ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

**Федеральное государственное
образовательное бюджетное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
им. проф. М. А. БОНЧ-БРУЕВИЧА»**

Ф. В. Филиппов
А. Н. Губин

HTTP + PHP
В ПРИМЕРАХ И ЗАДАЧАХ

УЧЕБНОЕ ПОСОБИЕ

СПб ГУТ)))

**САНКТ-ПЕТЕРБУРГ
2015**

УДК 004.42(075.8)
ББК 32.973-018я73
Ф53

Рецензенты:
кандидат технических наук, доцент кафедры
процессов управления и информационных систем
Национального минерально-сырьевого университета «Горный»
С. А. Матюхин,
кандидат технических наук, доцент кафедры
информационных управляющих систем СПбГУТ
В. А. Рогачев

Утверждено редакционно-издательским советом СПбГУТ
в качестве учебного пособия

Филиппов, Ф. В.
Ф53 HTTP + PHP в примерах и задачах : учебное пособие / Ф. В. Филиппов, А. Н. Губин ; СПбГУТ. – СПб., 2015. – 68 с.

Рассматриваются практические аспекты языка PHP, используемые при разработке информационных управляющих систем на базе методов протокола HTTP. Пособие может быть использовано при изучении дисциплин «Технология программирования», «Технологии обработки информации» и «Технологии проектирования программного обеспечения информационных систем».

Предназначено для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии».

УДК 004.42(075.8)
ББК 32.973-018я73

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

В середине 90-х гг. очень популярной стала технология WWW (World Wide Web) – «всемирная паутина». Это набор протоколов и программ для интернета, представляющих информацию в гипертекстовом формате. Знаменитый браузер Mosaic, созданный в Национальном центре по применению суперЭВМ (National Center for Supercomputer Applications, NCSA), был первым графическим Web-браузером и способствовал популяризации WWW. Web разработана в 1989 г. в Европейской лаборатории физики частиц (European Laboratory for Particle Physics, CERN) Тимоти Бернерсом-Ли (Timothy Berners-Lee). В настоящее время всеми стандартами, имеющими отношение к Web, ведает Консорциум World Wide Web (W3C).

Для упаковки и передачи данных в Web применяются протоколы MIME (Multipurpose Internet Mail Extensions), TCP/IP (Transmission Control Protocol/Internet Protocol), FTP (File Transfer Protocol) и Telnet. Специально для Web разработаны указатели URL (Uniform Resource Locator), протокол HTTP (Hyper Text Transfer Protocol), язык HTML (Hyper Text Markup Language) и интерфейс CGI (Common Gateway Interface).

Цель настоящего пособия – познакомить с основными методами протокола HTTP и их практическим использованием для обмена информацией во всемирной паутине. Все задания на практические и лабораторные работы выполняются на языке PHP, который фактически является одним из основных средств разработки скриптов серверных приложений. В качестве среды выполнения заданий выбрана кроссплатформенная сборка локального веб-сервера XAMPP, которая, так же как и Денвер, является свободно распространяемым програмным продуктом, но, по мнению авторов, по отношению к последнему обладает некоторыми преимуществами.

1. ПРОТОКОЛ HTTP

HTTP – это протокол прикладного уровня, который размещается поверх TCP и в основном известен как транспортный канал для World Wide Web и локальных сетей. Однако это классический протокол, который используется помимо гипертекста для многих других задач, например в серверах доменных имен и системах распределенного управления объектами посредством своих методов запросов, кодов ошибок и заголовков. Сообщение HTTP представляется в MIME-подобном формате; оно содержит метаданные о сообщении (например, тип его содержания и длину) и информацию о запросе и ответе, например метод, используемый для отправки запроса.

Существуют два основных компонента, от которых зависит Web: сетевые протоколы TCP/IP и HTTP. Почти все события в Web происходят через HTTP, и этот протокол преимущественно используется для обмена документами (такими как Web-страницы) в World Wide Web.

HTTP – это протокол приложения клиент-сервер, через который взаимодействуют две системы, обычно использующие соединение TCP/IP. HTTP-сервер – это программа, слушающая на порте машины входящие HTTP-запросы.

HTTP-клиент через сокет открывает соединение с сервером, отправляет сообщение с запросом на конкретный документ и ждет ответа от сервера. Сервер отправляет сообщение, содержащее код нормального или аварийного завершения, заголовки с информацией об ответе и (если запрос обработан успешно) требуемый документ. Общий формат HTTP-сообщения одинаков для запросов и ответов:

```
начальная-строка
заголовок-сообщения (или заголовки)
[тело-сообщения]
```

В сообщение может входить любое число заголовков, и каждый из них располагается на отдельной строке (т. е. каждому заголовку предшествуют символы возврата каретки и перевода строки). Тело сообщения присутствует необязательно, но если оно имеется, то отделяется от заголовков двумя последовательностями CRLF.

В протоколе HTTP используются постоянные и непостоянные соединения. Непостоянные соединения применяются по умолчанию в версии 1.0 HTTP, в то время как постоянные соединения – в версии HTTP 1.1. Соединение называют непостоянным (non-persistent connection), если любое TCP-соединение закрывается сразу же, как только сервер отправляет клиенту требуемый объект. Это означает, что соединение используется только для одного запроса и одного ответа и не сохраняется для других запросов и ответов.

В случае постоянных соединений сервер, отправив ответ, оставляет соединение открытым, и, таким образом, следующие запросы и ответы между теми же клиентом и сервером могут отправляться через это же самое соединение. Такое соединение сервер закрывает лишь после того, как оно не используется в течение некоторого интервала времени.

HTTP-заголовки

HTTP-сообщение состоит из начальной строки, за которой следуют набор заголовков, пустая строка и некоторые данные. Начальная строка задает действие, требуемое от сервера, тип возвращаемых данных или код состояния.

HTTP-заголовки можно подразделить на три крупные категории: заголовки, посылаемые в запросе, заголовки, посылаемые в ответе, и те, которые можно включать как в запросы, так и в ответы. Заголовки запросов указывают возможности клиента, например, типы документов, которые может обработать клиент, в то время как заголовки ответов предоставляют информацию о возвращенном документе.

Заголовки запросов

К числу наиболее важных HTTP-заголовков, которые можно включать в запросы, но нельзя включать в ответы, относятся:

- заголовок Accept

Это список MIME-типов, принимаемых клиентом, в формате тип/подтип. Элементы списка должны разделяться запятыми:

```
Accept: text/html, image/gif, */*
```

Элемент */* указывает, что все типы будут приняты и обработаны клиентом. Если тип запрошенного файла не может быть обработан клиентом, возвращается ошибка HTTP 406 «Not acceptable» (недопустимо);

- заголовок From

Указывает адрес электронной почты в Интернете учетной записи пользователя, под которой работает клиент, направивший запрос:

```
From: sender@gmail.ru;
```

- заголовок Referer

Позволяет клиенту указать адрес (URI) ресурса, из которого получен запрашиваемый URI. Этот заголовок дает возможность серверу сгенерировать список обратных ссылок на ресурсы для будущего анализа, регистра-

ции, оптимизированного кэширования и т. д. Он также позволяет прослеживать в целях последующего исправления устаревшие или введенные с ошибками ссылки:

Referer: http://www.professorweb.ru

• Заголовок User-Agent

Представляет собой строку, идентифицирующую приложение-клиент (обычно браузер) и платформу, на которой оно выполняется. Общий формат имеет вид: программа/версия библиотека/версий, – но это не неизменный формат:

User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.17

Эта информация может использоваться в статистических целях, для отслеживания нарушений протокола и для автоматического распознавания клиента. Она позволяет приспособить ответ так, чтобы не нарушить ограниченные возможности конкретного клиента, например неспособность поддерживать HTML-таблицы.

