верхнего левого угла прямоугольника.
4 и 5 - координаты нижнего правого угла прямоугольника.
6 - идентификатор цвета.

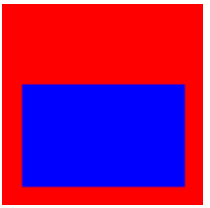
Кроме того, существует функция, которая рисует залитый цветом прямоугольник

```
imagefilledrectangle(1,2,3,4,5,6);
```

аргументы определяются, как в предыдущем случае. Рассмотрим пример

```
<?
/* создаем окно изображения размером 100x100 пикселей */
$image = imagecreate(100,100);
// создаем цвет, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет, в данном случае синий
$blue = imagecolorallocate($image,0,0,255);
// рисуем круг
imagefilledrectangle($image,10,40,90,90,$blue);
/* посылаем браузеру заголовок о том, что будет выводиться рисунок */
Header ("Content - type: image/png");
// выводим красный квадрат, с синим прямоугольником
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

который выведет в браузер



Рисование многоугольников

Для рисования многоугольников служит следующая функция

```
imagepolygon(1,2,3,4);
```

где 1 - идентификатор изображения.
2 - массив, содержащий координаты точек.
3 - целочисленное значение, задающее количество вершин.

4 - идентификатор цвета.

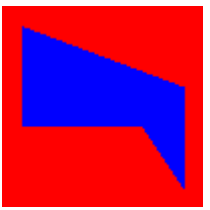
Для создания многоугольника, окрашенного в определенный цвет, существует функция

```
imagefilledpolygon(1,2,3,4);
```

с идентичными параметрами. Приведем простой пример

```
<?
/* создаем окно изображения размером 100x100 пиксе-
лей */
$image = imagecreate(100,100);
// создаем цвет, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет, в данном случае синий
$blue = imagecolorallocate($image,0,0,255);
// массив с координатами вершин многоугольника
$points = array (10,10,90,40,90,90,70,60,10,60);
// создаем многоугольник
imagefilledpolygon($image, $points, count($points)/2,
$blue);
/* посылаем браузеру заголовок о том, что будет выво-
диться рисунок */
Header ("Content - type: image/png");
// выводим красный квадрат, с синим многоугольником
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

который создаст в браузере следующую картинку



Задание прозрачности цвета

Для создания прозрачных цветов служит функция
imagecolortransparent(1,2);

где 1 - идентификатор изображения.
2 - идентификатор цвета, который и будет прозрачным в заданном изображении.

Рассмотрим пример

```
<?
/* создаем окно изображения размером 100x100 пикселей */
$image = imagecreate(100,100);
// создаем цвет, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет, в данном случае синий
$blue = imagecolorallocate($image,0,0,255);
// массив с координатами вершин многоугольника
$points = array (10,10,90,40,90,90,70,60,10,60);
// создаем многоугольник
imagefilledpolygon($image, $points, count($points)/2, $blue);
// определяем прозрачный цвет
imagecolortransparent ($image, $blue);
/* посылаем браузеру заголовок о том, что будет выводиться рисунок */
Header ("Content - type: image/png");
/* выводим красный квадрат, с прозрачным  многоугольником */
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

В результате получим

Вывод текста

Для вывода строки текста в область рисунка служит функция

```
imageString(1,2,3,4,5,6);
```

где 1 - идентификатор изображения.
2 - идентификатор размера шрифта (от 1 до 5).
3 и 4 - координаты верхнего левого угла прямоугольника, в который будет вписана строка.
5 - сама строка.
6 - идентификатор цвета шрифта.

Приведем простой пример

```
<?
/* создаем окно изображения 100x100 пикселей */
$image = imagecreate(100,100);
// создаем цвет, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет, в данном случае синий
$blue = imagecolorallocate($image,0,0,255);
// формируем текст
imageString($image,5,30,40,"PHP4",$blue);
/* посылаем браузеру заголовок о том, что будет выводиться рисунок */
Header ("Content - type: image/png");
// выводим красный квадрат, с синим текстом
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

В окне браузера будем иметь

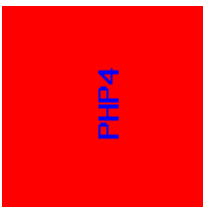
Другая функция

```
imageStringUp(1,2,3,4,5,6);
```

с аналогичными аргументами, предназначена для вывода строки, только в вертикальном направлении

```
<?
/* создаем окно изображения размером 100x100 пиксе-
лей */
$image = imagecreate(100,100);
// создаем цвет, в данном случае красный
$red = imagecolorallocate($image,255,0,0);
// создаем цвет, в данном случае синий
$blue = imagecolorallocate($image,0,0,255);
// формируем текст
imageStringUp($image,5,45,65,"PHP4",$blue);
/* посылаем браузеру заголовок о том, что будет выво-
диться рисунок */
Header ("Content - type: image/png");
// выводим красный квадрат, с синим текстом
imagePng($image);
// очищаем память
imageDestroy($image);
?>
```

На экране будет



Кроме представленных выше возможностей, PHP может работать со шрифтами TrueType, однако, для этого они должны быть откомпилированы с библиотекой FreeType. Данная библиотека позволяет существенно расширить возможности работы со шрифтами.

ФАЙЛЫ И ДИРЕКТОРИИ

Необходимость в работе с файлами и папками (директориями) встает перед любым программистом очень часто. Если скрипты не используют базы данных, то файлы становятся единственными хранителями любой информации. Применение файлов, как хранилищ информации, позволяет использовать их в самых разнообразных ситуациях, а сама возможность работы с файлами существенно расширяет диапазон применения любого языка программирования.

Поэтому, одним из важнейших инструментов любого языка программирования является именно возможность работы с файлами (проверка существования, чтение данных из файла или запись в файл) и язык PHP предоставляет нам достаточно большое число возможностей для решения подобных задач.

Включение файлов в документ

Для того, чтобы включить в документ содержимое другого файла используются директивы include() и require(), которые мы уже вкратце рассматривали. При таком включении фрагмент кода включаемого файла (будь то простой текст, HTML, PHP или JavaScript код) обрабатывается так же, как фрагмент из самого включающего файла. Разница между этими двумя директивами только в том, что include(), так же, как это делают функции (подробнее об этом смотрите далее), может возвращать значения, а require() - нет.

Если вы знакомы с расширением .shtml, в котором используется технология SSI (Server Side Includes - вложение со стороны сервера), то, примерно, для тех же целей служат рассмотренные здесь команды. Однако, они предоставляют нам намного больше возможностей.

Допустим, имеется документ, состоящий из десятка страниц, причем, в большей части этих страниц имеется повторяющийся код (неважно какой, HTML или PHP). В этом случае имеет смысл вынести повторяющийся смысловой или программный модуль в отдельный файл и подключать его динамически во все остальные файлы, в данном случае, в наш начальный документ.

Если завтра нам понадобится изменить программный код данного фрагмента, то нужно будет поменять его только в одном включаемом файле, а не на всех десятках страниц основного

файла. Такое использование директивы include() сэкономит много времени на отладку всего приложения - основного скрипта. Приведем пример

```
<?
// включаемый файл с именем date.txt
$bg='#fffddd';
$font='#003366';
$date=date("Сегодня: d.m.Y.");
?>

<?
// основной, включающий файл index.php
include ('date.txt');
/* здесь можно написать include ("date.txt") или include 'date.txt', или include "date.txt" */
?>

<HTML>
<HEAD>
<TITLE>
INCLUDE DIRECTIVE
</TITLE>
</HEAD>
<BODY BGCOLOR=<?=$bg?>>
<FONT COLOR=<?=$font?>>
Добрый день.
<BR>
<?=$date?>
<BR>
Некоторый текст...
</BODY>
</HTML>
```

В окне браузера будет выведено

Добрый день.
Сегодня: 02.05.2004г.
Некоторый текст...

Фон документа будет светло - желтый, что определяется

оператором bgcolor, а буквы - темно синими, благодаря использованию fontcolor.

Теги определения цвета фона, текста и вывода даты можно записать и таким образом

```
<BODY BGCOLOR=<? echo $bg ?>>
<FONT COLOR=<? echo $font ?>>
.....
<? echo $date ?>
```

Еще одним примером может служить вариант, когда в отдельные файлы организуют, так называемую, шапку документа (начало HTML - кода, логотип, меню и т.д.) и завершение документа (баннеры, счетчики, контактная информация и т.п.).

Проверка существования файла

Для проверки существования файла в указанной папке служит функция

```
file_exists();
```

которая принимает строку, содержащую путь (полный или относительный) к файлу. Если файл найден, возвращается true, иначе false. Рассмотрим пример

```
<?
$fp="date.txt";
if(!file_exists($fp))
{
echo "<CENTER><font size=4 color=#003300>Файл с таким именем не существует!<HR>";
exit;
// выход из скрипта без его дальнейшего выполнения
}
include "$fp";
/* здесь можно написать include $fp , но нельзя писать include '$fp' поскольку имя файла подставляться не будет */
echo $date;
?>
```

В случае существования такого файла он будет выводить на экран текущую дату.

Используя эту конструкцию, можно переписать приведенный выше пример, следующим образом

```
<?
$fp="date.txt";
if (!file_exists($fp))
{
echo "<CENTER><font size=4 color=#003300>Файл с таким именем не существует!<HR>";
exit;
}
include "$fp";
?>

<HTML>
<HEAD>
<TITLE>
INCLUDE DIRECTIVE
</TITLE>
</HEAD>
<BODY BGCOLOR=<? echo $bg ?>>
<FONT COLOR=<? echo $font ?>>
Добрый день.
<BR>
<? echo $date ?>
<BR>
Некоторый текст...
</BODY>
</HTML>
```

Приведем теперь некоторые правила описания пути к файлу.

"filename"	Просто указание имени файла означает, что он находится в текущей директории.
"./dir/filename"	Файл содержится в папке dir, находящейся внутри текущей директории. Длина цепочки из папок не ограничена.

"../filename"	Файл лежит в предыдущей директории. Каждый знак ../ расценивается, как возврат в родительскую директорию.
"../dir/filename"	Файл находится в папке dir, которая лежит в предыдущей директории.

Используя эти правила текст скрипта можно записать в виде

```
<?
$fp="./dir/date.txt";
if (!file_exists($fp))
{
echo "<CENTER><font size=4 color=#003300>Файл с таким именем
не существует!<HR>";
exit;
}
include "$fp";
echo $date;
exit;
?>
```

Такая запись подразумевает, что файл date.txt находится в директории dir, которая лежит внутри текущей папки.

Открытие и закрытие файла

Для открытия файла для дальнейшей работы с ним существует функция fopen(), которая имеет два параметра

```
fopen(filename, mode);
```

Если filename начинается с http:// (без учета регистра), открывается соединение с указанным сервером и возвращается указатель файла. Поскольку редиректы (перенаправления) в HTTP не обрабатываются, вы должны включать в указание директории завершающие слеши.

Если filename начинается с ftp:// (без учета регистра), открывается ftp соединение с указанным сервером и возвращается указатель файла. Если сервер не поддерживает режим пассивного

ftp, данная операция завершится ошибкой. Вы можете открывать через ftp файлы, как для чтения, так и для записи (но не обе операции одновременно).

Если filename начинается как - нибудь иначе, открывается файл вашей файловой системы, и возвращается указатель открытого файла. Если открыть файл, по какой - либо причине, не удается функция fopen() возвращает false.

Параметр mode определяется следующим образом

- 'r' - Открыть только для чтения, и поместить указатель на начало файла.
- 'r+' - Открыть для чтения и для записи, и поместить указатель на начало файла.
- 'w' - Открыть только для записи, поместить указатель на начало файла и очистить все содержимое файла. Если файл не существует, создается новый файл
- 'w+' - Открыть для чтения и для записи, поместить указатель на начало файла и очистить все содержимое файла. Если файл не существует, создается новый файл.
- 'a' - Открыть только для записи, и поместить указатель на конец файла - будет происходить дозапись в конец файла. Если файл не существует, создается новый файл.
- 'a+' - Открыть для чтения и для записи, и поместить указатель на конец файла. Если файл не существует, создается новый файл.

Параметр mode может также содержать символ 'b', который используется в системах, различающих бинарные и текстовые файлы. Если данное значение в вашей системе не имеет смысла, оно игнорируется.

Для того, чтобы использовать функцию fopen(), мы должны присвоить результат ее работы переменной, т.е. создать указатель на файл, который открывается этой функцией

```
$fp = fopen ("test.php","r");
//открыть для чтения
$fp = fopen ("test.php","w");
// открыть для записи
$fp = fopen ("test.php","a");
// открыть для добавления
```

или

```
$fp = fopen("/home/rasmus/file.txt", "r");
$fp = fopen("http://www.php.net/", "r");
$fp = fopen("ftp://user:password@example.com/", "w");
```

По окончании работы с файлом его следует закрыть. Данную процедуру выполняет функция fclose(), которой в качестве аргумента нужно передать указатель на ранее открытый файл (в нашем случае это $fp). Таким образом, для закрытия файла test.php, описанного в предыдущем примере, нам необходим следующий код

```
fclose($fp);
```

Эта функция закрывает указатель на файл $fp и возвращает true при удачной операции и false при ошибке. Хотя интерпретатор PHP при завершении выполнения скрипта сам закрывает все открытые файлы, все же лучше сделать это в самой программе.

Чтение и запись в файл

Для чтения строк из файла, открытого функцией fopen(), может применяться функция fgets(), которая используется следующим образом (и является некоторым аналогом другой функции чтения fread())

```
$fp = fopen ("test.php","r");
$stroka = fgets("$fp",1024);
```

Эта функция за каждый раз своего выполнения возвращает только одну строку файла, при этом она перемещает внутренний указатель файла на следующую строку, которую она прочитает при следующем к ней обращении. Поэтому, если вам необходимо прочитать файл целиком, необходимо использовать эту функцию в цикле.

Заметьте, что функция fgets() использует дополнительный параметр length, равный в нашем примере 1024, который указывает максимальную длину строки или максимальное число символов, которое можно прочесть из файла, до того момента пока не встретится конец строки или файла. Если размер строки пре-

вышает это число, то функция возвратит ее в "урезанном" виде, объемом в length байт.

По умолчанию этот параметр установлен в 1024 байт или в один килобайт. Заметим, что если вы используете файлы больших размеров, то при чтении таких файлов может переполниться буфер выполнения PHP (его объем указывается в файле конфигурации), а это приведет к зависанию всей системы.

Приведем теперь простой пример использования этой функции для чтения большого файла

```
$file = fopen("filename", "r");
while ($line = fgets($file, 4096))
{
echo $line;
}
```

Обратите внимание, что здесь задается не имя файла, а указатель на файл, возвращаемый функцией fopen().

Для определения достижения конца файла существует функция feof(), которая возвращает true при достижении конца файла и false в противном случае. Функции нужно передать указатель на файл

```
feof($fp);
```

Напишем теперь скрипт, который будет читать файл и выводить его содержимое строка за строкой.

```
<?
$file = 'date.txt';
$fp = fopen($file,"r") or die("Невозможно открыть $file");
/* следующий цикл будет работать, пока не достигнут конец файла */
while(! feof($fp))
{
$stroka = fgets($fp,1024);
// читаем очередную строку и
echo $stroka.'<BR>';
// выводим ее в браузер
}
?>
```

Этот скрипт выведет в браузер содержимое заданного файла

```
Первая строка
Вторая строка
...
И так до последней строки файла
```

Если использовать файл date.txt из первого примера этой главы, на экране мы получим

```
// включаемый файл с именем date.txt
$bg='#fffddd';
$font='#003366';
$date=date("Сегодня: d.m.Y.");
```

Обратите внимание, что здесь мы использовали функцию die(). Напомним, что если при выполнении некоторого действия возвращается результат false, она выводит свой аргумент на экран браузера и заканчивает выполнение программы скрипта.