Заголовки ответов

В ответы могут включаться следующие заголовки:

• Заголовок Content-Type

Используется для указания типа данных, отправляемых получателю или, в случае метода HEAD, типа данных, которые были бы отправлены в ответ на запрос GET:

Content-Type: text/html

• Заголовок Expires

Представляет собой момент времени, после которого информация в документе становится недостоверной. Клиенты, использующие кэширование, в частности прокси-серверы, не должны хранить в кэше эту копию ресурса после заданного времени, если только состояние копии не было обновлено более поздним обращением к исходному серверу:

Expires: Fri, 19 Aug 2012 16:00:00 GMT

• Заголовок Location

Определяет точное расположение другого ресурса, к которому может быть перенаправлен клиент. Если это значение представляет собой полный URL, сервер возвращает клиенту «redirect» для непосредственного извлечения указанного объекта:

Location: http://www.samplesite.com

Если ссылка на другой файл относится к серверу, должен указываться частичный URL.

• Заголовок Server

Содержит информацию о программном обеспечении, используемом исходным сервером для обработки запроса:

```
Server: Microsoft-IIS/7.0
```

Общие заголовки

Несколько заголовков могут включаться как в запрос, так и в ответ, например:

• Заголовок Date

Используется для установки даты и времени создания сообщения:

```
Date: Tue, 16 Aug 2012 18:12:31 GMT
```

• Заголовок Connection

В HTTP/1.0 мы могли использовать в запросе заголовок Connection, указывая, что хотим сохранить соединение после отправки ответа. Теперь такое поведение принято по умолчанию, и в HTTP/1.1 можно использовать заголовок Connection, чтобы указать, что постоянное соединение не нужно:

```
Connection: close
```

HTTP-запросы

Каждый клиент посылает запрос, и сервер на него отвечает. Все запросы и ответы состоят из трех частей, а именно: строки запроса или ответа, секции заголовков и тела сущности (любого содержания, отправляемого вместе с сообщением, например, это страницы HTML для отображения в браузере или данные формы, пересылаемые на сервер).

Клиент связывается с сервером в назначенном номере порта (по умолчанию равном 80) и запрашивает у сервера документ, задавая HTTP-команду, называемую методом, за которой следует адрес документа и номер версии HTTP. Клиент также отправляет серверу необязательную информацию в заголовках, чтобы сообщить серверу о своей конфигурации и приемлемых для него форматах документов. Информация заголовка дается в одной строке вместе с именем и значением заголовка. После заголовков клиент посылает пустую строку. Затем клиент отправляет дополнительные данные. Это могут быть данные формы, отправляемые на сервер методом POST, или файл, копируемый на сервер методом PUT.

Запросы клиентов подразделяются на три секции. Первая строка сообщения всегда должна содержать HTTP-команду, называемую методом, за которой следуют URI, идентифицирующий файл или ресурс, запрашиваемый клиентом, и номер версии HTTP:

```
GET /default.aspx HTTP/1.1
```

Теперь исследуем каждую из этих секций. Метод – это HTTP-команда, начинающая первую строку запроса клиента. Метод информирует сервер о цели запроса клиента. Для HTTP определены семь методов: GET, HEAD, POST, OPTIONS, PUT, DELETE и TRACE, но HTTP-серверы могут также реализовать методы расширения, не определенные протоколом HTTP. Заметим, что названия методов зависят от регистра клавиатуры, поэтому, например, слово get не будет распознано как допустимый метод.

Метод GET используется для запроса информации, расположение которой на сервере определяется заданным URI. Этот метод широко применяется браузерами, чтобы извлекать документы для просмотра. Результат запроса GET генерируется разными способами. Это может быть файл, доступный с сервера, вывод программы, вывод, полученный на устройстве, и т. д.

Когда клиент в своем запросе использует метод GET, сервер отправляет ответ, содержащий строку состояния, заголовки и метаданные. Если сервер не может обработать запрос из-за ошибки или отсутствия авторизации, он отправляет объяснение в текстовом виде, помещая его в ответе в секцию данных.

Секция тела о сути запроса GET всегда остается пустой. Запрошенный клиентом ресурс (файл или программа) идентифицируется по его полному пути на сервере. Любая дополнительная информация, например значения из формы, которую клиенту нужно отправить серверу, присоединяется вслед за URI как строка запроса:

```
GET /default.aspx?name=Alex HTTP/1.1
```

Метод HEAD функционально аналогичен методу GET, не считая того, что сервер ничего не помещает в секцию данных ответа. Методом HEAD запрашивается только заголовочная информация по файлу или ресурсу. Для запроса HEAD HTTP-сервер должен отправить в заголовках ту же информацию, которую он бы отправил в ответ на запрос GET. Данный метод используется, если клиенту нужна информация о документе, но не нужно получать сам документ.

Метод POST позволяет отправить данные серверу в клиентском запросе. Эти данные посылаются программе обработки данных, к которой у сервера есть доступ. Метод POST может использоваться для многих приложений, например для обеспечения входных данных сетевых служб,

программ интерфейса командной строки и т. д. Данные отправляются на сервер в секции тела клиентского запроса. Обработав запрос POST и заголовки, сервер передает это тело программе, указанной в URI.

В методе OPTIONS запрашивается информация о поддержке HTTP на Web-сервере. Метод OPTIONS может применяться с URL, чтобы извлечь информацию о конкретном документе или (с групповым символом *), чтобы получить информацию о возможностях сервера в целом. Информация возвращается в заголовках ответа.

HTTP-ответы

Препроцессная обработка на стороне сервера подразумевает вызов программы-интерпретатора, которая обрабатывает запрашиваемый файл скрипта, исполняет его команды. Результат работы интерпретатора передается веб-серверу, который, в свою очередь, возвращает их клиенту (рис. 1).

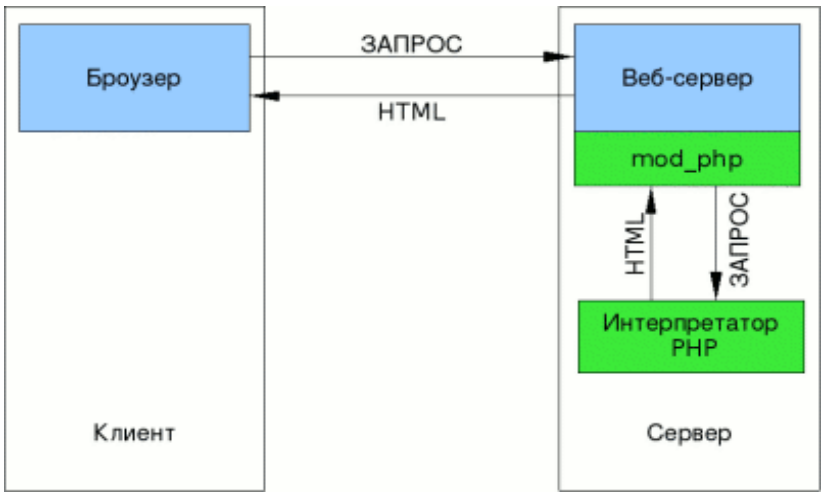

Рис. 1. Препроцессинг HTML на сервере

Ответ сервера на запрос клиента также подразделяется на три части. Первая строка – это строка ответа сервера, содержащая номер версии HTTP, код состояния, указывающий состояние запроса, и краткую фразу, описывающую это состояние. Далее следует информация заголовков, за ней – пустая строка и тело сущности (которое может быть пустым, например в ответах на запросы HEAD и OPTIONS).

В качестве версии HTTP указывается та версия, которую сервер использует в ответе. Код состояния представляет собой трехбайтовое число, определяющее результат обработки сервером запроса клиента. Описание, следующее за кодом состояния, просто дает удобное для восприятия пользователем значение кода состояния. Хотя существует несколько определенных кодов состояния, сервер вправе устанавливать дополнительные коды. Некоторые наиболее распространенные коды приведены в табл. 1.

Таблица 1

Коды состояния запроса HTTP

Код	Состояние запроса
100	Продолжить (Continue)
101	Переключение протоколов (switching protocols)
200	OK – запрос получен и обработан
201	Создан (created)
202	Принят
203	Неавторитеная информация (non-authoritative information)
204	Файл пуст (no content)
205	Сброс содержимого (reset content)
206	Частичное содержимое (partial content)
300	Многократный выбор (multiple choices)
301	Ресурс перемещен постоянно (moved permanently)
302	Ресурс перемещен временно (moved temporarily)
303	Смотри другой (see other)
304	Неизмененный (not modified)
305	Использовать прокси-сервер (use proxy)
400	Неправильный формат запроса (bad request)
401	Запрос неавторизирован (unauthorized)
402	Payment Required
403	Нет доступа (forbidden)
404	Не найден
405	Метод не разрешен (method not allowed)
406	Неприемлемый (not acceptable)
407	Требуется аутентификация на прокси-сервере (proxy authentication required)
408	Превышение тайм-аута запроса (request time-out)
411	Требуется длина (при использовании метода POST)
412	Не выполнено предыдущее условие (precondition failed)
413	Объект запроса слишком велик (request entity too large)
414	Запрашиваемый URL слишком велик (request URL too large)
415	Неподдерживаемый тип информации (unsupported media type)
500	Ошибка сервера (server error)
501	Не реализован (not implemented)
502	Неправильный шлюз (Bad Gateway)
503	Нехватка ресурсов (out of resources)
504	Превышен тайм-аут шлюза
505	Неподдерживаемая весрия HTTP (HTTP version not supported)

После строки состояния сервер отправляет клиенту в заголовках информацию о себе и запрошенном документе. Заголовки завершаются пустой строкой (т. е. двумя идущими подряд последовательностями CRLF).

Если клиент запрашивал данные и запрос обработан успешно, эти данные будут отправлены в теле сущности после заголовков ответа. Они могут представлять собой копию запрошенного файла или содержание, сгенерированное динамически, например страницу ASP.NET или сценарий на стороне сервера. Если запрос клиента не выполнен, могут быть предоставлены дополнительные данные, объясняющие, почему сервер не смог выполнить этот запрос.

В HTTP 1.0 сервер, завершив отправку запрошенных данных, отсоединяется от клиента, и транзакция на этом заканчивается, если только не был отправлен заголовок Connection: Keep-Alive. Однако в HTTP 1.1 сервер должен поддерживать соединение, позволяя клиенту делать дополнительные запросы, даже если заголовок Connection не был отправлен. Если не нужно такое поведение, следует отправить заголовок Connection: close, который указывает, что после отправки ответа соединение должно быть закрыто.

2. ЛОКАЛЬНЫЙ ВЕБ-СЕРВЕР XAMPP

XAMPP (**X** +**A**pache + **M**ySQL + **P**HP + **P**erl) – это кроссплатформенная сборка локального веб-сервера. Ее можно установить на операционной системе **X**: Linux, Windows, Mac OS, Solaris. Локальный веб-сервер – это набор программ, которые позволяют разрабатывать сайты на локальном компьютере без подключения к Интернету. Он аналогичен серверам, которые расположены у хостеров. Если при создании сайта используются только html-страницы с дизайном CSS, то локальный сервер не нужен. Но если сайт динамичный, т. е. создается с использованием PHP, Perl, MySQL, то для его отладки и тестирования понадобится локальный веб-сервер.

XAMPP – свободно распространяемая программа. Весь процесс установки не требует никакого вмешательства со стороны пользователя, кроме нажатия кнопки «Next». После установки XAMPP открывается панель управления сервером (рис. 2).

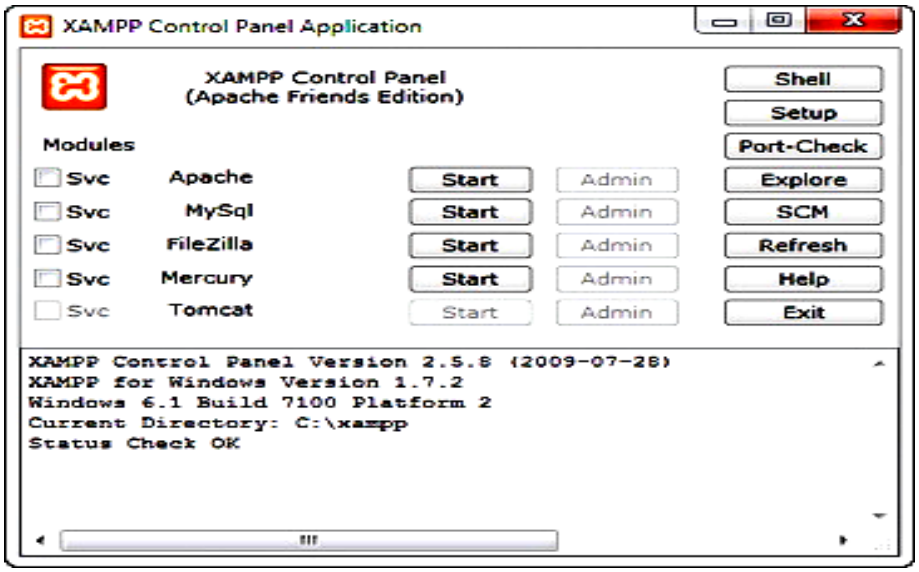

Рис. 2. Панель управления XAMPP

Элементы панели управления

- **Checkboxes (Svc)** – устанавливает модуль в качестве службы Windows (если вы хотите запускать какой-либо модуль (Apache, MySQL и т. д.) как службу, то необходимо отметить соответствующие флажки).
- **Admin** напротив модуля **Apache** – запускает администрирование модулем и открывает окно приветствия XAMPP. Там необходимо выбрать язык и попасть в веб-интерфейс XAMPP, где можно посмотреть демонстрационные примеры в разделе Demos, узнать информацию о модулях и т. д.
- **Admin** напротив модуля **MySQL** – запускает phpMyAdmin.
- **Shell** – запускает командную строку для работы с сервером.

- **Setup** – открывает командную строку для настройки XAMPP.
- **Port-Check** – проверяет порты.
- **Explore** – открывает папку, в которую установлен XAMPP, в нашем случае C:\xampp.
- **SCM** – открывает окно управления службами Windows.
- **Refresh** – обновляет состояния модулей.

Запускается XAMPP с помощью ярлыка на рабочем столе или в меню пуск. Если вы при установке отказались от создания ярлыков, то можете запустить XAMPP с помощью файла xampp-control.exe в директории C:\xampp, там же можно увидеть все остальные ярлыки, которыми можно запустить отдельные модули, деинсталировать XAMPP и т. п.

Запускаем модули Apache и MySQL (кнопки Start), набираем в браузере localhost (или жмем Admin напротив модуля Apache) и попадаем в веб-интерфейс XAMPP, здесь вы можете ознакомиться с документацией, примерами в разделе Demos и т. д. В разделе Security находятся настройки безопасности. Перейдя по указанной ссылке http://localhost/security/xamppsecurity.php, можно задать пароль для администратора баз данных MySQL и установить пароль на директорию XAMPP для ограничения доступа из локальной сети.

Для того чтобы создать новый сайт на локальном веб-сервере, необходимо создать папку с названием сайта в папке C:\xampp\htdocs\, например site, и затем создать там тестовый php-скрипт с кодом:

```
<?php
echo 'Мой сайт – site!';
?>
```

Сохраним его как index.php. Переходим в браузере по адресу http://localhost/site/ и, если все сделано правильно, видим «Мой сайт – site!», тем самым мы протестировали работоспособность сервера.