Рассмотрим теперь другую функцию чтения информации из файла fread(), которая выполняет бинарное чтение всей информации, которая находится в данном файле

```
fread($fp, length);
```

Она читает информацию заданной длинны length из файла, на который ссылается указатель $fp. Чтение заканчивается, когда прочитано length байт или достигнут eof - конец файла (конец строки игнорируется)

```
<?
// получить содержимое файла в одну строку
$fp = "date.txt";
$f = fopen($fp, "r") or die("Don't open file");
$c = fread($f, filesize($fp));
fclose($f);
echo $c;
?>
```

Для того же файла date.txt на экране получим

```
// включаемый файл с именем date.txt $bg='#fffddd'; $font='#003366'; $date=date("Сегодня: d.m.Y.");
```

Здесь, в отличие от предыдущего случая, не нужно использовать цикл для чтения строк и явно указывать условие конца файла, задаваемое функций feof($fp).

В этом примере мы использовали новую функцию

```
filesize($fp);
```

которая возвращает размер (в байтах) файла с именем $fp или false в случае ошибки его определения.

Приведем еще один пример проверки существования файла в некоторой директории (без использования функции die), чтения, находящейся в нем информации и вывода ее на экран

```
<?
$fp="date.txt";
if (file_exists($fp))
{
$f=fopen($fp,"r");
$ps=fread($f,filesize($fp));
fclose($f);
include "$fp";
exit;
}
else
{
echo "<CENTER><font size=4 color=#003366>Файл не существует!<hr>";
exit;
}
?>
```

В случае существования этого файла на экране увидим

```
// включаемый файл с именем date.txt $bg='#fffddd'; $font='#003366'; $date=date("Сегодня: d.m.Y.");
```

Такой вывод на экран возможен, если в файле находится любая текстовая информация без тегов <? ?>. Если такой файл от-

сутствует в текущей директории, то на экран будет выведено заданное в скрипте сообщение.

Рассмотрим теперь функцию записи в открытый файл текстовой строки

```
fputs($fp, $str, [length]);
```

fputs() - это аналог другой функции fwrite(), и обе функции полностью идентичны. Здесь $fp - указатель на открытый для записи файл, а $str - текстовая строка, предназначенная для записи. Параметр length не обязателен, и при его отсутствии записывается вся строка $str.

Рассмотрим и функцию

```
fwrite($fp, $str, [length]);
```

которая записывает содержимое строкой переменной $str в файловый поток, указанный в $fp. Если аргумент length присутствует, запись останавливается после записи length - го байта, или после записи всей строки string. Если аргумент length отсутствует, запись останавливается после записи всей строки string.

Для демонстрации описанных выше функций приведем пример создания простого счетчика посещений Web - страницы

```
<?
$fp="counter.txt";
$file = fopen($fp, "r") or die("Don't open file");
$c = fread($file, filesize($fp));
fclose($file);

$c++;
$file = fopen($fp, "w") or die("Don't open file");
fputs($file, $c);
fclose($file);
echo "<CENTER><font size=4 color=#003366>Вы  $c -й посетитель этой страницы";
?>
```

Для того, чтобы такой скрипт работал, нужно предварительно создать файл counter.txt и записать в него цифру 0.

Рассмотрим еще две полезные функции, одна из которых

```
file(filename);
```

Эта функция непосредственно работает с файлом и возвращает его содержимое в виде массива, где каждый элемент массива является строкой файла.

Рассмотрим простой пример, в котором N определяет число строк для считывания

```
<?
$fp="date.txt";
$a=file($fp);
$N=5;
for ($i=0; $i<$N; $i++)
{
echo $a[$i].'<br>';
}
?>
```

Для используемого ранее файла date.txt, на экране будем иметь

```
// включаемый файл с именем date.txt
$bg='#fffddd';
$font='#003366';
$date=date("Сегодня: d.m.Y.");
```

Другая интересная функция

```
readfile(filename);
```

выводит на экран (никаких echo использовать не нужно) содержимое указанного файла, а также возвращает число символов в файле (или количество байт). Причем, в отличие от file, эта функция не предусматривает присвоение прочитанного содержимого файла определенной переменной - это содержимое выводится автоматически при использовании только самой команды readfile($fp). Если попытаться присвоить ей переменную, то этой переменной будет присваиваться только число прочитанных символов

```
<?
```

```
$fp="date.txt";
$b = readfile($fp);
echo "<br>Файл ".$fp." содержит ".$b." символов";
?>
```

При запуске этого скрипта на экране получим

```
// включаемый файл с именем date.txt $bg='#fffddd'; $font='#003366'; $date=date("Сегодня: d.m.Y.");
Файл date.txt содержит 100 символов
```

Последние две функции самодостаточны и не требуют предварительного открытия файла для чтения.

Рассмотрим еще одну полезную функцию

```
rename(oldname, newname)
```

которая пытается переименовать файл с именем oldname в новый файл с именем newname. Функция возвращает true при успешном выполнении операции и false при сбое.

И в заключение приведем функцию копирования файлов

```
copy(source, dest);
```

которая создает копию файла source с новым именем dest. Возвращает true при успешном завершении копирования и false в противном случае. Рассмотрим пример

```
if (!copy(date.'.txt', date.'.bak'))
{
print("Невозможно скопировать файл $file...<br>");
}
```

В результате получаем копию файла date.txt, но с расширением bak.

Работа с директориями

Тесно связаны с действиями над файлами операции с директориями или папками. Алгоритм работы с ними схож с операциями над файлами - вначале директорию необходимо открыть,

выполнить какие - либо действия и закрыть ее.

Рассмотрим первую функцию работы с директориями

```
$f=opendir($path);
```

которая открывает указанную в переменной $path директорию и возвращает служебный идентификатор соединения с этой директорией. Пути к директории $path следует указывать следующим образом.

.	Точка означает открытие директории
./dir/	Открытие папки dir, находящейся в текущей директории
..	Открытие папки на уровень выше текущей

Следующая функция

```
readdir($f);
```

читает директорию, на которую указывает идентификатор $f и открытую opendir. За каждый проход она возвращает имя файла или папки, которые находятся в указанной директории, и перемещает внутренний указатель на следующий объект директории. Так что для прочтения всей директории ее необходимо использовать в цикле.

Заметим, что эта функция возвращает служебные объекты папки "." и "..", которые можно отсекать при выводе оператором if.

Функция

```
closedir($f)
```

закрываем директорию, указывая в качестве аргумента идентификатор соединения с этой папкой.

Рассмотрим пример, в котором выполняется вывод всех файлов в текущем каталоге

```
<?
$handle=opendir('.');
echo "Directory handle: $handle<BR>";
```

```
echo "Files: ";
while ($file = readdir($handle))
{echo "$file<BR>";}
closedir($handle);
?>
```

В результате его работы на экране мы увидим

```
Directory handle: Resource id #1
Files:
.
..
Dosreg.php
date.txt
index1.htm
login-var.php
login.php
accounts
counter.txt
date.bak
```

Можно записать этот пример и в таком виде

```
<?
$dir = opendir(".");
while($file = readdir($dir))
echo "$file<BR>";
closedir($dir);
?>
```

Иногда использование функций работы с директориями может очень сильно облегчить написание различных скриптов. Например, в разделе "Функции. Руководства пользователя по PHP" можно видеть список всех PHP функций в алфавитном порядке.

Не трудно представить, сколько времени нужно потратить, чтобы вручную написать весь этот список с гиперссылками и в алфавитном порядке. Вот здесь очень помогают функции работы с директориями - описание каждой функция помещается в отдельный файл (в текущей директории) с именем, соответствующим названию функции и расширением htm (например, aaa.htm, bbb.htm, ccc.htm, ddd.htm, eee.htm). Затем создаем следующий

скрипт и также помещаем его в текущую директорию, например, в файле cont.php

```
<?
$i = 0;
$handle = opendir('.');
while($file = readdir($handle))
{
if ($file != '.' && $file != '..')
{
$func[$i] = $file;
/* формируем массив названий файлов с описаниями
функций */
$i++;
}
}

sort($func);
for ($q = 0; $q<sizeof($func); $q++)
{
echo "<a href = ".$func[$q].">".$func[$q]."</a><br>";
}
?>
```

Этот скрипт позволяет, при каждом заходе на страницу, получить только что сгенерированный список всех функций, т.е. файлов с именами функций, которые будут содержать описания самих функций

aaa.htm
bbb.htm
ccc.htm
cont.php
ddd.htm
eee.htm

На экран выводится список файлов, который представляют собой гиперссылки на каждый из этих файлов. Щелчок по такой ссылке сразу открывает содержимое этого файла.

Следующая функция работы с директориями позволяет создавать новые директории

```
mkdir(pathname, mode) or die("Don't create directory");
```

При ее использовании будет создана директория, указанная в pathname. Заметим, что если вы захотите указать mode в восьмеричной системе, то число должно начинаться с 0.

Запись вида

```
mkdir("/user/dir", 0700) or die("Don't remove directory");
```

создает новую директорию dir в папке user и возвращает true при успешном выполнении операции и false при ошибке.

Следующая функция

```
rmdir(dirname) or die("Don't remove directory");
```

пытается удалить директорию, указанную в dirname. Директория должна быть пустой, и настройки сервера должны допускать такую операцию. При возникновении ошибки возвращается 0, что соответствует значению false.

Рассмотрим еще одну, часто используемую функцию

```
is_dir($pp)
```

которая проверяет существование директории с заданным именем $pp. В качестве имени может выступать полный путь к нужной директории. Функция возвращает true, если $pp существует и это именно директория.

Рассмотрим пример, в котором проверяется наличие директории account и, если она не существует, то будет создана в текущем каталоге

```
<?
$pp="account";
if (!is_dir($pp))
{
mkdir($pp,0);
echo "Создана директория $pp";
}
else
{echo "Директория $pp существует";}
?>
```

Работа с WEB документами

Интересная особенность PHP заключается в том, что он может выступать не только в роли обработчика и исполнителя сценария, но и в качестве клиента сети. Если до этого мы манипулировали только локальными объектами, такими как файлы и директории, то сейчас мы познакомимся со способами взаимодействия с удаленными объектами.

Обращение к документам, расположенным на удаленном сервере, производится уже знакомой нам функцией fopen(). Однако, в этом случае, она может быть вызвана только в режиме чтения.

Далее с полученным дескриптором файла можно делать то же самое, что и с дескриптором локального файла. Например, можно вывести все содержимое этого файла на экран пользователя

```
if (!($fp = fopen("http://url.ru/doc.htm", "r"))
exit("Не удается установить соединение");
fpassthru($fp);
```

Функцией fpassthru() мы просто вывели на экран все содержимое Web - документа (doc.htm), то есть на своей странице мы получим точную копию той страницы, которую хотели бы получить.

Чаще всего работа с удаленными файлами проводится для получения необходимой информации из целого документа. Примером может служить установленный на сайте прогноз погоды, получаемый, например, с Метеобюро, или курс доллара с сайта

```
http://www.rbc.ru/
```

Все это и многое другое можно получить с помощью сетевых возможностей PHP.

Извлечение части информации производится с помощью регулярных выражений. Использование регулярных выражений основано на расположении нужных вам данных в HTML коде документа.

СЕССИИ, ЗАГОЛОВКИ И COOKIES

Теперь рассмотрим такой важный механизм языка PHP, как сессии. Рано или поздно практически перед каждым Web - разработчиком встает проблема передачи данных через несколько страниц.

Один из способов такой передачи - использование скрытых элементов формы "hidden", который мы уже рассматривали. На каждой странице сайта мы размещаем эти элементы, внося в них с помощью PHP значения и передавая эти значения далее, другой странице. Конечно, такой способ вполне возможен, но он не очень рационален. Представьте себе, сколько таких элементов надо вставить на сайте объемом, например, в 100 страниц.

Сессии

В PHP 4.0 реализован очень удобный и функциональный механизм работы с сессиями, который позволяет сохранять любые данные, связанные с пользователем, и использовать их на протяжении всего времени нахождения пользователя на этом сайте.

Реализация сессий

Любая сессия открывается с помощью функции

```
session_start();
```

создающей специальный служебный файл с именем, соответствующим ID сессии, в который, впоследствии, будут записаны все данные, связанные с текущей сессией.

Место размещения этих файлов зависит от настроек PHP, которые выполняются в файле php.ini. В режиме работы "по умолчанию", файлы сессий хранятся в папке TMP, которая должна находиться в корневом каталоге диска, на котором установлен PHP интерпретатор. Если вы используете в своих скриптах сессии, нужно очищать директорию с этими временными файлами. В ней, со временем, может накопиться большое количество ненужных уже временных файлов.

Эта же функция используется для продолжения текущей сессии. Таким образом, она должна быть вызвана на каждой странице, использующей данные текущей сессии. В PHP предусмотрено

два способа передачи ID сессии (сокращенно SID)

- Метод GET - посетитель будет видеть в своем браузере адресную строку вида

```
http://server.com/main.php?PHPSESSID=bdd95bcd4e1e2ef5ec57fc83a69bba86
```

- С помощью Cookie - посетитель страницы не будет видеть признаков существования сессии, как в предыдущем случае, а SID будет передаться через Cookie (эти "куки" мы рассмотрим далее).

Следующий шаг в работе с сессиями - запись данных в сессию, которая выполняется с помощью функции

```
session_register();
```

Она сохраняет в файл текущей сессии значения указанных переменных, которыми вы в любой момент можете воспользоваться.

Регистрация данных в сессию должна выглядеть примерно следующим образом

```
session_start();
session_register('name', 'birth');
$name = "Bill Smit";
$birth = "21 марта";
```

Организовав сессию и вызвав на любой странице данного сайта функцию session_start(), вы можете обратиться к посетителю по его имени и будете знать его дату рождения. Имя и дата могут, конечно, вводиться в полях любой формы. Запись информации в файле сессии имеет вид

```
birth|s:7:"21 марта"; name|s:11:"Bill Smit";
```

Обратной функции session_register() является функция

```
session_unregister();
```

которая удаляет данные из текущей сессии. В скобках должно быть указано (в апострофах и через запятую) имя переменной, заданное функцией session_register(), которое нужно удалить из данной сессии. Эта функция используется довольно редко, но иногда бывает очень полезной. Например, в том случае, если вы регистрируете в сессию большое количество переменных, и чтобы не перезагружать файл текущей сессии, можно удалить оттуда уже ненужные данные. Например

```
session_unregister('name');
```

теперь в файле сессии будет записано

```
birth|s:7:"21 марта";
```

Заметим, что для версии PHP интерпретатора, начиная с 4.2, сессия может открываться функцией

```
msession_create (session)
```

хотя можно, по - прежнему, использоваться и приведенный выше код.

Работа с сессиями

Нередко при работе с сессиями нам требуется определить ее ID. Этим занимается функция с соответствующим названием

```
$ID=session_id();
```

которая в качестве результата возвращает ID текущей сессии. На этой функции основан механизм подсчета посетителей на сайте в данный момент.