Настройка виртуальных хостов

Для того чтобы просматривать в браузере созданный сайт, необходимо настроить виртуальный хост. Для этого необходимо отредактировать файл C:\xampp\apache\conf\extra\httpd-vhosts.conf. Открываем файл в текстовом редакторе. В этом файле имеется некоторая информация, все строки которой начинаются с #. В конец файла добавляем следующий код:

```
NameVirtualHost 127.0.0.1

<VirtualHost 127.0.0.1>
  ServerName localhost
  ServerAdmin admin@localhost
</VirtualHost>
```

Здесь указываем имя виртуального хоста и описываем локальный хост. Теперь надо добавить данные, чтобы заработал сайт. Пусть сайт будет называться site. Конечно, сайт должен называться, например, www.site.ru, но нет необходимости указывать полное имя сайта, так как это название будет использоваться только на локальном компьютере. Под только что добавленным кодом добавляем еще код:

```
<VirtualHost 127.0.0.1>
  ServerName site
  ServerAlias www.site
  ServerAdmin admin@site.ru
  DocumentRoot "C:/xampp/htdocs/site/www/"
  ErrorLog "C:/xampp/htdocs/site/logs/error.log"
  CustomLog "C:/xampp/htdocs/site/access.log" combined
<Directory "C:/xampp/htdocs/site/www/">
  AllowOverride All
</Directory>
</VirtualHost>,
```

где

ServerName site – название нашего сайта (которое можно писать без .ru);
ServerAlias www.site – альтернативное название сайта;
ServerAdmin admin@site.ru – почта администратора ресурса;
DocumentRoot "C:/xampp/htdocs/site/www/" – папка с файлами сайта;
ErrorLog "C:/xampp/htdocs/site/logs/error.log" – журнал ошибок;
CustomLog "C:/xampp/htdocs/site/access.log" combined – журнал посещений.

Когда потребуется создать еще один сайт, будет достаточно скопировать и добавить последний код, заменив в нем название сайта на новое.

Редактирование файла hosts

Осталось добавить созданный сайт в базу данных доменных имен операционной системы Windows. Для этого надо отредактировать файл hosts, расположенный в папке C:/Windows/Sistem32/drivers/etc/. В нем будет некоторая информация, также закомментированная решеткой #. Ниже добавляем следующую конструкцию:

```
127.0.0.1 site
```

При создании еще одного сайта, скажем site2, нужно добавить в этот файл:

```
127.0.0.1 site2
```

Настройка почты

При использовании в качестве локального веб-сервера Denwer все письма с локального сайта отправлялись в папку sendmail, это достаточно удобно, но в XAMPP такой функции не предусмотрено, в XAMPP есть собственный почтовый сервер, но он потребует немало времени для его настройки

С помощью описанной ниже инструкции можно создать такую же «заглушку», как и в Denwer.

1. В папке sendmail (C:\xampp\sendmail) создадим файл sendmail.php с кодом:

```
<?php
define('DIR','c:/xampp/tmp/sendmail/');
$stream = '';
$fp = fopen('php://stdin','r');
while($t=fread($fp,2048))
{
if( $t===chr(0) )
break;
$stream .= $t;
}
fclose($fp);
$fp = fopen(mkname(),'w');
fwrite($fp,$stream);
fclose($fp);

function mkname($i=0)
{
$fn = DIR.date('Y-m-d_H-i-s_').$i.'.eml';
if ( file_exists($fn) )
return mkname(++$i);
else return $fn;
}
?>
```

Предполагается получение писем в кодировке UTF-8. Если требуется получать письма в кодировке CP1251, то необходимо заменить строку

```
fwrite($fp,$stream);
```

на строку

```
fwrite($fp,iconv("UTF-8","CP1251",$stream));
```

2. В файле php.ini (C:\xampp\php) заменяем строку

```
sendmail_path = "C:\xampp\sendmail\sendmail.exe -t"
```

на строку

```
sendmail_path = C:\xampp\php\php.exe c:\xampp\sendmail\sendmail.php
```

Строка должна быть раскомментирована (убрать символ точки с запятой «;» в начале строки).

3. Создаем папку sendmail в папке tmp (C:\xampp\tmp\sendmail).

4. Перезапускаем сервер Apache, и теперь все отправленные письма будут в папке C:\xampp\tmp\sendmail.

В заключение этого раздела проанализируем информацию о настройках веб-сервера и PHP с помощью встроенной функции phpinfo(), запустив следующий скрипт:

```
<?php
phpinfo();
?>
```

В результате будет предоставлена полная информация о характеристиках установленной версии языка и веб-сервера (рис. 3).

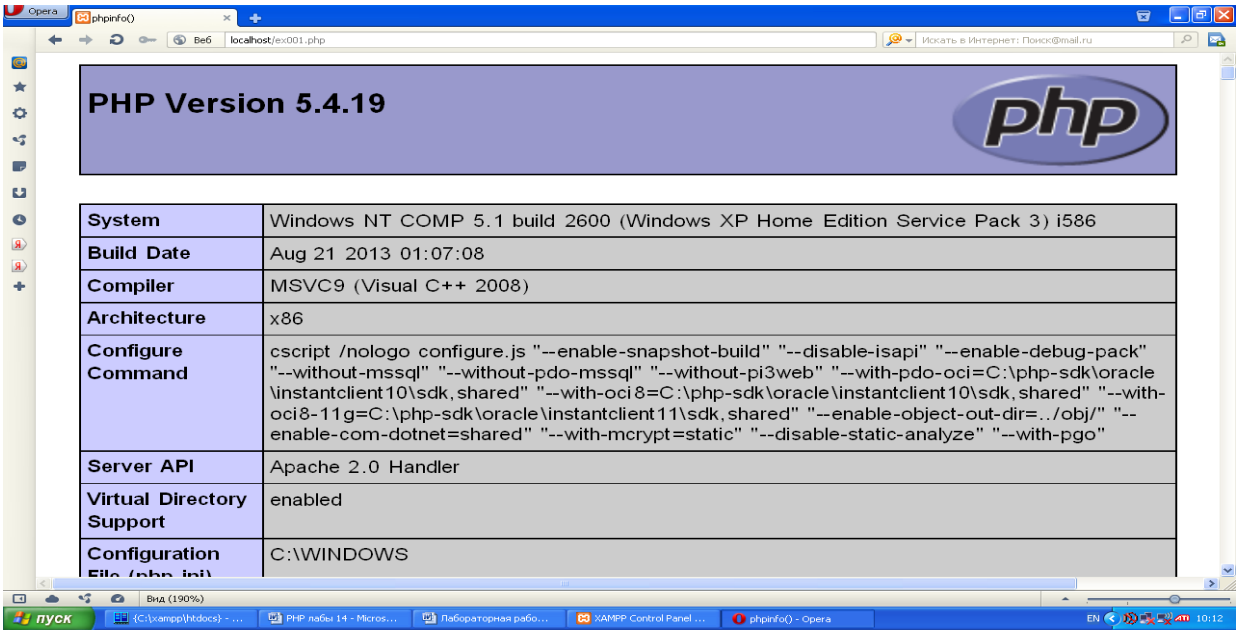

Рис. 3. Информация о настройках, предоставленная phpinfo()

В частности, среди настроек в PHP Variables представлены:

```
_SERVER["SERVER_NAME"] localhost
_SERVER["SERVER_ADDR"] 127.0.0.1
_SERVER["SERVER_PORT"] 80
_SERVER["REMOTE_ADDR"]          127.0.0.1
_SERVER["DOCUMENT_ROOT"]    C:/xampp/htdocs
_SERVER["CONTEXT_DOCUMENT_ROOT"]        C:/xampp/htdocs
```

```
_SERVER["SCRIPT_FILENAME"]    C:/xampp/htdocs/ex001.php
_SERVER["REMOTE_PORT"]        1034
_SERVER["GATEWAY_INTERFACE"]      CGI/1.1
_SERVER["SERVER_PROTOCOL"]  HTTP/1.1
_SERVER["REQUEST_METHOD"]   GET
```

Данные надстройки определяют значения имени (loalhost), адреса (127.0.0.1) и порта (80) сервера, корневой директорий (C:/xampp/htdocs) для размещения сайтов, имя скрипта (C:/xampp/htdocs/ex001.php), запустившего функцию phpinfo(), версию протокола HTTP/1.1 и интерфейса CGI/1.1.

3. СИНТАКСИС И ПРОГРАММНЫЕ ЕДИНИЦЫ PHP

Подробное и полное описание синтаксиса языка PHP можно найти во множестве доступных источников. В этом разделе представлены лишь самые необходимые сведения для решения предлагаемых ниже задач.