Бывают случаи, когда становится очень неудобным использовать ID сессии, например, из - за их громоздкости и не наглядности, так как id сессии вида

```
7542b069d57510a99eaeb31391b15cbf
```

нам практически ничего не скажет. В этом случае более разумным становится использование функции

```
session_name();
```

которая выполняет две роли. Во - первых, она может возвращать имя текущей сессии (по умолчанию - PHPSESSID). В этом случае ее следует использовать без аргументов. Во - вторых, эта функция может устанавливать имя текущей сессии. Рассмотрим пример

```
session_start();
echo session_name();
session_name("MySession");
echo '<br>'.session_name();
```

Данный пример выведет на экран браузера

```
PHPSESSID
MySession
```

Безусловно, такие названия сессии воспринимаются намного лучше, чем сложные ID. Завершает работу сессии функция

```
session_destroy();
```

Она уничтожает файл, связанный с текущей сессией, что иногда является очень удобным. Но здесь возникает проблема - часто мы не знаем, где именно необходимо уничтожить сессию. Например, если данные сессии используются страницами всего сайта, то мы не можем уничтожить сессию на определенной странице, так как не знаем, какая именно страница будет последней, просмотренной посетителем страницей сайта.

Поэтому использование данной функции возможно лишь в том случае, если мы заранее знаем, на какой именно странице действие сессии должно прекратиться.

Пример сессии

Возможно, вы видели на сайтах надпись примерно следующего содержания: "Сейчас на сайте ... человек". Вот и мы сейчас напишем код скрипта, позволяющий считать число посетителей сайта. Есть два способа реализовать механизм подсчета посетителей, которые работают в режиме "Онлайн"

- С помощью IP адреса посетителя - разные посетители имеют разные IP.
- С помощью сессий PHP - для каждого посетителя заводится уникальная сессия.

Для начала создадим файл, который будет заменять нам базу данных, и назовем его base.txt. Далее, в каком - либо другом файле, например online.php, напишем сам скрипт

```
<?
session_start();
session_set_cookie_params('0');
$id = session_id();
```

Мы открываем новую сессию (или продолжаем, если она уже открыта), затем "прячем" сессию в cookie, чтобы пользователю не мешали длинные URL, а ID сессии присваиваем соответствующей переменной. Далее

```
$CurrentTime = time();
$base = "base.txt";
```

присваиваем переменной $CurrentTime текущее время в секундах, начиная с 1970 года (время UNIX).

Теперь мы определяем файл - базу base.txt и присваиваем ему определенную переменную - массив

```
$file = file($base);
```

В нем будут храниться необходимые данные в следующей форме (каждая строка файла будет находиться в соответствующем элементе массива $file)

```
ID сессии1|Время последней активности
ID сессии2|Время последней активности
```

Затем мы формируем новый массив $line из строк данного файла, т.е. элементов массива $file. Поскольку в каждой строке массива $file только две записи, разделенные знаком |, то массив $line для каждой строки массива $file будет содержать только два элемента - $line[0] и $line[1]. Во втором из них содержится время

последней записи в файл $file, т.е. время последней активности пользователя с некоторым id

```
$k = 0;
for ($i = 0; $i < sizeof($file); $i++)
{
$line = explode("|", $file[$i]);

if (($CurrentTime - $line[1]) < 600)
{
$ResFile[$k] = $file[$i];
$k++;
}
}
```

В этом блоке работает цикл, который, каждую строку массива $file, поочередно "разбивает" символом | (это результат работы функции explode) и формирует двухэлементный массив $line. Затем сравниваем время последней активности посетителя сайта с данным id (элемента массива $line[1]) с допустимой, которая на 10 минут меньше текущего времени. Здесь использована функция sizeof(), которая позволяет определить размерность массива.

Если последняя запись в файл была сделана не более 10 минут назад, то данная строка файла - базы записывается в новый формируемый массив $ResFile. В этом файле будет столько строк, сколько пользователей не превысило лимит времени в 10 минут.

Если же время последней активности более 10 минут от текущего времени, такая строка игнорируется, и элемент массива $ResFile для этого пользователя не создается. Далее мы будем иметь дело только с новым массивом $ResFile, который содержит записи активных пользователей. Пишем программу далее и обрабатываем теперь массив $ResFile, создавая новый массив $line, структура которого будет аналогична предыдущему

```
for ($i = 0; $i<sizeof($ResFile); $i++)
{
$line = explode("|", $ResFile[$i]);
if ($line[0]==$id)
{
$line[1] = trim($CurrentTime);
```

```
$sid = 1;
}
$ResFile[$i] = implode("|", $line);
}
```

В этом блоке первые две строки полностью аналогичны предыдущей записи. Далее мы сравниваем ID сессии, записанный в массиве $ResFile, с текущим ID. Если они равны, обновляем время последней активности, делая его равным текущему времени. Функция trim(string) вырезает лишние пробелы в начале и конце строки string.

Затем мы определяем переменную $sid, которая будет сигнализировать о том, что данный ID сессии уже есть в файле base.txt, и присваиваем ей единицу.

Далее мы используем функцию implode(), которая превращает полученный массив $line в строку с разделителем | и присваиваем его тому же массиву $ResFile. Таким образом, мы полностью проверили весь массив, разбирая каждую строку, и независимо от того, была ли изменена эта строка или нет (т.е. была ли она обновлена), возвращаем ее в массиве $ResFile.

Теперь перезаписываем файл - базу, сохраняя там новый, модифицированный массив $ResFile. Напомним, что этот массив содержит только данные активных пользователей.

```
$fp = fopen($base, "w");
for ($i = 0; $i<sizeof($ResFile); $i++)
{
fputs($fp, $ResFile[$i]);
}
fclose($fp);
```

Затем рассмотрим случай, если текущего ID сессии нет в файле. Тогда мы его просто добавляем

```
if (!$sid)
{
$fp = fopen($base, "a-");
$line = $id."|".$CurrentTime;
fputs($fp, $line);
fclose($fp);
}
```

Теперь все записи и перезаписи закончены, и нам остается только вывести количество открытых сессии, а значит и количество посетителей на сайте в данный момент. Так как число таких сессий равно количеству строк в файле, то пишем

```
$file = file($base);
echo '<br>'.sizeof($file);
?>
```

Подключите данный скрипт на каждой странице вашего сайта функцией include(), и вы всегда сможете определить количество человек на сайте в данную минуту.

Приведем теперь текст всего скрипта целиком в несколько измененном виде

```
<?
session_start();
session_set_cookie_params('0');
$id = session_id();

$CurrentTime = time();
$LastTime = time() - 10;
$base = "base.txt";

$file = file($base);
$k = 0;
$siz=sizeof($file);

for ($i = 0; $i < $siz; $i++)
{
$line = explode("|", $file[$i]);

if (($CurrentTime - $line[1]) < 600)
{
$ResFile[$k] = $file[$i];
$k++;
}
}
$si=sizeof($ResFile);
for ($i = 0; $i< $si;   $i++)
{
```

```
$line = explode("|", $ResFile[$i]);

if ($line[0]==$id)
{
$line[1] = trim($CurrentTime);
$sid = 1;
}
$ResFile[$i] = implode("|", $line);
}

$fp = fopen($base, "w");
for ($i = 0; $i<sizeof($ResFile); $i++)
{
fputs($fp, $ResFile[$i]);
}
fclose($fp);

if (!$sid)
{
$fp = fopen($base, "a-");
$line = $id."|".$CurrentTime;
fputs($fp, $line);
fclose($fp);
}

$file = file($base);
echo "<br>Число пользователей на сайте =
".sizeof($file);
?>
```

Вывод числа пользователей можно организовать только на одной определенной странице. Для этого нужно убрать из приведенного файла две последние строчки ($file и echo) и создать новый файл со следующим текстом

```
<?
$base = "base.txt";
$file = file($base);
echo "<br>Число пользователей на сайте =
".sizeof($file);
?>
```

который всегда можно подключить на страничку администратора функцией include() или переходить на него по гиперссылке.

Заголовки

Вначале скажем несколько слов о HTTP заголовках (headers). В соответствии со спецификацией HTTP, этот протокол поддерживает передачу служебной информации от сервера к браузеру, которая оформлена в виде специальных заголовков.

Таким образом, HTTP headers - это средство общения сервера с удаленным клиентом, т.е. вашим браузером. Обычно каждый заголовок состоит из одной строки ASCII текста с именем и значением, заключенной в HTML теги < >. Сами заголовки никак не отображаются в окне браузера, но, как правило, могут сильно изменить отображение сопутствующего документа.

Механизм отправки HTML заголовков в языке PHP представлен функцией header(). Особенность протокола HTTP заключается в том, что заголовок должен быть отправлен до посылки других данных, поэтому функция header() должна быть вызвана в самом начале документа и выглядеть следующим образом

```
header("HTTP заголовок", [replace]);
```

Опциональный, не обязательный параметр replace может принимать значения типа bool (true или false) и указывает на то, должен ли быть замещен предыдущий заголовок подобного типа, либо добавить данный заголовок к уже существующему, отправленному ранее.

Вместо функции header() часто используется функция headers_sent(), которая в качестве результата возвращает true в случае успешной отправки заголовка и false в обратном случае.

Рассмотрим теперь наиболее используемые HTTP заголовки.

"Cache-control: value"

Заголовок управления кэшированием Web - страниц. Вообще говоря, данная функция является одной из самых распространенных при использовании заголовков.

Такой заголовок может быть использован со следующими значениями (value)

- no-cashe - Запрет кеширования. Используется в часто обновляемых страницах и страницах с динамическим содержанием. Его действие подобно META тегу "Pragma: no-cache".
- public - Разрешение кеширования страницы, как локальным клиентом, так и прокси сервером.
- private - Разрешение кеширования только локальным клиентом - браузером.
- max-age - Разрешение использования кэшированного документа в течение заданного времени в секундах, например

```
header("Cache-control: private, max-age = 3600")
/* Кеширование локальными клиентами и использование в течение 1 часа */
```

"Expires: HTTP-date"

Устанавливает дату и время, после которого документ считается устаревшим. Дата должна указываться в следующем формате (на английском языке)

День недели (сокр.) число (2 цифры) Месяц (сокр.) год часы:минуты:секунды GMT

Например

Fri, 09 Jan 2002 12:00:00 GMT

Текущее время в этом формате возвращает функция gmdate()

```
echo gmdate("D, d M Y H:i:s")."GMT";
```

Возможно использование данного HTTP заголовка для запрета кеширования. Для этого необходимо указать прошедшую дату.

"Last-Modified: HTTP-date"

Указывает дату последнего изменения документа. Дата должна задаваться в том же формате, что и в случае с заголовком Expires. Данный заголовок можно не использовать для динамических страниц, так как многие серверы (например, Apache) для

таких страниц сами выставляют дату модификации.

Можно сделать страницу всегда обновленной

```
header("Last-Modified: ".gmdate("D, d M Y H:i:s")." GMT");
```

"Location: абсолютный URL"

Полезный заголовок, который перенаправляет браузер на указанный URL адрес. Его действие сравнимо с META тегом Refresh, который имеет вид

```
<meta http-equiv="refresh" content="0; url=someurl">
```

Этот заголовок в PHP может быть использован, например, так

```
if ($login!=$admin_login)
header("Location: http://www.server.com/login.php");
else
header("Location: http://www.server.com/admin.php?
login=$login");

if (!headers_sent())
exit("Произошла ошибка! Пройдите <a href='http://www.
server.com/login.php'>авторизацию</a> заново");
```

Мы, конечно, привели здесь не все HTTP заголовки, но рассмотрели наиболее полезные и самые используемые.

PHP и Cookie

В предыдущем параграфе мы разобрали взаимосвязь протокола HTTP и языка PHP на уровне HTTP заголовков. Сейчас мы познакомимся еще с одним специфическим HTTP заголовком - cookie. В дословном переводе с английского это "печенье" или кусочек, а по - русски говорят "куки". В данном случае имеется в виду, кусочек информации, записанной сервером на компьютер пользователя. В дальнейшем эту информацию можно извлечь, причем сделать это может только тот сервер, который ее туда записал. Эта информация ограничена размером в 4 Кб.

В процессе развития WWW - технологий и внедрения языков

Web - программирования в Интернет, перед разработчиками программ возникла очень серьезная проблема - как сохранять результаты выполнения алгоритма для каждого конкретного пользователя на некоторое время. Сам по себе, протокол HTTP не имеет возможности фиксирования результатов программных процессов. Использование сессий также не является решением проблемы, так как их действие прекращается сразу после разрыва соединения с сервером.

Проблема разрешилась с внедрением механизма cookies, которые обладают замечательным свойством - они сохраняются на винчестере (жестком диске) пользователя, и могут храниться там практически неограниченное время.

По своей сути cookies - это обычные текстовые файлы, хранящиеся в специальной директории, используемой браузером (обычно эта папка называется Temporary Internet Files), и вы можете увидеть их, зайдя в эту директорию. Быстрый доступ к ней для браузера Internet Explorer осуществляется через пункты Главного меню "Сервис -> Свойства обозревателя -> Временные файлы Интернета -> Настройка -> Просмотр файлов".

Реализация cookie

Реализация механизма cookies в PHP представлена единственной функцией setcookie(). Как и в случае с HTTP заголовками, эта функция должна быть вызвана до отправки каких - либо данных удаленному клиенту. В этой функции не допускаются "пустые" символы, то есть пробел, символы перевода строки и так далее. Функция имеет следующий синтаксис

```
setcookie(имя "куки", значение "куки", срок годности, информация о пути, домен, защищенность)
```

Все параметры, кроме имени cookie, являются необязательными. Если cookie посылается только с этим параметром, то она сразу же уничтожается удаленным клиентом, поэтому сам по себе этот параметр не несет информационной нагрузки. Полнофункциональной cookie делают два следующих параметра - значение, заложенное в "куке" и время (срок годности или expire), до которого эта cookie может быть использована.

Значением, которое несет cookie, может быть любая строка ASCII символов. Например, можно установить cookie с именем и

фамилией посетителя, которые он до этого ввел в поле формы

```
$data = $name."||".$surname;
setcookie ("username", $data);
```

Функция setcookie() возвращает True при успешном выполнении операции и False при сбое. Поэтому и здесь можно использовать функцию die() для определения результата выполняемых действий.

Имени "куки", например, "Test", соответствует HPH переменная с тем же именем - $Test, а значение этой переменной будет совпадать со значением "куки".

Cookie, установленная в вышеуказанном примере, будет уничтожена сразу после закрытия браузера пользователем, так как "по умолчанию" срок жизни cookie устанавливается в ноль. Чтобы изменить этот порядок, необходимо указать третий параметр - expire. Определение этого параметра можно произвести двумя способами

- Задать относительный срок действия с помощью функции time(), к которой прибавляется время в секундах для хранения cookie. Например, чтобы определить cookie на два часа, необходимо написать

```
setcookie("test 1", "это тестовая кука", time() + 3600 * 2);
// 3600 - количество секунд в часе
```

- Второй способ - задание абсолютного срока истечения годности cookie. Он устанавливается с помощью функции mktime(), которая возвращает конкретную дату удаления "куки". Если необходимо задать срок жизни cookie до полуночи 1 декабря 2004 года, то следует определить cookie следующим образом

```
setcookie("test 2", "кука с абсолютной датой удаления", mktime(0, 0, 0, 12, 1, 2004));
```

или

```
$date= mktime(0, 0, 0, 12, 1, 2004);
setcookie("test 2", "кука с абсолютной датой удаления",$date);
```

Следующий, но необязательный параметр пути ограничивает область действия cookie в пределах определенных директорий. Причем в эту область входят все пути, начинающиеся со значения в этом параметре. Например

```
setcookie("test 3, "", 0, "/mus");
```

Мы установили "куку", пропустив параметры значения и времени и определив область действия теми путями, которые начинаются со строки "/mus". В этот путь входят и директория "/music/", и директория "/museums/". Для того чтобы однозначно определить путь, необходимо завершить указанный путь слешем. Таким образом, для ограничения действия куки каталогом "/mus", необходимо было написать "/mus/".