Классический пример простейшей программы, которая выводит на экран фразу «Hello, World!», на языке PHP может выглядеть так:

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

Имена всех переменных в PHP обязательно начинаются с символа $, а их тип обнаруживается интерпретатором автоматически. Например:

```
$x = 25;     //  $x переменная типа integer
$y = 25.0;  //  $y переменная типа float
$z = "25";  //  $z переменной типа string
$u = true;  //  $u переменная типа boolean
```

Для вывода на экран значений переменных надо в операторе echo или print использовать двойные кавычки, как показано в примере 1.

Пример 1. Использование кавычек в операторах вывода

```
<?php
$a = "Переменная по имени a.";
    echo "Hello, World! Echo. $a<br>";
    echo 'Hello, World! Echo. $a<br>';
    print "Hello, World! Print. $a<br>"
?>
```

В результате выполнения фрагмента примера 1 браузер отобразит следующие три строки (тег
 обеспечивает перевод строки):

```
Hello, World! Echo. Переменная по имени a.
Hello, World! Echo. $a
Hello, World! Print. Переменная по имени a.
```

Область видимости переменной может быть глобальной или локальной. Пример 2 поясняет это.

Пример 2. Область действия переменных

```
<?php
 $a=1; $b=2; // глобальные переменные
 function Sum () {
```

```
    $a=-1; $b=-2;// локальные переменные
    $a += $b;
  echo "Сумма локальных переменных: $a<br>";
    global $a,$b;
    $a += $b;
  echo "Сумма глобальных переменных: $a<br>";
  }
Sum();
?>
```

В результате выполнения фрагмента примера 2 браузер отобразит следующий результат:

```
Сумма локальных переменных: -3
Сумма глобальных переменных: 3
```

Статическая переменная (определяется как static $a) существует только в локальной области видимости, но при этом не теряет своего значения, когда выполнение программы оставляет эту область.

Для создания констант используется функция define("имя константы" , "значение константы"):

```
define("PORT", "443");
```

При работе со строковыми переменными для соединения двух или более строковых значений в одну строку используется точка (.):

```
$a = "Дважды "; $b = "два"; $c = $a.$c;
echo "$a.$b";          // выведет Дважды два
```

Оператор условий if в PHP имеет несколько реализаций. Обычная форма if:

```
if($name == "sasha")  echo "Привет, Саша!";
                else        echo "Вы кто?";
```

Вторая форма удобна в том случае, если необходимо вставить HTML-код:

```
<?php
    if($s == "yes") :
?>
    <h1>Поздравляем!</h1>
<?php
    endif;
?>
```

В качестве переключателя удобно использовать оператор switch, очень похожий на эквивалентный оператор в C++.

```
switch($what) {
    case 33:          echo "what is 33"; break;
    case "sun":       echo "what is sun"; break;
    default:  echo "what is unknown"; }
```

Как видно из этого примера, у оператора switch в PHP больше возможностей, чем в C++. И еще один пример, который показывает существенное отличие данного оператора в PHP от эквивалентного в C++:

```
switch ($var) {
    case $a: echo "значение var совпадает с a"; break;
    case $b: echo "значение var совпадает с b"; break; }
```

Для организации циклов в PHP можно использовать оператор while. Он имеет два вида – привычный while()

```
$a=0;
while($a <100) {
        $a++;
        echo $a;
}
```

и while(): endwhile;

```
$a=0;
while($a <100):
        $a++;
        echo $a;
endwhile;
```

Также есть оператор do { } while(), использование которого здесь подробно объяснять не будем. И последний оператор for также имеет две формы:

```
for($i = 0; $i < 10; $i++){ echo ($i); }
```

или

```
for($i = 10; $i > 0; $i--):
    echo ($i);
endfor;
```

Семейство функций is..()

Функция isset() используется для определения, присвоено ли переменной какое-либо значение. Если значение присвоено, функция возвращает true:

```
if (isset($a) ) echo "$a";
```

Функция unset() используется для уничтожения переменных, при этом освобождается вся оперативная память, связанная с переменными:

void unset (*любая переменная* [, *любая переменная* [, ...]])

Функции is_int(), is_integer() и is_long() определяют, является ли переменная целым числом:

bool is_int (*любая переменная*);
bool is_long (*любая переменная*);
bool is_integer (*любая переменная*).

Функции is_double(), is_float()и is_real() определяют, является ли переменная числом с плавающей запятой:

bool is_double (*любая переменная*);
bool is_float (*любая переменная*);
bool is_real (*любая переменная*).

Функция is_string() определяет, является ли переменная строкой:

bool is_ *string* (*любая переменная*).

Преобразование типов данных

Существует несколько правил преобразования.

Если строка начинается с допустимого числового значения, то данная строка при выполнении над ней числовой операции будет преобразована в целое число. Если же строка не начинается с числа, то она будет преобразована в 0.

```
< ?php
   $str = "23SomeText";
   $res = 5 + $str;
   echo $res;      // выведет 28
   echo $str;      // выведет 23SomeText
?>
```

Строка переводится в число с плавающей точкой только в том случае, если число занимает всю строку. Если же в строке будут встречены какие-то буквенные символы, то она преобразуется в целое число.

```
< ?php
   $str = "2.3";
   $res = 5 + $str;
   echo $res;          // выведет 5.8
?>
```

Кроме того, существует способ явного преобразования типов:

```
< ?php
   $a = 99.2;
   $a = (int)$a;                    // = 99
   $b = (double)$a;                 // = 99.0
   $c = (string)$a;                 // = "99"
?>
```

Пример 3. Формирование таблицы умножения

Требуется сформировать таблицу умножения для заданного диапазона целых чисел. Диапазон чисел задается пользователем.

В файле table.html создадим форму с двумя полями для ввода двух чисел диапазона с передачей данных по методу post:

table.html

```
<html>
<head> </head>
   <body bgcolor="#99cc99">
     <h2 style="color:#333333;">Таблица умножения</h2>
     <form method="post" action="table.php">
      <p>Введите число начала таблицы умножения</p>
      <input type="Text" name="number1"></input>
      <p>Введите число конца таблицы умножения</p>
      <input type="Text" name="number2"></input>
      <input type="Submit"></input>
     </form>
   </body>
</html>
```

Формирование таблицы будет начинаться при нажатии на кнопку Submit запуском файла table.php.

table.php

```
<html>
<head>
</head>
<body bgcolor="#99cc99">
<?php
      if($number1 == 0 || $number1 ==1)
            $number1 = 2;
      if($number2 == 0){
            $number2=10;
      }
      if($number1 >= $number2){
            echo "<h2>Введите первое число меньше второго!</h2>";
            return;
      }
      echo "<table border=3 cellpadding=5><tr>";
      for($z = 1; $z <=10; $z++)
            echo "<th bgcolor=\"#99cc33\">".$z."</th>";
      echo "</tr>";
      for($i = $number1; $i <= $number2; $i++) {
            echo "<tr>";
            for( $y = 1; $y <= 10; $y++){
```

```
                echo "<td>".$i*$y."</td>";
            }
            echo "</tr>";
        }
        echo "</table>";
?>
</body>
</html>
```

Пример 4. Построение простейшего калькулятора

Построить простейший калькулятор, выполняющий выбранную пользователем операцию.

В файле calc.html строится форма для ввода операндов и выбора выполняемой операции. Введенные данные передаются на обработку в файл calc.php методом post.

calc.html

```
<html> <head> </head>
<body bgcolor="#aab7ab">
<h2 align="center">КАЛЬКУЛЯТОР</h2>
<form action="calc.php" method="post">
      <table align="center" cellspacing=9><tr>
            <td colspan=2>Первое число</td>
            <td colspan=2><input type="text" name="num1"
      value="0"></input></td>
      </tr><tr>
            <td colspan=2>Второе число</td>
            <td colspan=2><input type="text" name="num2"
      value="0"></input></td>
       </tr><tr>
             <td><input type="radio" name="operation" value="plus"
      checked> + </input></td>
             <td><input type="radio" name="operation"
       value="minus"> − </input></td>
              <td><input type="radio" name="operation"
        value="multiply"> * </input></td>
               <td><input type="radio" name="operation"
        value="divise"> / </input></td>
      </tr><tr>
               <td colspan=4 align="center"><input
        type="submit"></input></td></tr></table>
</form>
</body></html>
```

Выполнение выбранной операции над введенными данными будет начинаться при нажатии на кнопку Submit запуском файла calc.php.

calc.php

```
<html><head></head>
<body bgcolor="#aab7ab">
<h2> РЕЗУЛЬТАТ:</h2>
<h1>
<?php
      switch($operation){
            case "plus": echo ($num1+$num2); break;
            case "minus": echo ($num1-$num2); break;
            case "multiply": echo ($num1*$num2); break;
            case "divise": if($num2==0) {
echo "На ноль делить нельзя? ";break;}
           else{echo ($num1/$num2); break; }
            default: echo "Операция не задана"; break; }
?>
</h1></body></html>
```

Функции

Чтобы определить функцию, необходимо воспользоваться оператором function. Для того чтобы возвращать значения из функции, используется оператор return:

```
 function MyFunc ($param)
{if($param == 0)
        return;
  echo ("The parameter is ".$param);
}
. . . .
$number = 91;
MyFunc($number);
```

По умолчанию в функцию аргументы передаются по значению, но их можно передавать и по ссылке, используя & перед именем параметра функции.

Пример 5. Передача аргументов функции по ссылке

```
function Change (&$num1, &$num2)
{       $changer = $num1;
        $num1 = $num2;
        $num2 = $changer;
}
$number1 = 10;
$number2 = 25;
Change ($number1, $number2);
echo $number1; //выведет 25
echo $number2; //выведет 10
```

Можно устанавливать для параметров значение по умолчанию, но, как и в C/C++, такие параметры необходимо помещать в конец.

```
function MyFunc($first, $second = 0)
{       $result = $first + $second;
        return ($result);
}
$num = MyFunc(500);
```

В PHP функции можно вкладывать одну в другую. При этом любая функция, включая и вложенные функции, видна во всем коде, т. е. следующий код будет верен.

Пример 6. Вложение функций

```
function FirstFunc ($param) { // возводит число во вторую степень
  function SecondFunc($num){  // возводит число в третью степень
        return ($num * $num * $num);
  }
  return ($param * $param);
}
$someNumber = 10;
echo (FirstFunc ($someNumber));
echo (SecondFunc (5));
```

Примеры скриптов

Ниже приведены примеры скриптов на PHP, дающие дополнительные сведения об особенностях использования различных программных конструкций. Все примеры даны в виде единого листинга в формате <?php.

Пример 7 (условный оператор)

```
echo "Условный оператор <br>";
$min = -100;
$max = 100;
$i = rand($min, $max);
if ($i > 0) {echo "<p>Число положительное</p>";}
else if ($i < 0) {echo "<p>Число отрицательное</p>";}
else {echo "<p>Ноль</p>";}
```

Пример 8 (цикл for)

```
echo "<h2>Таблица умножения</h2>";
echo "<table>";
for ($i=1; $i<=10; $i++) {
  echo "<tr>";
```

```
 for ($j=1; $j<=10; $j++) {
        echo "<td style = 'background-color: silver;width:25px; height:25px; text-align:center; vertical-align: middle;'>".($i*$j)."</td>";
        }
 echo "</tr>";
 }

echo "</table>";
```

Пример 9 (массивы)

```
$fruits = array("banana", "plum", "apple",  "peach");
sort($fruits); // сортировка – см. документацию PHP "Функции массивов"
$out = "";
foreach ($fruits as $f)
    $out .= "<li>$f</li>"; // конкатенация строк
echo "<ul>$out</ul>";

echo "<p>В массиве fruits ".count($fruits)." элем.</p>";
$fruits[5] = "pinapple";

print_r($fruits); // обратите внимание – элемент с индексом 4 в этом примере будет не определен

echo $fruits[4]; // ошибка! этот элемент не определен
$fruits[4] = "";
echo $fruits[4]; // элемент определен, но содержит пустую строку
```

Пример 10 (многомерные массивы)

```
$vertex[1][0][0] = 1;
$vertex[0][1][0] = 1;
$vertex[0][0][1] = 1;

print_r($vertex);
```

Пример 11 (ассоциативные массивы)

```
$coords[0]["X"] = 55;
$coords[0]["Y"] = 32;
$coords[1]["X"] = 27;
$coords[1]["Y"] = 0.56;

print_r($coords);

$page["head"] = "<head><title>PHP – it's easy</title></head>";
$page["body"] = "<body><p>A simple sample using associative
arrays</p></body>";

print_r($page);

$p = $page["head"].$page["body"];
```

Пример 12 (файловые операции)

```
$f = "read.me";                          // файл в текущей директории
if (file_exists($f))                     // проверка существования файла
  $text = file_get_contents($f);         // чтение из файла
```

file_put_contents($f, $p); // запись в файл, директория должна быть доступна для записи (access rights – 777)

Пример 13 (запись в конец файла)

```
$fd = fopen($f, "a");          // открытие для дозаписи (здесь "a" – append)
$str = "some text";
fwrite($fd, $str);             // запись в конец файла
fclose($fd);
```

Пример 14 (переменные окружения)

```
echo "<h1>Переменные окружения</h1>";
foreach ($_SERVER as $var=>$val)  {
      echo "<p>$"."_SERVER['$var'] = $val</p>";
      };
```

Пример 15 (перенаправление запроса)

```
$url = "index.html";
header("Location:$url"); // функция header является заголовком http, поэтому должна вызываться раньше любого вывода
```

Пример 16 (обработка параметров)

```
echo "<p>Пусть на сервер передан запрос вида
http://myserv.dom/test.php?id=2344&uname=vasya&nick=vasiliok&age=19
(использован метод GET)
<p>Требуется вывести все переменные из строки запроса.";
echo "<ol><li>Использование функции печати массива: <br>";
print_r($_GET);

foreach ($_GET as $key => $val){
  echo "<li>parameter: <b>".$key."</b> value: <b>".$val."</b>";
}

echo "</ul>\n<li>Поэлементный вывод (выводим только значения)<ul>";
  echo "<li>ID: <b>".$_GET["id"]."</b>";
  echo "<li>Firstname: <b>".$_GET["uname"]."</b>";
  echo "<li>Nickname: <b>".$_GET["nick"]."</b>";
  echo "<li>Age: <b>".$_GET["age"]."</b>";

echo "</ul></ol>";
```

```
echo "<p>Проверка, что переданы нужные параметры:";
if (isset($_GET["id"])) {
  echo "<p><b>do something...</b>";
} else {
  echo "<p><b>nothing to do ...</b>";
};

echo "<h2>Проверка, что передан нужный параметр и требуемое значение</h2>";
if ((isset($_GET["id"]))&&($_GET["id"] == 2344)){
  echo "<p><b>do something...</b>";
} else {
  echo "<p><b>nothing to do ...</b>";
}

?>
```

Задачи

Задача 1. Создайте форму с двумя текстовыми полями и кнопкой типа Submit. Пользователь вводит два числа и нажимает копку. Вызывается php-скрипт, который выводит результаты пяти действий над введенными числами. Пример: 5 + 2 = 7, 5 – 2 = 3, 5 * 2 = 10, 5/2 = 2,5, 5%2 = 1.

Задача 2. Создайте форму с текстовым полем, в которое пользователь вводит свой логин, и кнопкой типа Submit. Далее после нажатия на кнопку Submit вызывается php-скрипт, который проверяет, зарегистрирован ли этот пользователь. Зарегистрированных пользователей (разных логинов) должно быть 5. Если введен один из существующих логинов, должно выводится приветствие для этого человека. Например, введен логин sasha, должно быть выведено приветствие: «Здравствуйте, Александр!» Если введен неизвестный логин, должно быть выведено сообщение: «Вы незарегистрированный пользователь».