Следующим опциональным, т.е. необязательным параметром является параметр определения действия cookie в пределах указанного домена. Причем значению этого параметра "someserver.com" соответствует только сайт с адресом

```
http://someserver.com
```

а значению ".someserver.com" соответствуют

```
http://someserver.com
http://mail.someserver.com
http://my-someserver.com
```

т.е. все домены, кончающиеся данной строкой.

Последний параметр функции setcookie() указывает на то, что данная cookie должна быть послана через защищенное соединение (HTTPS). Этот параметр необходим при установке cookie с конфиденциальными данными

```
$value="Это пример Cookie";
$cook="my_cookie";
setcookie($cook, $value, time() + 3600 * 24 * 5, "/", ".myphp.dem.ru", 1);
```

Обращение, к уже установленной cookie, выполняется через ее имя. Продолжая приведенный выше пример, прочесть cookie можно следующим образом

```
echo "У вас сохранены следующие данные:<br>";
echo $my_cookie;
```

на экран браузера будет выведено сообщение

```
Это пример Cookie
```

Обращение к данным, сохраненным в cookie, также может производиться через массив $HTTP_COOKIE_VARS (для версий PHP до 4.0.6, где "по умолчанию" register_globals=On). Он схож с другими подобными массивами, такими как $HTTP_POST_VARS и другими, и содержит все значения, прочтенные из cookie.

Удаление cookie производится отправкой новой cookie с именем удаляемой, без каких - либо дополнительных параметров, например

```
setcookie("my_cookie");
echo "Следующие данные были удалены: <br>".$my_cookie;
```

ОБЪЕКТНО - ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

В этой главе мы познакомимся с принципами объектно - ориентированного программирования (ООП) и их реализацией в языке PHP 4. Без понимания ООП будет трудно создать что - то существенное, тем более, что практически все современные языки программирования являются объектно - ориентированными. Поняв эту тему однажды, не придется к ней возвращаться, если вы захотите изучить другой язык.

Вообще говоря, PHP 4 не является в полной мере объектно-ориентированным языком, поэтому очень часто можно обойтись и без использования объектов и классов, но иногда они сильно облегчают жизнь.

Но не стоит, и злоупотреблять ими, так как неоправданно большое количество используемых классов не только затрудняет понимание кода программы, но нередко приводит к снижению ее производительности.

Перейдем теперь к описанию некоторых самых основных принципов построения ООП, объектов и классов.

Принципы ООП

Слова "объектно - ориентированный" подразумевают, что некоторые действия направлены на определенный объект. Объектами в нашей повседневной жизни выступают все окружающие нас предметы - автомобиль, книга, стол, CD - диск и т.д.

Давайте рассмотрим такой привычный для нас объект, как обычный телевизор. Внутри этого объекта находятся множество других объектов - микросхемы, провода, электронно - лучевая трубка и т.д. Но при использовании или взаимодействии с телевизором мы об этом обычно не задумываемся. В этом заключается первый принцип ООП - инкапсуляция или вложенность объектов.

Мы также знаем, что, нажав на определенную кнопку, мы включим телевизор, удерживая или щелкая другую - увеличим или уменьшим громкость. При этом от объекта мы получаем только результат его работы, не задумываясь о его внутренних процессах. Это составляет второй принцип - абстракцию.

Наконец, третий принцип ООП называется наследованием. Он заключается в том, что, например, цветной телевизор произо-

шел от черно - белого, а телевизор с плазменным экраном - от обычного цветного. При этом каждый потомок наследовал свойства и функции предшественника, дополняя их своими, качественно новыми. Наследование позволяет расширить возможности объекта, не создавая при этом новый объект с нуля.

Классы

Класс это коллекция переменных и функций, которые работают с этими переменными. Класс служит шаблоном для любого объекта и создается следующим образом

```
<?
class MyClass
{
// определение класса
}
?>
```

Класс может содержать внутри себя собственные, определяющие этот класс, переменные, которые называются свойствами класса. Кроме того, класс, как правило, содержит функции, которые называются методами класса. Для разграничения методов и свойств следует запомнить, что методы ассоциируются с глаголами в нашем обыденном языке, а свойства - с прилагательными или существительными. Тем самым метод всегда подразумевает действие, а свойство - признак объекта.

Возьмем, например, такой объект, как шариковая ручка. Его свойствами могут являться слова "пластмассовая", "синяя", "новая" и т.д., а методом будет лишь то, что она пишет (конечно, если вы не найдете другой способ ее применения).

Доступ к свойствам и методам класса достигается с помощью указания пути к нему. Элементы пути разделяются знаком стрелки ->. Первым элементом пути является название класса, а вторым - название свойства или метода этого класса. Давайте посмотрим, как это делается

```
class Array_class
{
var $array = array();
// Определение свойства
```

```
    function getUniqSum()
    {
    // Получает сумму уникальных элементов
    return array_sum(array_unique($this->array));
    }

    function getSortedMerge()
    {
    /* Возвращает отсортированный массив из ключей и
элементов массива */
    $result = array_merge(array_keys($this->array), ar-
ray_values($this->array));
    sort($result);
    return $result;
    }
    }
```

Таким образом, мы определили небольшой класс работы с массивами. Такой класс имеет только одно свойство - $array.

Обратите внимание, как мы записали путь к нему. Как уже говорилось, первым элементом пути должно быть название класса, но ввиду того, что это свойство и так находится в самом классе, то название класса меняется на слово this.

Далее мы определяем два метода класса, представляющие собой функции getUniqSum() и getSortedMerge(). Обратите внимание, что свойства классов всегда являются глобальным в пределах этого класса, то есть нам не требовалось в каждой функции писать

```
global $this->array
```

Также заметьте, где ставится знак $ - его место в самом начале описания пути. Причем он ставится даже перед указанием пути к методу класса, т.е. к некоторой функции.

Наследование классов

Теперь создадим новый класс, который будет наследовать все возможности родительского класса, созданного выше. Создание такого класса выполняется ключевым словом extends

```
class Advanced_array extends Array_class
{
function advanced_array($size)
{
/* заполняет массив подряд идущими числами, чередуя их знаки */
$z = 1;
for ($i = 0; $i < $size; $i++)
{
$this->array[$i] = $i *$z;
$z = - $z;
}
}

function getSizeofMerge()
{
/* возвращает число неповторяющихся элементов массива, полученного getSortedMerge() */
$merge = $this->getSortedMerge();
return sizeof(array_unique($merge));
}
}
```

Обратите внимание на функцию с названием самого класса - это так называемый конструктор класса, который автоматически вызывается при создании экземпляра класса.

Теперь рассмотрим ход работы с полученными классами и определение переменных

```
$my = new Array_class;
$my->array = array(1, 2, 6, 1);
echo $my->getUniqSum();
$my = new Advanced_array(4);
echo $my->getSizeofMerge();
```

Сначала мы создаем новый экземпляр класса Array_class и определяем его свойство array, затем выводим сумму элементов без учета повторяющихся (в нашем примере выведется 9). Далее мы создаем экземпляр класса Advanced_array, который наследует все свойства и методы Array_class.

Обратите внимание, что при создании класса не требуется

указывать никаких параметров, но так как у Advanced_array есть конструктор, требующий параметры, то в скобках мы указываем эти данные.

Таким образом, не пришлось определять свойство класса Advanced_array, так как за нас это сделал конструктор. Затем мы применяем метод getSizeofMerge(), который сам по себе использует метод родителя getSortedMerge(), и выводим полученные данные.

ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ

Непосредственно перед запуском программы на PHP, сервер передает ей свои "внутренние" переменные, которые называются "переменными окружения". Любые из этих переменных можно непосредственно использовать в PHP программах - скриптах, получая их значения тем или иным способом.

Переменные сервера

Посмотреть переменные окружения или серверные переменные можно, вставив в любой PHP скрипт простую команду

```
phpinfo();
```

при запуске такого скрипта на экран будет выведена подробная информация о многих характеристиках сервера и самого интерпретатора PHP.

Начинаются эти сведения с общей информации о PHP и сервере, которая приведена на рис.1.

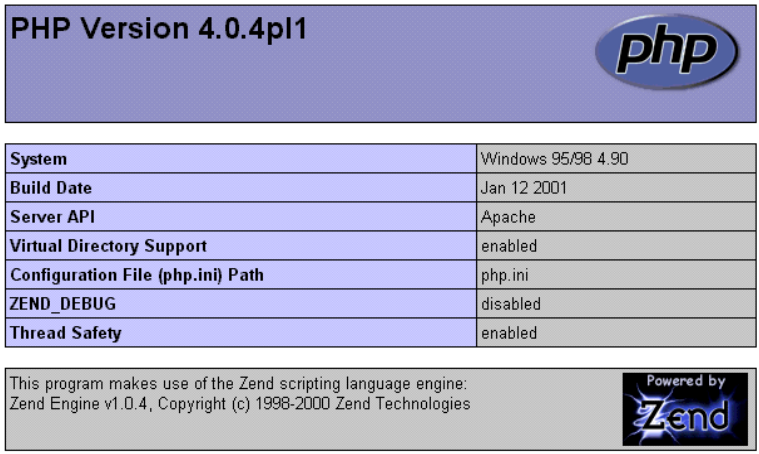


Рис.1 Общая информация о сервере и версии PHP.

Далее идут сведения о настройках файла php.ini, причем, приводятся текущие настройки данного сервера и настройки "по умолчанию", рекомендуемые разработчиками PHP.

Variable	Value
COMSPEC	C:\WINDOWS\COMMAND.COM
CONTENT_LENGTH	24
CONTENT_TYPE	application/x-www-form-urlencoded
DOCUMENT_ROOT	c:/web/home/kau/www
HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	ru
HTTP_CONNECTION	Keep-Alive
HTTP_HOST	kau
HTTP_REFERER	http://kau/Admin/Admin.php
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
PATH	C:\PROGRAMF\ABRIAM~1 \PERL\BIN;C:\PROGRAMF\ABRIAMER\PERL\BIN;C:\WINDOWS;C:\WINDOWS\COMMAND;C:\;d:\NC5R;C:\GRAF
REMOTE_ADDR	127.0.0.1
REMOTE_PORT	1064
SCRIPT_FILENAME	c:/web/home/kau/admin/adm.php
SERVER_ADDR	127.0.0.1
SERVER_ADMIN	you@your.address
SERVER_NAME	kau
SERVER_PORT	80
SERVER_SIGNATURE	*Apache/1.3.14 Server at kau Port 80*
SERVER_SOFTWARE	Apache/1.3.14 (Win32) mod_perl/1.24_01 PHP/4.0.4pl1 DAV/1.0.2
WINDIR	C:\WINDOWS
GATEWAY_INTERFACE	CGI/1.1
SERVER_PROTOCOL	HTTP/1.1
REQUEST_METHOD	POST
QUERY_STRING	
REQUEST_URI	/Admin/adm.php
SCRIPT_NAME	/Admin/adm.php

Рис.2. Переменные окружения сервера.

Затем приводится список переменных окружения сервера (Apache Environment), которые можно непосредственно использовать в своих PHP программах (рис.2). Некоторые из них носят общий характер и дают характеристики системы в целом, другие содержат сведения об используемом сервере, а третьи непосредственно относятся к данному скрипту.

Приведем краткое описание некоторых из переменных окружения сервера, которые используются наиболее часто

HTTP_REFERER

- адрес страницы, с которой был осуществлен переход на текущую страницу.

HTTP_USER_AGENT

- представляет собой строку, содержащую версию используемого клиентского навигатора.

REMOTE_PORT

- номер порта, используемого клиентской машиной для связи с сервером.

REMOTE_ADDR

- IP - адрес клиента, запрашивающего доступ к странице.

SCRIPT_FILENAME

- путь к текущей PHP - программе.

REQUEST_METHOD

- содержит метод, используемый для доступа к странице. Может принимать значения - GET, POST, HEAD, PUT.

QUERY_STRING

- состав адресной строки запроса, с помощью которого была вызвана текущая страница.

Предопределенные PHP переменные

После серверных переменных приводится состав и имена переменных, с которыми непосредственно работает интерпретатор PHP (PHP variables). Примерный состав этих переменных для сервера Apache 1.3 и PHP версии 4.04 приведен на рис.3. Такие переменные называют предопределенными переменными интерпретатора. Они содержатся в ассоциативных PHP массивах, вида $HTTP_SERVER_VARS[], где в качестве индекса массива используется имя PHP переменной или переменной окружения сервера.

Теперь мы должны понять и по возможности научиться использовать эти переменные в PHP скриптах (сценариях). Нужно хорошо знать способ доступа к ним - это необходимо для дальнейшего понимания программ на языке PHP.

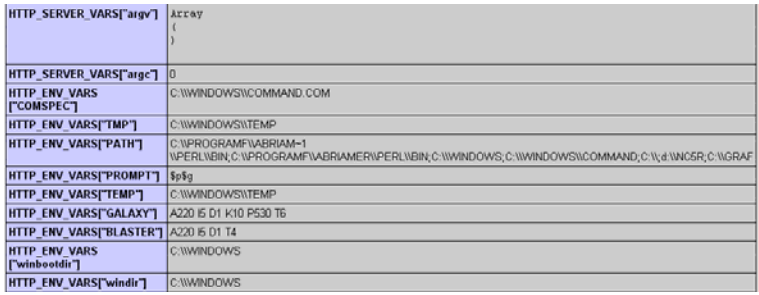
Перейдем теперь к описанию некоторых предопределенных (определенных "по умолчанию") PHP переменных

$HTTP_GET_VARS[]

- ассоциативный массив переменных, переданных текущей программе методом GET.

PHP Variables

Variable	Value
PHP_SELF	/Admin/adm.php
HTTP_POST_VARS["adm"]	1
HTTP_POST_VARS["pass"]	1
HTTP_POST_VARS["ok"]	OK
HTTP_SERVER_VARS ["COMSPEC"]	C:\\WINDOWS\\COMMAND.COM
HTTP_SERVER_VARS ["CONTENT_LENGTH"]	24
HTTP_SERVER_VARS ["CONTENT_TYPE"]	application/x-www-form-urlencoded
HTTP_SERVER_VARS ["DOCUMENT_ROOT"]	c:/web/home/kau/www
HTTP_SERVER_VARS ["HTTP_ACCEPT"]	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
HTTP_SERVER_VARS ["HTTP_ACCEPT_ENCODING"]	gzip, deflate
HTTP_SERVER_VARS ["HTTP_ACCEPT_LANGUAGE"]	ru
HTTP_SERVER_VARS ["HTTP_CONNECTION"]	Keep-Alive
HTTP_SERVER_VARS ["HTTP_HOST"]	kau
HTTP_SERVER_VARS ["HTTP_REFERER"]	http://kau/Admin/Admin.php
HTTP_SERVER_VARS ["HTTP_USER_AGENT"]	Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
HTTP_SERVER_VARS["PATH"]	C:\\PROGRAMF\\ABRIAM~1\\PERL\\BIN;C:\\PROGRAMF\\ABRIAMER\\PERL\\BIN;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C:\\;d:\\NC5R;C:\\GRAF
HTTP_SERVER_VARS ["REMOTE_ADDR"]	127.0.0.1
HTTP_SERVER_VARS ["REMOTE_PORT"]	1064
HTTP_SERVER_VARS ["SCRIPT_FILENAME"]	c:/web/home/kau/admin/adm.php
HTTP_SERVER_VARS ["SERVER_ADDR"]	127.0.0.1
HTTP_SERVER_VARS ["SERVER_ADMIN"]	you@your.address
HTTP_SERVER_VARS ["SERVER_NAME"]	kau
HTTP_SERVER_VARS ["SERVER_PORT"]	80
HTTP_SERVER_VARS ["SERVER_SIGNATURE"]	*Apache/1.3.14 Server at kau Port 80*
HTTP_SERVER_VARS ["SERVER_SOFTWARE"]	Apache/1.3.14 (Win32) mod_perl/1.24_01 PHP/4.0.4pl1 DAV/1.0.2
HTTP_SERVER_VARS ["WINDIR"]	C:\\WINDOWS
HTTP_SERVER_VARS ["GATEWAY_INTERFACE"]	CGI/1.1
HTTP_SERVER_VARS ["SERVER_PROTOCOL"]	HTTP/1.1
HTTP_SERVER_VARS ["REQUEST_METHOD"]	POST
HTTP_SERVER_VARS ["QUERY_STRING"]	
HTTP_SERVER_VARS ["REQUEST_URI"]	/Admin/adm.php
HTTP_SERVER_VARS ["SCRIPT_NAME"]	/Admin/adm.php
HTTP_SERVER_VARS ["PATH_TRANSLATED"]	c:/web/home/kau/admin/adm.php
HTTP_SERVER_VARS ["PHP_SELF"]	/Admin/adm.php



HTTP_SERVER_VARS["argv"]	Array ( )
HTTP_SERVER_VARS["argc"]	0
HTTP_ENV_VARS["COMSPEC"]	C:\\WINDOWS\\COMMAND.COM
HTTP_ENV_VARS["TMP"]	C:\\WINDOWS\\TEMP
HTTP_ENV_VARS["PATH"]	C:\\PROGRAMF\\ABRIAM~1 \\PERL\\BIN;C:\\PROGRAMF\\ABRIAMER\\PERL\\BIN;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C:\\;d:\\NC5R;C:\\GRAF
HTTP_ENV_VARS["PROMPT"]	pg
HTTP_ENV_VARS["TEMP"]	C:\\WINDOWS\\TEMP
HTTP_ENV_VARS["GALAXY"]	A220 I5 D1 K10 P530 T6
HTTP_ENV_VARS["BLASTER"]	A220 I5 D1 T4
HTTP_ENV_VARS["winbootdir"]	C:\\WINDOWS
HTTP_ENV_VARS["windir"]	C:\\WINDOWS

Рис.3. Переменные PHP.

$HTTP_POST_VARS[]

- ассоциативный массив переменных, переданных текущей программе методом POST.

$HTTP_SERVER_VARS[]

- ассоциативный массив переменных, переданных текущей PHP программе от HTTP сервера.

$GLOBALS[]

- ассоциативный массив всех определенных, на настоящий момент, PHP переменных.

Для вывода на экран всех значений формы можно использовать именно эти, предопределенные переменные. Это зарезервированные переменные, которые сами берут значения из окружения сервера (Server Environment). Такие переменные можно выводить функцией PRINT или PRINT_R, а не только ECHO. Например

```
<? print_r($HTTP_POST_VARS); ?>
```

Для вывода всех переменных, определенных в данном скрипте можно использовать команду

```
<? print_r($GLOBALS); ?>
```

Возможен и другой способ вывода на экран переменных окружения сервера или предопределенных переменных. Для этого нужно вставить в скрипт строку вида

```
echo "Адрес ссылки: $HTTP_REFERER";
```

Для того, чтобы непосредственно обратиться к некоторой переменной окружения в самой PHP программе нужно использовать ее имя в качестве индекса ассоциативного массива, заключенного в кавычках. Например

```
$ref=$HTTP_SERVER_VARS["HTTP_REFERER"];
```

Приведем и другой способ вывода таких переменных на примере одной из них и напишем сценарий, выдающий значение выбранной переменной в самой PHP программе. Обратите внимание на то, что аналогично можно получить значение любой из указанных на рис.2 величин, например

```
HTTP_ACCEPT_LANGUAGE,
HTTP_USER_AGENT,
HTTP_HOST,
WINDIR,
SERVER_SOFTWARE,
SERVER_NAME,
SERVER_ADDR,
SERVER_PORT,
REMOTE_ADDR,
QUERY_STRING
```

Рассмотрим переменную REMOTE_ADDR или HTTP_USER_AGENT. Программа - сценарий для получения значения REMOTE_ADDR может иметь вид

```
<?
$addr = getenv ("REMOTE_ADDR");
echo "Ваш IP адрес: $addr";
?>
```

Здесь была использована функция getenv(), чтобы присвоить значение переменной окружения REMOTE_ADDR переменной

PHP с именем $addr. Для того, чтобы вывести на экран значение этой величины использована обычная команда echo "Ваш IP адрес: $addr".

Аналогично, напишем программу для вывода значения переменной HTTP_USER_AGENT, которая будет иметь вид

```
<?
$agent = getenv ("HTTP_USER_AGENT");
echo "Вы используете: $agent";
?>
```

Все указанные выше переменные платформо - зависимы и зависят от операционной системы, установленного сервера и его конфигурации.

Передача параметров

Значения переменных PHP можно передавать не только через формы, но и как обычные параметры адресной строки, записывая их определенным образом после имени скрипта. Рассмотрим эту возможность более подробно и разберем случаи, когда это вообще допустимо.

Напишем на языке PHP простой сценарий - принять имя и возраст пользователя и выдать полученную информацию через обычную HTML программу. Наиболее простой способ передачи данных сценарию - это непосредственный набор их в URL строке браузера. Например, это может быть такой набор

```
http://servername/script.php?name=Serg&age=50
```

PHP сценарий должен распознать два параметра - name и age, и вывести на экран следующую информацию

```
Привет, name! Я знаю Вас!
Вам age лет, не так ли?!
```

Рассмотрим вначале более простую, но очень полезную задачу. Как получить в сценарии строку параметров, переданную после знака вопроса "?". Для этого нужно проанализировать переменную окружения QUERY_STRING, которая в PHP доступна под именем $QUERY_STRING.

В следующем примере показано, как это можно сделать

```
<html>
<body>
<?
echo "Данные из командной строки: $QUERY_STRING";
?>
</body>
</html>
```

Запишем этот сценарий в файл test.php в корневом каталоге сайта. Запустим его, набрав командную строку в браузере

```
http://localhost/test.php?aaa+bbb+ccc+ddd
```

Сценарий должен выдать строку

```
Данные из командной строки: aaa+bbb+ccc+ddd
```

Однако, выдача такой строки будет зависеть от настроек в файле конфигурации php.ini. Если там включена опция register_globals = On, что обычно устанавливалось в версиях PHP до 4.2 , то написанная программа будет работать. В противном случае, программу следует изменить следующим образом

```
<html>
<body>
<?
$query = getenv ("QUERY_STRING");
echo "Данные из командной строки: $query";
?>
</body>
</html>
```

Строку

```
$query = getenv ("QUERY_STRING");
```

можно записать и так

```
$query =$HTTP_SERVER_VARS["QUERY_STRING"];
```

В более новых версиях PHP (после версии 4.2, когда параметр register_globals обычно находится в режиме Off) введены новые предопределенные PHP переменные, использование которых считается более предпочтительным. Теперь вместо

```
$HTTP_SERVER_VARS
```

нужно писать

```
$query = $_SERVER["QUERY_STRING"]
```

хотя и прежняя переменная $HTTP_SERVER_VARS и функция getenv() также могут еще использоваться, но при условии register_globals = On.

Опция register_globals

Параметр register_globals находится в файле настроек php.ini, который обычно хранится в папке C:\windows или в папке с установленным интерпретатором PHP. Отключение этого параметр (режим Off) является еще одним средством PHP для повышения безопасности всей системы. Режим Off отключает внедрение пользовательских переменных в среду переменных программы на PHP. Опция полностью изолирует внутренние переменные от предоставляемых пользователем данных.

Во всех руководствах по PHP и в предыдущих параграфах обычно написано, что данные, полученные из формы, или переданные из адресной строки

```
test.php?per=pel&var=val
```

автоматически становятся переменными PHP

```
$per
```

и

```
$var
```

со значениями

pel

и

val

Это правильно, если в php.ini установлено register_globals = On, тогда все полученные скриптом данные будут назначены соответствующим переменным, но такой скрипт легко взломать. При выключенной опции register_globals = Off данные из командной строки или из формы в PHP скрипт передаются, но при этой настройке, интерпретатор PHP не превращает их в переменные.

В последних версиях PHP, обычно начиная с 4.2, в целях безопасности PHP "по умолчанию" настраивается так, чтобы переданные значения автоматически не назначались переменным. Если в файле php.ini опция register_globals = Off, то можно получить значение переменной, обратившись к массиву, соответствующему способу передачи данных в скрипт - GET или POST.

Например, при передаче параметров через форму методом GET или в командной строке

test.php?var=val

строка скрипта

```
echo $_REQUEST['var'];
```

выведет на экран значение

val

Можно использовать и такую запись

```
echo $_GET['var'];
```

Если мы получаем данные из формы, отправленной методом POST, то все поля этой формы содержатся в предопределенном массиве $_POST.

То же самое касается и серверных переменных, таких, как REMOTE_ADDR, PHP_SELF. В режиме register_globals = Off в

PHP программе их не существует, и получить их значения можно, обратившись к массивам $_SERVER, $_ENV

```
echo $_SERVER[' REMOTE_ADDR '];
```

или функцией getenv

```
echo getenv('HTTP_REFERER');
```

Разработчики PHP настоятельно рекомендуют выключать register_globals, и пользоваться данными, полученными от пользователя, только обращаясь к соответствующим массивам.

Если нет возможности переделать готовый скрипт, а register_globals находится в режиме Off, то можно воспользоваться функциями extract и import_request_variables, введя в PHP скрипт приведенный ниже код

```
import_request_variables("GPC");
extract($_SERVER);
```

Причем писать его нужно именно в таком порядке - вначале назначаются переменные, пришедшие от пользователя, а потом - предопределенные серверные переменные, чтобы первые не могли заменить одноименные серверные.

Функция

```
extract (var_array [, extract_type [, prefix]])
```

используется для импорта переменных из ассоциативного массива var_array в текущую таблицу символов любого скрипта, рассматривая ключи массива, как имена переменных, а значения, как значения этих переменных. Для каждой пары key/value она создает переменную в текущей таблице символов с учетом заданных необязательных параметров extract_type и prefix. Причем, начиная с PHP версии 4.0.5, сама функция возвращает количество извлеченных переменных.

Функция extract() проверяет каждый ключ/key - образует ли он верное имя переменной, а также смотрит, нет ли конфликтов с существующими именами в таблице символов данного скрипта. Способ, которым обрабатываются неверные ключи и конфликты переменных, определяется параметром extract_type, который мо-

жет принимать следующие значения

EXTR_OVERWRITE - если есть конфликт, существующая в скрипте переменная перезаписывается.

EXTR_SKIP - если есть конфликт, существующая переменная не перезаписывается.

EXTR_PREFIX_SAME - если есть конфликт, вставить префикс prefix перед именем извлекаемой переменной.

EXTR_PREFIX_ALL - вставить префикс prefix перед всеми именами извлекаемых переменных. Начиная с версии PHP 4.0.5, это могут быть и числа.

EXTR_PREFIX_INVALID - вводить префикс prefix только перед неправильными именами переменных. Эта возможность была добавлена в PHP 4.0.5.

EXTR_IF_EXISTS - перезаписывать переменную, только в том случае, если она уже существует в текущей таблице символов скрипта, иначе - ничего не делать. Этот режим используется для определения списка правильных переменных и последующего извлечения только тех переменных, которые вы определили вне массива $_REQUEST. Эта возможность была добавлена в PHP версии 4.2.0.

EXTR_PREFIX_IF_EXISTS - создавать префиксированные имена переменных только с том случае, если версия без префикса той же самой переменной существует в текущей таблице символов. Этот режим также был добавлен в PHP 4.2.0.

Если extract_type не определен, "по умолчанию" принимается, что он EXTR_OVERWRITE.

Новый метод передачи переменных

Интерпретатор PHP версии 4.2 и выше, продолжает поддерживать старые способы передачи данных в скрипт. Так что любые старые приложения будут работать без внесения каких - то изменений, если установить режим register_globals = On.

Кроме них введены и используются несколько другие переменные окружения, которые будут работать и в режиме register_globals = Off. Все они представляют собой массивы, так что и обращаться к ним нужно, как к массивам.

$_GET - содержит переменные, пришедшие по методу GET

или из командной строки. Аналогичен старому массиву $HTTP_GET_VARS (который ещё доступен, но не рекомендуется).

$_POST - содержит переменные, пришедшие по методу POST. Аналогичен старому массиву $HTTP_POST_VARS (который ещё доступен, но не рекомендуется).

$_COOKIE - содержит переменные, предоставляемые скрипту через HTTP cookies. Аналогичен старому массиву $HTTP_COOKIE_VARS (который ещё доступен, но не рекомендуется).

$_SERVER - содержит переменные окружения сервера (например, REMOTE_ADDR). Это переменные, которые устанавливаются Web - сервером и относятся к среде окружения выполнения текущего скрипта. Аналогичен старому массиву $HTTP_SERVER_VARS (который ещё доступен при register_globals = On, но не рекомендуется).

$_ENV - содержит переменные окружения. Переменные, предоставляемые скрипту через среду окружения. Аналогичен старому массиву $HTTP_ENV_VARS (который ещё доступен, но не рекомендуется).

$_REQUEST - содержит все переменные GET, POST, и COOKIE, т.е. все, что пришло от пользователя, и которому, в смысле безопасности, доверять не рекомендуется. Этот массив не имеет прямых аналогов в версиях PHP до 4.1.

$_SESSION - содержит HTTP переменные, зарегистрированные на данный момент в сессии скрипта. Аналогичен старому массиву $HTTP_SESSION_VARS (который ещё доступен, но не рекомендуется).

$GLOBALS - содержит ссылку на каждую переменную, доступную в данный момент в глобальной области видимости данного скрипта. Ключами этого массива являются имена глобальных переменных.

$_FILES - переменные, предоставляемые скрипту через

HTTP POST - загрузку файлов. Аналогичен старому массиву $HTTP_POST_FILES (который ещё доступен, но не рекомендуется).

Приведем теперь результат работы функции phpinfo() для PHP версии 4.3.5, которые показаны на рис.4. На этом рисунке приведены переменные окружения сервера Apache версии 1.3.

Переменные интерпретатора PHP 4.3.5 приведены ниже на рис.5. Эти переменные автоматически являются глобальными в любых функциях и областях видимости и вы можете обращаться к ним из любого места скрипта. Например функция

```
function exam(name)
{print $_GET["name"];}
```

будет нормально работать.