Задача 3. Составьте программу «Угадай число». При нажатии на кнопку «Загадать число» компьютер должен «загадать» число в диапазоне от 1 до 999. Пользователю предлагается отгадать число (т. е. ввести его в текстовое поле). После нажатия на кнопку «Отгадал?» запускается php-скрипт, который проверяет, отгадал ли пользователь число, и, если нет, пишет текст: «Неверно, загаданное число больше (или меньше)». Далее идет ссылка на текстовое поле с надписью: «Попробуй еще раз!» Если пользователь угадал число, то большими красными буквами выводится: «ВЕРНО (угадано за … попыток)».

Задача 4. Пользователь вводит произвольный диапазон чисел (например, от –3 до 2), php-скрипт должен выводить таблицу отношений числа 10 ко всем числам введенного диапазона. Например:

10/ –3 = –3,3333
10/ –2 = –5
10/ –1 = –10
10/ 1 = 10
10/ 2 = 5

Скрипт должен останавливать цикл, если происходит деление на 0. Перед выводом таблицы отношений проверить, какое число из двух введенных больше, и цикл проводить от меньшего к большему независимо от порядка ввода.

Задача 5. Составьте программу «Угадай число» для цикла по условию. На странице с формой пользователь вводит число и нажимает на клавишу Enter. До тех пор пока пользователь не отгадает число, вызывается скрипт, который сначала выводит сообщение «Не отгадали», а затем снова вызывает файл с формой. Цикл не вызывается, если число отгадано. В процессе угадывания, так же как и ранее (задача 3), предусмотрите подсказки: загаданное число больше (или меньше).

Задача 6. Используя цикл по условию, с проверкой условия при выходе из цикла решите следующую задачу. Известно, что число бактерий удваивается каждый час. Пользователь вводит начальное число бактерий (например, 20) и конечное (например, 1 000 000). Далее начинает работать цикл, выводящий в разных строках примерно следующее:

время 1 ч – уже есть 40 бактерий,
время 2 ч – уже есть 80 бактерий
и т. д., пока не будет превышена конечная цифра.

При вводе числа бактерий пользователем скрипт должен проверить, чтобы начальное число было меньше конечного.

Задача 7. Создайте функцию, которая в качестве аргумента получает произвольное число и затем выводит таблицу умножения на это число в виде (например) 3 · 1 = 3,3 · 2 = 6 и т. д. до 3 · 10 = 30.

Далее, вызывая эту функцию в цикле, получите в таблице (оформленной в HTML) таблицу умножения для чисел от 2 до 9.

Задача 8. В HTML-форме пользователь вводит строки текста. После нажатия на кнопку «ГОТОВО» запускается PHP-скрипт с функцией, кото-

рая в качестве аргументов получает эти три строки и формирует из них таблицу с тремя ячейками. Выше таблицы предусмотрите бегущую строку с текстом первой строки.

Задача 9. В HTML-форме пользователь вводит три числа *a*, *b* и *c*, и после нажатия на кнопку «ГОТОВО» запускается PHP-скрипт, который решает квадратное уравнение: $ax^2 + bx + c = 0$. Для этого в скрипте предусмотрите функцию, которая получает три аргумента (*a*, *b* и *c*) и возвращает значение дискриминанта $D = b^2 - 4ac$. Далее в программе проверяется значение дискриминанта и в зависимости от него вычисляется значение корней или выводится сообщение «Корней нет».

Задача 10. Создайте функцию, которая в качестве параметров принимает два числа и выводит первое число в степени второго. Если второе число не задано, число должно выводится в первой степени.

Задача 11. В первом массиве длиной 4 запишите имена сотрудников, во втором (тоже длиной 4) – их фамилии. Далее выведите их в таблицу с одной строкой и четырьмя ячейками (в каждой имя и фамилия). Сделайте таблицу размером 2 на 2.

4. ПОДКЛЮЧЕНИЕ ВНЕШНИХ ФАЙЛОВ

Как видно из примеров (примеры 3, 4), PHP очень тесно интегрируется с гипертекстом. Такой стиль кодирования называют «спагетти». На самом же деле, для профессиональных разработчиков PHP предлагает возможности разделения кода и данных. В PHP можно распределять содержимое сценариев по нескольким файлам. Например, в одном файле находятся функции и константы, а во втором – код программы, которая выполняется.

Для того чтобы подключить файл, необходимо воспользоваться функцией require(«имя_файла»), либо альтернативной формой для require – include(«имя_файла»):

Пример 17. Использование функции require()

Файл p1.php

```
<?php
  function DemoFunc ( ) {return "Hello";}
?>
```

Файл p2.php

```
<?php
  $var = require ("p1.php");
  echo ($var);   // выводит Hello
?>
```

Есть два основных различия между функциями require() и include():

− функция require() в случае ошибки при подключении файла тут же закончит выполнение PHP-скрипта, т. е. вызовет функцию exit(), а функция include() продолжит выполнение скрипта дальше;

− каждый раз, когда в программе встречается функция require(), на ее место интерпретатор PHP вставляет текст того файла, который она подключает.

При использовании include() происходит то же самое, за исключением того, что при еще одном обращении к этому же include() снова произойдет еще одна вставка, чего не произойдет при использовании функции require().

Пример 18 показывает последнее различие при условии существования двух файлов File1.php и File2.php.

Пример 18. Различие между функциями require() и include()

```
for ($i = 1; $i <= 2; $i++) { require ("File".$i.".php");}
// интерпретатор PHP выдаст ошибку
for ($i = 1; $i <= 2; $i++) {include ("File".$i.".php");}
// интерпретатор PHP сработает
```

Если подключение файла происходит внутри функции, то весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции, т. е. код будет иметь локальную область видимости (пример19).

Пример 19. Подключение файла внутри функции

Файл add.php

```
<?php
  $var1 = 'var1 из add.php';
  $var2 = 'var2 из add.php';
?>
```

Файл test.php

```
<?php
   function func() {
    // объявили $var1 глобальной переменной
    global $var1;
       include 'add.php';
       echo "Внутри функции: $var1 $var2";
  }
  func();
  echo "<br>В глобальной области: $var1";
 ?>
```

Так как внутри функции мы объявили переменную $var1 глобальной, она становится доступной и в глобальной области видимости.

5. МАССИВЫ

Индексом массива в PHP может быть как целое число (по умолчанию отсчет элементов начинается как обычно – с нуля), так и строка. Массивы в PHP могут содержать элементы нескольких разных типов.

Для инициализации массива существуют несколько способов. Вот один из способов:

```
$massiv[0] = "first";
$massiv[1] = "second";
$massiv[2] = "third".
```

То же самое можно записать и другим способом:

```
$massiv[] = "first";
$massiv[] = "second";
$massiv[] = "third".
```

Данный код создаст массив из трех элементов и присвоит индексы элементам массива по умолчанию: 0, 1, 2, т. е. будет создан массив с последовательной индексацией.

Можно присваивать любой целый индекс по своему усмотрению:

```
$massiv[25] = "first";
$massiv[10] = "second";
$massiv[49] = "third".
```

Данный код создает массив из трех элементов, но с индексами 10, 25 и 49. Отсюда понятно, что определить количество элементов в массиве по наибольшему индексу в массиве нельзя. Для этого используется функция count(имя массива). Данная функция возвращает 0 в случае, если такого массива нет или если в массиве нет элементов. Если имеется массив с последовательной индексацией, то можно использовать функцию count() для работы с массивом (пример 20).

Пример 20. Использование функции count()

```
$massiv = array("first", "second", "third");
$num = count($massiv);
for($i = 0; $i < $num; $i++){ echo (massiv[$i]."<br>"); }
```

Можно совмещать способы инициализации, например:

```
$massiv[25] = "first";
$massiv[10] = "second";
$massiv[ ] = "third".
```

В данном случае третьему элементу массива присвоится индекс 26.