Apache Environment

Variable	Value
COMSPEC	C:\WINDOWS\COMMAND.COM
CONTENT_LENGTH	24
CONTENT_TYPE	application/x-www-form-urlencoded
DOCUMENT_ROOT	c:/web/home/kau/www
HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	ru
HTTP_CONNECTION	Keep-Alive
HTTP_HOST	kau
HTTP_REFERER	http://kau/Admin/Admin.php
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
PATH	C:\PROGRAMF\ABRIAM~1\PERL\BIN;C:\PROGRAMF\ABRIAMER\PERL\BIN;C:\WINDOWS;C:\WINDOWS\COMMAND;C:\;d:\NC5R;C:\GRAF
REMOTE_ADDR	127.0.0.1
REMOTE_PORT	1074
SCRIPT_FILENAME	c:/web/home/kau/admin/adm.php
SERVER_ADDR	127.0.0.1
SERVER_ADMIN	you@your.address
SERVER_NAME	kau
SERVER_PORT	80
SERVER_SIGNATURE	<ADDRESS>Apache/1.3.14 Server at kau Port 80</ADDRESS>
SERVER_SOFTWARE	Apache/1.3.14 (Win32) mod_perl/1.24_01 PHP/4.3.5RC3 DAV/1.0.2
WINDIR	C:\WINDOWS
GATEWAY_INTERFACE	CGI/1.1
SERVER_PROTOCOL	HTTP/1.1
REQUEST_METHOD	POST
QUERY_STRING	*no value*
REQUEST_URI	/Admin/adm.php
SCRIPT_NAME	/Admin/adm.php

Рис.4. Переменные сервера.

PHP Variables

Variable	Value
_REQUEST["adm"]	1
_REQUEST["pass"]	2
_REQUEST["ok"]	OK
_POST["adm"]	1
_POST["pass"]	2
_POST["ok"]	OK
_SERVER["COMSPEC"]	C:\\WINDOWS\\COMMAND.COM
_SERVER ["CONTENT_LENGTH"]	24
_SERVER["CONTENT_TYPE"]	application/x-www-form-urlencoded
_SERVER ["DOCUMENT_ROOT"]	c:/web/home/kau/www
_SERVER["HTTP_ACCEPT"]	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
_SERVER ["HTTP_ACCEPT_ENCODING"]	gzip, deflate
_SERVER ["HTTP_ACCEPT_LANGUAGE"]	ru
_SERVER ["HTTP_CONNECTION"]	Keep-Alive
_SERVER["HTTP_HOST"]	kau
_SERVER["HTTP_REFERER"]	http://kau/Admin/Admin.php
_SERVER ["HTTP_USER_AGENT"]	Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
_SERVER["PATH"]	C:\\PROGRAMF\\ABRIAM~1 \\PERL\\BIN;C:\\PROGRAMF\\ABRIAMER\\PERL\\BIN;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C:\\;d:\\NC5R;C:\\GRAF
_SERVER["REMOTE_ADDR"]	127.0.0.1
_SERVER["REMOTE_PORT"]	1074
_SERVER["SCRIPT_FILENAME"]	c:/web/home/kau/admin/adm.php
_SERVER["SERVER_ADDR"]	127.0.0.1
_SERVER["SERVER_ADMIN"]	you@your.address
_SERVER["SERVER_NAME"]	kau
_SERVER["SERVER_PORT"]	80
_SERVER ["SERVER_SIGNATURE"]	<ADDRESS>Apache/1.3.14 Server at kau Port 80</ADDRESS>
_SERVER ["SERVER_SOFTWARE"]	Apache/1.3.14 (Win32) mod_perl/1.24_01 PHP/4.3.5RC3 DAV/1.0.2
_SERVER["WINDIR"]	C:\\WINDOWS
_SERVER ["GATEWAY_INTERFACE"]	CGI/1.1
_SERVER ["SERVER_PROTOCOL"]	HTTP/1.1
_SERVER ["REQUEST_METHOD"]	POST
_SERVER["QUERY_STRING"]	*no value*
_SERVER["REQUEST_URI"]	/Admin/adm.php
_SERVER["SCRIPT_NAME"]	/Admin/adm.php
_SERVER ["PATH_TRANSLATED"]	c:/web/home/kau/admin/adm.php
_SERVER["PHP_SELF"]	/Admin/adm.php
_SERVER["argv"]	Array ( )
_SERVER["argc"]	0
_ENV["COMSPEC"]	C:\\WINDOWS\\COMMAND.COM
_ENV["TMP"]	C:\\WINDOWS\\TEMP
_ENV["PATH"]	C:\\PROGRAMF\\ABRIAM~1 \\PERL\\BIN;C:\\PROGRAMF\\ABRIAMER\\PERL\\BIN;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C:\\;d:\\NC5R;C:\\GRAF
_ENV["PROMPT"]	pg
_ENV["TEMP"]	C:\\WINDOWS\\TEMP
_ENV["GALAXY"]	A220 IS D1 K10 PS30 T6
_ENV["BLASTER"]	A220 IS D1 T4
_ENV["winbootdir"]	C:\\WINDOWS
_ENV["windir"]	C:\\WINDOWS

Рис.5. Переменные PHP.

Для просмотра содержимого массивов всех PHP переменных можно использовать следующий код

```
<pre>
<?
echo '<FONT COLOR = RED>
<HR><BR>$_SERVER<BR>';
var_dump($_SERVER);
echo '<HR><BR>$_POST<BR>';
var_dump($_POST);
echo '<HR><BR>$_GET<BR>';
var_dump($_GET);
echo '<HR><BR>$_ENV<BR>';
var_dump($_ENV);
echo '<HR><BR>$_REQUEST<BR>';
var_dump($_REQUEST);
echo '<HR><BR>$_SESSION<BR>';
var_dump($_SESSION);
?>
</pre>
```

Который выведет на экран информацию вида

```
$_SERVER
array(32) {
["COMSPEC"]=>
string(24) "C:\\WINDOWS\\COMMAND.COM"
["CONTENT_LENGTH"]=>
string(2) "22"
["CONTENT_TYPE"]=>
string(33) "application/x-www-form-urlencoded"
["DOCUMENT_ROOT"]=>
string(19) "c:/web/home/kau/www"
["HTTP_ACCEPT"]=>
string(3) "*/*"
["HTTP_ACCEPT_ENCODING"]=>
string(13) "gzip, deflate"
["HTTP_ACCEPT_LANGUAGE"]=>
string(2) "ru"
["HTTP_CONNECTION"]=>
string(10) "Keep-Alive"
```

```
["HTTP_HOST"]=>
string(3) "kau"
["HTTP_REFERER"]=>
string(26) "http://kau/Admin/Admin.php"
["HTTP_USER_AGENT"]=>
string(59) "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98;
Win 9x 4.90)"
["PATH"]=>
string(123)
"C:\\PROGRAMF\\ABRIAM~1\\PERL\\BIN;C:\\PROGRAMF\\AB
RIA-
MER\\PERL\\BIN;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C
:\\;d:\\NC5R;C:\\GRAF"
["REMOTE_ADDR"]=>
string(9) "127.0.0.1"
["REMOTE_PORT"]=>
string(4) "1033"
["SCRIPT_FILENAME"]=>
string(29) "c:/web/home/kau/admin/adm.php"
["SERVER_ADDR"]=>
string(9) "127.0.0.1"
["SERVER_ADMIN"]=>
string(16) "you@your.address"
["SERVER_NAME"]=>
string(3) "kau"
["SERVER_PORT"]=>
string(2) "80"
["SERVER_SIGNATURE"]=>
string(55) "Apache/1.3.14 Server at kau Port 80"
["SERVER_SOFTWARE"]=>
string(61) "Apache/1.3.14 (Win32) mod_perl/1.24_01
PHP/4.3.5RC3 DAV/1.0.2"
["WINDIR"]=>
string(11) "C:\\WINDOWS"
["GATEWAY_INTERFACE"]=>
string(7) "CGI/1.1"
["SERVER_PROTOCOL"]=>
string(8) "HTTP/1.1"
["REQUEST_METHOD"]=>
string(4) "POST"
["QUERY_STRING"]=>
```

```
string(0) ""
["REQUEST_URI"]=>
string(14) "/Admin/adm.php"
["SCRIPT_NAME"]=>
string(14) "/Admin/adm.php"
["PATH_TRANSLATED"]=>
string(29) "c:/web/home/kau/admin/adm.php"
["PHP_SELF"]=>
string(14) "/Admin/adm.php"
["argv"]=>
array(0) { }
["argc"]=>
int(0)
}


$_POST
array(3) {
["adm"]=>
string(0) ""
["pass"]=>
string(0) ""
["ok"]=>
string(8) " OK "
}


$_GET array(0) {
}


$_ENV
array(9) {
["COMSPEC"]=>
string(24) "C:\\WINDOWS\\COMMAND.COM"
["TMP"]=>
string(17) "C:\\WINDOWS\\TEMP"
["PATH"]=>
string(123)
"C:\\PROGRAMF\\ABRIAM~1\\PERL\\BIN;C:\\PROGRAMF\\AB
RIA-
```

```
MER\\PERL\\BIN;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C
:\\;d:\\NC5R;C:\\GRAF"
    ["PROMPT"]=>
    string(5) "$p$g "
    ["TEMP"]=>
    string(17) "C:\\WINDOWS\\TEMP"
    ["GALAXY"]=>
    string(22) "A220 I5 D1 K10 P530 T6"
    ["BLASTER"]=>
    string(13) "A220 I5 D1 T4"
    ["winbootdir"]=>
    string(11) "C:\\WINDOWS"
    ["windir"]=>
    string(11) "C:\\WINDOWS"
    }


    $_REQUEST
    array(3) {
    ["adm"]=>
    string(0) ""
    ["pass"]=>
    string(0) ""
    ["ok"]=>
    string(8) " OK "
    }


    $_SESSION
    NULL
```

Добавление переменных в массив $_SESSION автоматически регистрирует эти переменные в сессии, также как это делалось с помощью функции session_register().

ЛИТЕРАТУРА

1. PHP and MySQL. // http://phpclub.unet.ru.
2. Сайт авторов PHP. // http://www.php.org.
3. Документация по PHP. // http://www.citforum.ru.
4. Платонов К. // http://www.mjk.msk.ru/~dron/html/php.shtml.
5. Руководство по PHP. // http://phpru.net.
6. Trachtenberg A., Sklar D. (Перевод Н.В. Костроминой) - Ведение в PHP // http://myphp.net.ru/doc/index.php.
7. Уроки по PHP. // http://www.myphp.dem.ru/lessons/index.php?1.
8. Описание PHP 4. http://www.php-nuke.ru.
9. Описания скриптов. // http://www.codenet.ru.
10. Описания скриптов. // http://php.net/register_globals.
11. Кухарчик А. - Создание сайтов. // http://virtual.bresttelecom.by/php/.
12. Климант Ю.В. - Уроки по Web - программированию на PHP 4. // http://ipg.h1.ru/aboutme/aboutme.html
13. Бородин Д. - Вопросы безопасности PHP. // http://faq.phpclub.net/security_ changes4.1.txt.
14. Котеров Д.В. - Самоучитель PHP 4. // СПб., "БХВ - Петербург", 2001, 576с.
15. Мелони Д. - PHP 4 в действии. // М., "Лучшие книги", 2002, 400с.
16. Зандстра М. - Освой самостоятельно PHP 4 за 24 часа. // М., "Вильямс", 2001, 384с.
17. Кузнецов С.Д. - PHP 4.0. Руководство пользователя. // М., "Майор", 2001. 176с.
18. Ратшиллер Т., Геркен Т. - PHP 4. Разработка Web-приложений. // СПб., "Питер", 2001, 384с.
19. Костарев А.Ф. - PHP в Web-дизайне. // СПб., "БХВ - Петербург", 2002, 592с.
20. Лежнин Ф. - Функция date() - вывод даты и времени в PHP. // www.webclub.ru.
21. Бородин Д. - Примеры форматирования текста с помощью printf/sprintf // http://www.php.spb.ru.

ПРИЛОЖЕНИЕ

Регулярные выражения

Регулярные выражения - это самый мощный инструмент работы со строками, который смогли придумать современные программисты. С их помощью можно проводить анализ строк на содержание последовательностей символов, производить замену на основе этой выборки, разбивать строки на массивы и многое другое.

Регулярное выражение представляет собой некий шаблон, который используется для различных действий в зависимости от функции, использующей это регулярное выражение. Для задания шаблона используются специальные символы. Ниже представлена краткая таблица специальных символов и их значений.

Символы	***Значение***	***Примеры***
Символы, указывающие расположение элемента в строке		
^	Указывает на то, что символы после этого знака должны находиться в начале строки - маркер начала строки.	^Заголовок или ^abc - строка, начинающаяся с "abc".
$	Символы до этого знака должны находиться в конце строки - маркер конца строки.	Содержание$ или abc$ - строка, заканчивающаяся на "abc".
Escape последовательности		
\.	Шаблону соответствует знак точки в строке.	Конец строки\.
\n	Символ перевода строки.	Первая строка\nВторая строка
\r	Символ возврата каретки.	Текст\r
\t	Символ табуляции.	\tКрасная строка
\v	Символ вертикальной табуляции.	\vТекст
Задание группы символов		
[ ]	Задают группу символов. Соответствует любому символу из перечисленных в группе. Есть возможность задания диапазона символов с помощью знака "-".	[abchyt] [а-я] [a-z] [123]
^	В группе символов соответствует	[^\n\t]

	отрицанию последующих символов, то есть указывает символы, не соответствующие шаблону.	
.	Соответствует любому символу, кроме перевода строки "\n".	.омпьютер
Количественные показатели		
*	Символ перед знаком не присутствует либо повторяется любое число раз.	Текст\n*Дальше текст или шаблону w* соответствуют строки what, agwt
?	Символ перед знаком встречается ноль или один раз.	Длинн?ое или шаблону w?r соответствуют строки ara, awra.
+	Предыдущий символ повторяется один или большее число раз.	100+ или шаблону w+ соответствуют строки what, agwt
{n}	Символ перед знаком повторяется n-ое число раз.	Длин{2}оше{3}
{min,max}	Задает диапазон числа повторений предыдущего символа.	^ab{3,7}
{min,}	Предыдущий символ повторяется min или большее число раз.	Слово.{5,}
Логическое определение		
|	Эффект подобен оператору || (or) в логическом выражении.	Раз|Два|Три
( )	Логическая группировка выражений.	(Может) + Повторяться
\	Следующий символ является специальным. Так же применяется для указания символов, которые могут использоваться в качестве модификаторов.	\n - соответствует символу перевода строки * - символ "*", а * - модификатор

Язык PHP располагает, как собственным механизмом работы с регулярными выражениями (POSIX), так и способами, заимствованными у другого серверного языка программирования Perl. Внешне их легко различить по названиям функций

- Функции первого типа (POSIX), т.е. из языка PHP начинаются с символов "ereg".
- Функции второго, из Perl начинаются с "preg".

Но названия функций не единственное их отличие - прежде всего они содержат некоторые различия в синтаксисе регулярных выражений. В частности, Perl - подобные функции требуют дополнительного разделителя

```
$str="регулярное выражение";
// это просто символьная строка
$preg=preg_replace("/р.+е/i","<i>[вырезано]</i>", $str);
$ereg=eregi_replace("р.+е", "<i>[вырезано]</i>", $str);
echo $preg."<br>".$ereg;
```

Как видно, мы используем функции замены части строки с помощью регулярных выражений. Обратите внимание на шаблон функции preg_replace - в качестве разделителя здесь выступают слеши, причем после закрывающего разделителя следует модификатор i, указывающий, что шаблон является нечувствительным к регистру. Тот же эффект достигается при использовании POSIX функции с суффиксом i, например, eregi_replace.

Результат выполнения этих функций одинаков

[вырезано]
[вырезано]

Кратко рассмотрим другие функции работы с регулярными выражениями.

Функции	***Синтаксис***	***Описание***
ereg, eregi, preg_match	функция(pattern, string, [regs])	Ищет в строке string соответствия с регулярным выражением pattern, и сохраняет их в массиве regs (если указано).
preg_match_all	preg_match_all(pattern, subject, matches, [order])	Осуществляет глобальное сопоставление с шаблоном,

		результаты заносит в matches.
split, spliti, preg_split	функция(pattern, string, [limit])	Разбивает строку в массив посредством регулярного выражения.
preg_grep	preg_grep(pattern, input)	Возвращает массив из элементов массива input, соответствующих шаблону pattern.

Рассмотрим теперь пример перевода времени в стандартное время Unix. Предположим, у нас имеется дата в формате

часы:минуты:секунды - день.месяц.год

Но нам потребовалось отображать дату в виде

день.месяц.год часы:минуты

Напишем сценарий, который будет на первом этапе приводить дату к виду

часы:минуты:секунды месяц/день/год

а затем с помощью функции strtotime() переведем эту запись в стандартное время UNIX, которое можно отображать в любом виде

```
$str = "10:50:30 - 10.10.04";
// $str содержит некоторую дату
$str = preg_replace("!(\d{2})\.(\d{2})\.(\d{2})!", "\\2/\\1/\\3", $str);
echo $str;
```

С помощью регулярного выражения изменяем формат записи дня, месяца и года, причем, каждый этот элемент отделяем скобками. Во втором параметре функции мы ссылаемся на найденные соответствия в скобках

0 - содержит строку, соответствующую всему шаблону (в нашем примере "10.03.02").

1 - содержит символы, соответствующие только первому элементу, заключенному в скобки (то есть "10").

2 - содержит символы, соответствующие только второму элементу, заключенному в скобки (то есть "03").

На этом этапе мы получаем дату

```
10:50:30 - 10/10/04
```

Теперь переводим это время в стандартное время UNIX

```
$str = str_replace("-", "", $str);
// вырезаем знак "-"
$time = strtotime($str);
echo $time;
```

В результате получим

```
1097380230
```

Теперь можно использовать переменную $time так, как нам будет нужно. Способы перевода времени UNIX в обычное представление мы уже рассматривали.

Рассмотрим теперь функцию split()

```
array split(string pattern, string string, int [limit]);
```

Эта функция строит массив на основе анализа строки string, где разделителем является строка, заданная шаблоном pattern. Если параметр limit установлен, возвращённый массив будет содержать максимум limit элементов с последним элементом, содержащим остаток строки string. Если возникла ошибка, split() возвращает false.

Принцип работы этой функции на понятном языке выглядит следующим образом - разбить строку на компоненты (подстроки), соответствующий правилам, описанным в параметре pattern. Правило может выглядеть, например, так - слова разделены запятой и любым количеством пробелов, причем пробелов может не быть. Это вполне реальное правило. На его основе можно по-

строить массив ключевых слов, содержащихся в строке и разделенных запятыми, а после запятой может идти любое количество пробелов (что совсем не обязательно).

В качестве простого примера можно рассмотреть шаблон вида

```
, *.
```

каждый символ в шаблоне играет большую роль. Обратите внимание на пробел между запятой и звездочкой - это необходимый элемент шаблона. Первый символ означает обязательную запятую после слова, а комбинация " *" (помните, что пробел - элемент шаблона) - любое количество пробелов или их отсутствие. Звездочка является модификатором и говорит, что идущий перед ней символ может встречаться 0 или больше раз.

Рассмотрим теперь, как все это выглядит в реальных скриптах

```
<?
$str = "test,  one,  to,  sree";
$regs = split(", *",$str);
for ($i = 0; $i <= count($regs)-1; $i++)
{
echo $i." : ".$regs[$i].'<br>';
}
?>
```

В результате работы данного кода на экране браузера увидим

```
0 : test
1 : one
2 : to
3 : sree
```

Причем, строка $str может выглядеть и так

```
"test, one, to, sree"
```

или так

```
"test,one,to,sree"
```

или иметь такой вид

```
"test,one,  to,    sree"
```

В любом случае результат работы скрипта будет совершенно одинаковым.

Перейдем к более сложному примеру, который реально может быть использован во многих скриптах. Попытаемся определить версию браузера Internet Explorer с помощью переменной $HTTP_USER_AGENT. На экране увидим

```
Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
```

Сначала попытаемся определить правило выборки номера версии обычным языком. Перед номером версии обязательно идет комбинация символов "MSIE" и пробел, а заканчивается номер версии точкой с запятой.

Теперь запишем это в терминах регулярных выражений

```
MSIE ([^;]+)
```

Как видите, здесь используются скобки различной конфигурации

- Круглые скобки определяют тот элемент, который мы ищем в строке (в нашем случае - шаблон номера версии).
- Квадратные - определяют набор символов, каждый из которых может составлять часть номера версии.
- Знак ^ представляет собой отрицание, т.е. конструкция [^;] в переводе на русский означает "любой символ, кроме точки с запятой".
- Знак + после квадратных скобок говорит, что таких символов (отличных от ;) должно быть минимум 1 (или больше).
- Следующие круглые скобки обозначают границы искомой подстроки. Подстрока, соответствующая шаблону в круглых скобках, сохраняется в специальной переменной.

Таким образом, регулярное выражение MSIE ([^;]+) переводится на русский язык, как "все символы, отличные от точки с

запятой, следующие за набором символов MSIE и пробелом".

Далее нам необходимо получить номер версии. Для этого используем функцию ereg() (или ее регистронезависимый аналог eregi())

```
int ereg(string pattern, string string, array [regs]);
```

В параметр pattern содержится шаблон, в параметре string - исходная строка, предназначенная для анализа, а в параметре regs передается массив найденных подстрок, соответствующих шаблону в круглых скобках. Первый (с индексом 0) элемент массива представляет собой подстроку, соответствующую всему шаблону pattern. Если подстрока, соответствующая шаблону pattern не найдена, то возвращается значение false иначе, количество найденных подстрок.

В данном случае вызов этой функции будет выглядеть следующим образом

```
ereg("MSIE ([^;]+)",$HTTP_USER_AGENT,$regs);
echo "HTTP_USER_AGENT: $HTTP_USER_AGENT <BR>";
echo "VERSION: $regs[1] ";
```

В результате ее работы на экране получим

```
HTTP_USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
VERSION: 5.5
```

Теперь рассмотрим, как с помощью регулярных выражений можно проверить правильность адреса электронной почты, например, maxx@mail.ru. Очевидно, что правдоподобный адрес должен иметь вид

```
слово@слово.слово
```

В терминах шаблонов произвольный символ обозначается знаком "**.**" (мы не будем сейчас учитывать тот факт, что в адресах допустимы не все символы) и в каждом слове должен быть по крайней мере один символ. Таким образом, шаблон слова будет иметь вид

".+"

Вспомним теперь, что "**.**" - это модификатор, и для явного указания точки (в качестве символа) нужно писать

"\."

Тогда образец шаблона будет иметь вид

".+@.+\..+"

Напишем теперь код реального скрипта, который будет выполнять проверку адреса электронной почты

```
<?
$mail= 'aaa@bbb.com';
if (ereg(".+@.+\..+", $mail))
{
echo "Правильный адрес!";
}
else
{
echo "Введите адрес заново!";
}
?>
```

После такой проверки мы можем быть уверены, что E - mail адрес действительно имеет вид "слово@слово.слово".

Регистрация сайта в поисковых системах

Для правильной регистрации сайта в поисковых системах Интернета текст на страницах вашего сайта должен иметь как можно больше ключевых фраз. Ключевые фразы - это слова или словосочетания, которые используются при поиске. В то же время, если их будет слишком много, некоторые поисковые системы могут отнести такой документ к разделу спам (мусор) и откажут в регистрации.

Обязательно проверьте в своих страницах наличие следующих строк

```
<TITLE>Название Вашей страницы</TITLE>
<META NAME="DESCRIPTION" CONTENT="Описание Вашей страницы">
<META NAME="KEYWORDS" CONTENT="Ключевые слова">
<META NAME="REVIZIT-AFTER" CONTENT="10 days">
```

Для нормальной работы поисковых систем необходимо прописывать полный адрес вашей страницы (также желательно указать конкретную страницу, например index.html). Причем, если вы используете фреймы, то лучше сделать вариант страницы без фреймов и именно ее указывать при регистрации. С этой страницы желательно сделать ссылки на большинство документов вашего сайта (или заглавные страницы разделов), т.к. поисковые машины не ограничиваются просмотром указанной страницы, а просматривают также документы, расположенные по ссылкам со страницы.

Для того, чтобы зарегистрировать свой сайт сразу в нескольких поисковых системах, лучше всего воспользоваться услугами служб - авторегистраторов. Приведем краткий список бесплатных служб авторегистрации

http://www.design.ru/free/addurl
http://www.1ps.ru/pr/default.php
http://www.registratura.ru
http://www.addme.com
http://www.doog.com
http://submitter.ru

Аутентификация

Аутентификация в PHP возможна только при запуске PHP, как Apache модуля и недоступна в CGI - версии. В PHP скрипте для Apache модуля можно использовать функцию header() для отправки сообщения "Authentication Required" в клиентский браузер, что вызывает появление в нем окна ввода Username/Password. После того, как пользователь ввел Username и Password, URL содержащий PHP скрипт, будет вновь автоматически вызван, но уже с переменными $PHP_AUTH_USER, $PHP_AUTH_PW и $PHP_AUTH_TYPE, в которых установлены имя пользователя, пароль и тип аутентификации. В настоящее время поддерживается только аутентификация типа "Basic".

Приведем пример фрагмента скрипта, который выполняет аутентификацию клиента для случая PHP версии 4.2 или выше при register_globals = Off

```
<?
if (!isset($_SERVER['PHP_AUTH_USER']))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
{echo " Для доступа на эту страницу требуется идентификация!"; exit;}
}
else
{
echo "Hello {$_SERVER['PHP_AUTH_USER']}";
echo "<br> Вы ввели пароль {$_SERVER['PHP_AUTH_PW']}";
}
?>
```

Этот фрагмента кода для PHP версии ниже 4.2 и register_globals = On будет иметь вид

```
<?
if (!isset($HTTP_SERVER_VARS['PHP_AUTH_USER']))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
```

```
{echo " Для доступа на эту страницу требуется идентификация!"; exit;}
}
else
{
echo "Hello {$HTTP_SERVER_VARS['PHP_AUTH_USER']}";
echo "<br> Вы ввели пароль {$HTTP_SERVER_VARS['PHP_AUTH_PW']}";
}
?>
```

его можно написать и так

```
<?
$user=$HTTP_SERVER_VARS['PHP_AUTH_USER'];
$pass=$HTTP_SERVER_VARS['PHP_AUTH_PW'];

if (!isset($user) or !isset($pass))
//if (!isset($HTTP_SERVER_VARS['PHP_AUTH_USER']))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
{echo "Для доступа на эту страницу требуется идентификация!"; exit;}
}
else
{
echo "Hello {$HTTP_SERVER_VARS['PHP_AUTH_USER']}";
echo "<br> Вы ввели пароль {$HTTP_SERVER_VARS['PHP_AUTH_PW']}";
}
?>
```

Чтобы максимально гарантировать совместимость со всеми клиентами (браузерами), ключевое слово "Basic" должно быть записано с первой "B" в верхнем регистре, управляющая realm строка обязана заключаться в двойные кавычки, и точно один пробел должен предшествовать коду "401" в строке "HTTP/1.0 401".

Вместо простого вывода на экран $PHP_AUTH_USER и $PHP_AUTH_PW, обычно требуется проверить правильность Username и Password путем запроса в базу данных или нахождения записи пользователя в некотором файле.

Имеются способы предотвратить использование кем - либо скрипта, который мог бы раскрыть пароль к странице, аутентифицированной с помощью традиционного, приведенного выше внешнего механизма. В этом случае, для идентификации внешнего пользователя, может использоваться переменная окружения сервера $HTTP_REFERER. Тогда, если внешняя аутентификация включена для данной конкретной страницы пользователя, переменные PHP_AUTH не будут установлены. Для этого в самое начало скрипта нужно добавить следующий блок кода

```
$ref = getenv ("HTTP_REFERER");
if ($ref=='')
{
echo "<center><font size=4 color=red>Вам отказано в доступе на эту страницу!"; exit;
}
```

который запрещает доступ к странице из адресной строки браузера.

Далее можно разрешить доступ только с одной конкретной страницы или с одного хоста с постоянным IP на защищенную паролем страницу некоторого сайта. Для этого нужно запомнить, например, ваш URL при первом посещении закрытой страницы и впоследствии предотвращать доступ с любого другого URL.

Приведем скрипт, дающий доступ только с одного URL с одним паролем

```
<?
// Первый блок
$ref = getenv ("HTTP_REFERER");
if ($ref=='')
{
echo "<center><font size=4 color=red>Запрещен ввод из адресной строки!"; exit;
}

// Второй блок
```

```
$user=$HTTP_SERVER_VARS['PHP_AUTH_USER'];
$pass=$HTTP_SERVER_VARS['PHP_AUTH_PW'];
if (!isset($user) or !isset($pass))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
echo "<center><font size=4 color=red>Для доступа на эту страницу требуется идентификация!"; exit;
}

// Третий блок
$file="$user"."_"."$pass".'.txt';
if(!file_exists($file))
{
$r=' ';
$fp = fopen($file, "w") or die("Не удается открыть файл!");
fwrite($fp, $r) or die("Не удается записать файл!");
fclose($fp);
}

// Четвертый блок
$fp = fopen($file, "rb") or die("Не удается открыть файл!");
$reff = fread ($fp, filesize($file)) or die("Не удается считать файл!");
fclose($fp);

// Пятый блок
if ($reff==' ')
{
$fp = fopen($file, "w") or die("Не удается открыть файл!");
fwrite($fp, $ref) or die("Не удается записать файл!");
fclose($fp);
$reff=$ref;
}

// Шестой блок
if ($ref!==$reff)
{
echo "<center><font size=4 color=red>Вам отказано в доступе на эту страницу!"; exit;
```

```
}

// Седьмой блок
echo "<hr>";
echo "Hello -
{$HTTP_SERVER_VARS['PHP_AUTH_USER']}";
echo "<br> Вы ввели пароль -
{$HTTP_SERVER_VARS['PHP_AUTH_PW']}";
echo "<hr>";
include "Close Page.php";
?>
```

Такой скрипт выполняет следующие действия

1. Первый блок этой программы предотвращает доступ к скрипту из адресной строки - о нем мы уже говорили.
2. Второй блок выполняет аутентификацию пользователя. Если, в этом сеансе работы, она была выполнена ранее - он пропускается.
3. Третий блок выполняется, если заданы имя и пароль пользователя и задает имя файла по этим данным. Если такого файла не существует, создает его с пусто записью. Если такой файл имеется, регистрация прекращается.
4. Четвертый блок читает данные из этого файла.
5. Пятый блок проверяет эти данные, и если они пустые, записывает в этот файл URL страницы, с которой был выполнен доступ к этой защищенной странице. Если содержимое файла имеет конкретное значение - этот блок пропускается.
6. Шестой блок проверяет совпадение текущего URL с записанным в заданном файле, и если они не равны запрещает доступ на закрытую страницу.
7. В случае их равенства, в седьмом блоке, на экран выводятся имя и пароль пользователя и выполняется переход на закрытую страницу Close Page.php.

Скрипт не позволяет заходить на закрытую страницу под одним паролем с разных URL, но может предоставлять доступ к ней с другим паролем, с тем же или другим URL в другом сеансе работы.

Создадим теперь скрипт, который будет выполнять все действия предыдущего скрипта, а так же сохранять имя пользователя

при первом посещении закрытой страницы, т.е. при регистрации - файл reg.php. Повторный вход на закрытую страницу через раздел регистрации будет запрещен. При следующем входе на эту страницу, через другой файл входа (log.php), он будет сравнивать сохраненное имя с вновь введенным и откажет в доступе, если эти имена и пароли не совпадают. Но даже при совпадении данных пользователя, в доступе будет отказано, если вы заходите с другого URL.

Приведем текст файла reg.php, на который мы попадаем по ссылке РЕГИСТРАЦИЯ

```
<?
$ref = getenv ("HTTP_REFERER");
if ($ref=='')
{
echo "<center><font size=4 color=red>Запрещен ввод из адресной строки!"; exit;
}

$user=$HTTP_SERVER_VARS['PHP_AUTH_USER'];
$pass=$HTTP_SERVER_VARS['PHP_AUTH_PW'];
if (!isset($user) or !isset($pass))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
echo "<center><font size=4 color=red>Для доступа на эту страницу требуется идентификация!"; exit;
}

$file="$user"."_"."$pass".'.txt';
if(!file_exists($file))
{
$fp = fopen($file, "w") or die("Не удается открыть файл для записи!");
fwrite($fp, $ref) or die("Не удается перезаписать файл!");
fclose($fp);
}
else
{
echo "<center><font size=4 color=red>Пользователь с таким именем уже существует!"; exit;
```

```
}
echo "<center><font size=4 color=green>Регистрация завершена!<br>Для входа на защищенную страницу используйте ссылку ВХОД";
?>
```

Теперь приведем файл log.php, для ссылки ВХОД

```
<?
$ref =$_SERVER['HTTP_REFERER'];
if ($ref=='')
{
echo "<center><font size=4 color=red>Запрещен ввод из адресной строки!"; exit;
}

$user=$HTTP_SERVER_VARS['PHP_AUTH_USER'];
$pass=$HTTP_SERVER_VARS['PHP_AUTH_PW'];
if (!isset($user) or !isset($pass))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
echo "<center><font size=4 color=red>Для доступа на эту страницу требуется идентификация!"; exit;
}

$file="$user".'_'."$pass".'.txt';
if(file_exists($file))
{
$fp = fopen($file, "rb") or die("Не удается открыть файл для чтения!");
$reff = fread ($fp, filesize($file)) or die("Не удается считать файл!");
fclose($fp);

if($ref!==$reff)
{
echo "<center><font size=4 color=red>Вам отказано в доступе на эту страницу!"; exit;
}
include "Close Page.php";
```

```
}
else
{
echo "<center><font size=4 color=red>Неверные данные пользователя!"; exit;
}
?>
```

Если мы используем PHP версии 4.2 и выше при register_globals = Off, все переменные $HTTP_SERVER_VARS и getenv() нужно заменить на новую переменную окружения $_SERVER.

Рассмотрим теперь вариант скрипта, который, выполняя аутентификацию, создает пользовательскую директорию с именем пользователя, а внутри нее создается файл index.php, в котором пишется информация о пользователе и его URL. Будем рассматривать вариант PHP интерпретатора версии 4.2 или выше с register_globals = Off. Такой скрипт так же состоит из двух файлов, первый из которых reg.php

```
<?
$ref = $_SERVER['HTTP_REFERER'];
if ($ref=='')
{
echo "<center><font size=4 color=red>Запрещен ввод из адресной строки!"; exit;
}

$user=$_SERVER['PHP_AUTH_USER'];
$pass=$_SERVER['PHP_AUTH_PW'];
if (!isset($user) or !isset($pass))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
echo "<center><font size=4 color=red>Для доступа на эту страницу требуется идентификация!"; exit;
}

$dir="$user";
$file="./$dir/index.php";
if (!is_dir($dir))
```

```
{mkdir($dir,0) or die("Невозможно создать директорию!");}
else
{echo "<center><font size=4 color=red>Пользователь с таким именем уже существует!"; exit;}

if(!file_exists($file))
{
$re=$user.' '.$pass.' '.$ref;
$fp = fopen($file, "w") or die("Не удается открыть файл для записи!");
fwrite($fp, $re) or die("Не удается повторно записать файл!");
fclose($fp);
}
else
{echo "<center><font size=4 color=red>Пользователь с таким именем уже существует!"; exit;}

echo "<center><font size=4 color=green>Регистрация завершена!<br>Для входа на закрытую страницу используйте ссылку ВХОД";
?>
```

Второй файл log.php

```
<?
$ref =$_SERVER['HTTP_REFERER'];
if ($ref=='')
{
echo "<center><font size=4 color=red>Запрещен ввод из адресной строки!"; exit;
}

$user=$_SERVER['PHP_AUTH_USER'];
$pass=$_SERVER['PHP_AUTH_PW'];
if (!isset($user) or !isset($pass))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
echo "<center><font size=4 color=red>Для доступа на эту
```

```
страницу требуется идентификация!"; exit;
    }

    $dir="$user";
    if (!is_dir($dir))
    {echo "<center><font size=4 color=red>Пользователь с таким именем не существует!"; exit;}

    $file="./$dir/index.php";
    if(file_exists($file))
    {
    $fp = fopen($file, "rb") or die("Не удается открыть файл для чтения!");
    $reff = fread ($fp, filesize($file)) or die("Не удается считать файл!");
    $p=explode(" ", $reff);
    fclose($fp);

    if($pass!==$p[1])
    {echo "<center><font size=4 color=red>Неверный пароль!"; exit;}

    if($ref!==$p[2])
    {echo "<center><font size=4 color=red>Вам отказано в доступе на эту страницу!"; exit;}

    echo "<font size=4 color=#003366>Name = <font size=4 color=green>$p[0]<br><font size=4 color=#003366>Pass = <font size=4 color=green>$p[1]<br><font size=4 color=#003366>URL = <font size=4 color=green>$p[2]<br>";
    }
    else
    {echo "<center><font size=4 color=red>Неверные данные пользователя!"; exit;}
    ?>
```

Этот файл, при верной аутентификации выведет на экран содержимое файла index.php, находящегося в директории с именем пользователя.

Рассмотрим другой пример HTTP аутентификации, который позволяет изменять имя и пароль. Вначале определим скрипт для

перерегистрации, который находится в файле reauth.php

```
<?
$ref =$_SERVER['HTTP_REFERER'];
if ($ref=='')
{
echo "<center><font size=4 color=red>Запрещен ввод из адресной строки!"; exit;
}

$user=$_SERVER['PHP_AUTH_USER'];
$pass=$_SERVER['PHP_AUTH_PW'];
if (!isset($user) or !isset($pass))
{
header("WWW-Authenticate: Basic realm=\"My Realm\"");
header("HTTP/1.0 401 Unauthorized");
echo "<center><font size=4 color=red>Для доступа на эту страницу требуется идентификация!"; exit;
}

$dir="$user";
if (!is_dir($dir))
{echo "<center><font size=4 color=red>Пользователь с таким именем не существует!"; exit;}

$file="./$dir/index.php";
if(file_exists($file))
{

$fp = fopen($file, "rb") or die("Не удается открыть файл для чтения!");
$reff = fread ($fp, filesize($file)) or die("Не удается считать файл!");
$p=explode(" ", $reff);
fclose($fp);
echo "$user<br>$pass<br>$ref";

if($pass!==$p[1])
{echo "<center><font size=4 color=red>Неверный пароль!"; exit;}
```

```
if($ref!==$p[2])
{echo "<center><font size=4 color=red>Вам отказано в доступе на эту страницу!"; exit;}

?>
<center><font size=4 color=#003366>Введите новые имя и пароль!
<form action="form.php" METHOD="POST">
<input type="text" name="use" value="">
<input type="text" name="pas" value=""><br>
<input type="submit" name ="Re" value="OK"><br>
<input type="hidden" name="ref" value="<?=$ref?>"><br>
<input type="hidden" name="user" value="<?=$user?>"><br>
<input type="hidden" name="pass" value="<?=$pass?>">
</form>
<?

}
else
{echo "<center><font size=4 color=red>Неверные данные пользователя!"; exit;}
```

Этот скрипт проверяет данные пользователя, и если он существует, выводит на экран форму для ввода новых имени и пароля. После их ввода и нажатия OK управление передается другому скрипту, который обрабатывает новые данные, создает новую папку пользователя и удаляет старую. Приведем теперь код этого скрипта, который расположен в файле form.php

```
<?
$user=$_POST['user'];
$pass=$_POST['pass'];
$use=$_POST['use'];
$pas=$_POST['pas'];

$ref=$_POST['ref'];
$Re=$_POST['Re'];

if (!IsSet($Re))
include "base.php";
```

```
    echo "US = $user<br>PS = $pass<br>REF = $ref<br>Re = $Re<br>";
    echo "<br>US = $use<br>PS = $pas<br>";

    $dir="$use";
    $file="./$dir/index.php";

    if (!is_dir($dir))
    {mkdir($dir,0) or die("Невозможно создать пользователя!");}
    else
    {echo "<center><font size=4 color=green>Такой пользователь уже существует!"; exit;}

    $re=$use.' '.$pas.' '.$ref;
    $fp = fopen($file, "w") or die("Не удается открыть файл для записи!");
    fwrite($fp, $re) or die("Не удается записать файл!");
    fclose($fp);

    $dir="$user";
    $file="./$dir/index.php";
    unlink($file) or die("Не удается удалить старый файл!");
    rmdir($dir) or die("Не удается удалить директорию!");

    echo "<center><font size=4 color=green>Перерегистрация завершена!<br>Для входа на закрытую страницу используйте ссылку ВХОД";
    ?>
```

Здесь рассмотрен вариант скрипта для register_globals = Off и новых версий PHP. Для режима On и старых версий PHP операторы типа $user=$_POST['user'] использовать не нужно, поскольку все переменные из формы в скрипт передаются автоматически.

Система голосования

Установка на сайте собственного опроса или системы голосования позволяет не только получить полезную информацию о том, что думают ваши посетители, но и придать Web - сайту некоторую интерактивность.

Лучше всего создать свой собственный скрипт, который позволяет проводить голосование на сайте. Конечно, это потребует от вас некоторых усилий, но зато вполне окупиться за счет хорошей конфигурируемости системы, тонкости настройки и полностью своего интерфейса.

Допустим, что мы на сайте хотим объявить следующее голосование

Как Вам нравится наш новый дизайн?

5 - Отлично.
4 - Хорошо
3 - Так себе.
2 - Плохо.
1 - Очень плохо.

Для начала нам нужно сделать этот опрос в виде HTML, определив имена для переменных

```
<html>
<body>
<form action="vote.php" method="get" target="_blank">
<b>Как Вам нравится наш новый дизайн?</b>
<p>
<input type="Radio" name="vote" value="1" checked>
Отлично.<br>
<input type="Radio" name=vote value=2>
Хорошо.<br>
<input type="Radio" name=vote value=3>
Так себе.<br>
<input type="Radio" name=vote value=4>
Плохо.<br>
<input type="Radio" name=vote value=5>
Очень плохо.
<p>
```

```
<input type="Submit" value="Голосовать!"><p>

<a href="vote.php" target="_blank">
<b>Текущие результаты</a>
</form>
</body>
</html>
```

В этом файле формы имеется ссылка на PHP скрипт "Текущие результаты", без номера ответа. Это сделано для того чтобы скрипт, при отсутствии переменной vote, просто выводит на экран результаты, без сопутствующей благодарности "Спасибо за Ваше мнение!" и каких - либо действий по записи результатов голосования.

Нашему скрипту потребуется собственная база данных с текстами вопросов и ответом к ним. Конкретное голосование - это один вопрос и набор ответов к нему, а также количество мнений на каждый ответ.

Отсюда получается, что для создания голосования достаточно создать новый текстовой файл, например, date.dat, сам скрипт void.php и html - форму - form.htm, содержащую тексты вопросов. Рассмотрим теперь формат текстового файла - его нужно создать предварительно, записав в него следующие данные

```
Как Вам нравится наш новый дизайн?
0~Отлично
0~ Хорошо
0~Так себе
0~Плохо
0~Очень плохо
```

Здесь 1 - я строка - это сам вопрос, остальные строки представляют собой пару - "количество мнений ~ ответ". Теперь все, что нам осталось, это обрабатывать полученные от формы результаты. Заметим, что в данном случае мы будем использовать установку register_globals = On и PHP версии 4.0.4.

Напишем код самого PHP скрипта

```
<?
// Файл с системой голосования
$fil="date".'.dat';
```

```
$data = file("$fil");
// Выводим благодарности, если это не просто просмотр результатов
if ($vote)
echo "<center><b><font color=red>Спасибо за Ваше мнение!</font></b><p>";

// Выводим заголовок голосования - 1я строка файла
echo "<center><b>$data[0]</b><p>";
// Выводим на экран список ответов и результатов, т.е. остальные строки
for ($i=1;$i<count($data);$i++)
{
$votes = split("~", $data[$i]);
// Выводим ответ ~ число ответов
if ($i==$vote)
{$votes[0]=$votes[0]+1;}
echo "$votes[1]: <b>$votes[0]</b><br>";
}

// Если это не просмотр результатов, а голосование, производим запись голосов
if ($vote)
{
$f = fopen($fil,"w");
fputs($f, "$data[0]");

for ($i=1;$i<count($data);$i++)
{
$votes = split("~", $data[$i]);

if ($i==$vote)
{$votes[0]=$votes[0]+1;}
fputs($f,"$votes[0]~$votes[1]") or die("Невозможно записать файл!");
}
fclose($f);
}
?>
```

Как видите, скрипт достаточно прост, не выполняет каких -

либо специфический действий и не потребляет много ресурсов. Поле каждого голосования, в файле date.dat, будут меняться общие результаты по всем голосованиям.

Дубовиченко
Сергей Борисович

Web программирование

Часть 1. Основы языка PHP

Учебник по информатике для ВУЗов

Подписано к печати 01.07.2004. Формат 60x84 1/16. Бумага офсетная.
Печать офсетная. Усл. изд. листов 18,0. Тираж 500. Цена договорная.

Издательство КАУ, Алматы, Казахстан,
Отпечатано в типографии Каз.ГАСА, Алматы, Казахстан

Дубовиченко Сергей Борисович

Член - корреспондент Казахстанской Международной Академии Информатизации

член Нью - Йоркской Академии Наук

член Европейского Физического Общества

Лауреат премии ЛКСМ Казахстана

Лауреат международного гранта Сороса

кандидат физико - математических наук

доцент Казахско - Американского Университета