Другой способ инициализации массива заключается в использовании функции array():

```
$massiv = array("first", "second", "third");
```

Функция создаст массив с тремя элементами, присвоив им индексы 0, 1 и 2 соответственно. Чтобы присваивать индексы по своему усмотрению, необходимо использовать в функции array() оператор «=>«:

```
$massiv = array(5 => "first",  15 => "second",  10 => "third");
$massiv = array(9 => "first",  "second",  21 => "third");
```

В первом случае создается массив из трех элементов с индексами 5, 10 и 15, во втором – с индексами 9, 10 и 21. Обратите внимание, что оператор => можно использовать перед любым элементом массива.

Чтобы присвоить элементу массива индекс в виде строки, необходимо выполнить такие же действия:

```
$massiv["first"] = "five";
$massiv["second"] = "six";
```

или

```
$massive = array( "first" => "five", "second" => "six").
```

Теперь рассмотрим функции, которые помогут работать с массивом, если он индексирован непоследовательно или индексирован строками. Следующие функции часто используются вместе для работы с элементами массива:

− reset(имя массива) – устанавливает указатель на первый элемент массива;

− each (имя массива) – возвращает очередную пару (ключ, значение) из массива, а когда массив закончится, возвращает FALSE;

− list(*$key, $value*) – присваивает переменным значение текущего индекса и элемента массива.

Пример 21. Использование функций reset(), list() и each()

```
reset($massiv);
while (list($key, $value) = each($massiv))
{
echo "Array element with key = ".$key." has value = ".$value."<br>";
}
```

Следующие две функции next(), prev() перемещают внутренний курсор в массиве на один элемент вперед или назад соответственно, т. е. возвращают значение следующего или предыдущего элемента соответственно. В случае если достигнут конец или начало массива, функции возвращают FALSE.

К функциям сортировки массивов относятся:

sort(array, flag) – сортировка массива, где flag – необязательный параметр:

SORT_REGULAR – обычная сортировка,

SORT_NUMBER – значения массива рассматриваются как числа,

SORT_STRING – значения массива рассматриваются как строки;

asort(array) – сортировка массива с сохранением порядка индексов;

rsort(array) – аналогична sort(), но в обратном порядке;

arsort(array) – аналогична asort(), но в обратном порядке;

ksort(array) – сортировка массив по индексам;

krsort(array) – аналогична ksort(), но в обратном порядке.

Функция usort() несколько сложнее. В качестве одного аргумента, как и все остальные функции сортировки, она принимает массив. Однако у нее есть и второй аргумент. Этим вторым аргументом является функция, определив которую, можно сообщить usort (), как осуществлять сортировку. В следующем примере массив сортируется по длине строк, содержащихся в его элементах. Функция strlen() возвращает длину строки.

Пример 22. Сортировка массива по длине строк

```
function by_length ($a, $b)
{       $1_a = strlen ($a);
        $1_b = strlen ($b);
        if ($l_a == $l_b) return 0;
        return ($l_a < $l_b) ? -1 : 1;
}
$countries = array (  "e"   => "United States" ,
                      "d"   => "United Kingdom",
                      "c"   => "Canada",
                      "b"   =;> "Costa Rica",
                      "a"   => "Germany");
usort ($countries, by_length);
while (list ($key, $val) = each ($countries))
{ echo "Element $key equals $val<BR>\n"; }
```

Возможно, вы заметили, что в результате снова потеряны наши строковые индексы. Функция uasort () действует так же, как usort (), но сохраняет исходные индексы. Функция uksort () действует аналогично функции usort (), но сравнивает ключи, а не значения элементов массива.

А что если мы хотим десортировать массив? Функция shuffle () с помощью имеющегося в PHP генератора случайных чисел переставляет элементы массива в случайном порядке. В примере 22 функция shuffle () используется вместе с функцией range () в целях создания массива из ста случайно расположенных чисел. Функция range () принимает два целочис-

ленных параметра, первый из которых должен быть меньше второго, и возвращает массив, состоящий из всех целых чисел между этими двумя значениями.

Пример 22. Десортировка массива

```
// Создать массив чисел от 1 до 100:
$ints = range (1, 100);
// Задать начальное состояние генератора случайных чисел:
srand (time());
shuffle ($ints);
while (list ( , $num) = each ($ints)) {
echo "$num<br>\n"; }
```

Массивы особенно удобны при работе с данными HTML-форм табличного вида. При желании предоставить пользователю возможность ввести в базу данных несколько имен можно воспользоваться следующей формой:

```
<input name="first1" type=text> <input name ="last1" type=text>
<input name="first2" type=text> <input name="last2" type=text>
<input name="first3" type=text> <input name="last3" type=text>
```

Каждое текстовое окно имеет уникальное имя («firstl», «lastl», «first2» и т. д.). При передаче формы для каждого текстового окна на форме будет создана переменная PHP ($firstl, $lastl, $first2 и т. д.). Однако если количество строк может меняться, как, вероятно, и будет, то более удобно представить каждое поле как массив данных. В HTML это делается с помощью квадратных скобок после имени элемента. Можно использовать цикл PHP for для размещения на форме необходимого числа текстовых окон (пример 23).

Пример 23. Цикл для размещения необходимого числа окон

```
<?php
$names = 3; ?>
<form action=”submit.php"   method=post>
<? for ($i = 1; $i <= $names; $i++): ?>
        first name:
        <input name="first[]" type=text><br>
        last name:
        <input name="last[] "  type=text><br><br>
<? endfor ?>
<br><input type=submit>
</form>
```

При передаче этой формы php создаст массивы с именами $first и $last, каждый элемент которых будет содержать значение текстового окна. Это позволяет легко обработать переданные данные в сценарии php. Допустим, что мы хотим добавить переданные имена в таблицу базы данных. Можно создать команду SQL INSERT для каждой пары имен при циклическом обходе массива (пример 24).

Пример 24. Циклическое использование команды SQL INSERT

```
<!— submit .php —>
<?php
$numrows = count ($first);
for ($i = 0; $i < $numrows; ++$i) {
$sql = "INSERT INTO Names ('First', 'Last') " . "VALUES (`$first[$i]', '$last[$i]')";
// Здесь код для выполнения запроса. // ... // ... } ?>
```

Задачи

Задача 12. В массив внесите 5 произвольных чисел. Далее, используя конструкцию foreach, выведите их и их квадраты в виде: $4^2 = 16$ $2^2 = 4$ $5^2 = 25$ и т. д. Оформите вывод в виде таблицы с одной строкой и пятью ячейками, при этом фон у каждой ячейки должен отличаться от фона других ячеек.

Задача 13. Создайте и заполните ассоциированный массив, в котором именами элементов будут: название страны, население и название столицы. Выведите из массива таблицу в три строки по две ячейки в каждой. В левой ячейке имя элемента, в правой – его значение.

Измените таблицу: 2 строки по 3 элемента. Выведите в первой строке только имена элементов, во второй – значения.

Задача 14. В HTML-форме пользователь вводит в четыре разных поля: фамилию, имя, возраст и e-mail. После нажатия на клавишу кнопки «ГОТОВО» запускается php-скрипт, который вносит эти данные в ассоциированный массив и далее выводит их, используя конструкцию while (list ($key, $val) = each ($countries)). В форме предусмотрите проверку того, что все поля перед отправкой не пустые.

Задача 15. Заполните в циклах первый массив квадратами чисел от 10 до 20, а второй – кубами чисел от 1 до 10. Далее объедините эти массивы и выведите объединенный.

Задача 16. В HTML-форме предусмотрите поля для ввода фамилий и результатов по прыжкам в длину для четырех спортсменов (т. е. всего 8). Пользователь вводит данные и в выпадающем списке выбирает «Сортиро-

вать по фамилиям» или «Сортировать по результатам». После нажатия кнопки «ГОТОВО» в зависимости от выбора в php-скрипте сначала объединяются фамилии и результаты в один массив и выводятся в отсортированном виде. Предусмотрите проверку, чтобы результат всегда был трехзначным числом (от 100 до 999) и все фамилии были вписаны.

Задача 17. В HTML-форме предусмотрите поля для ввода результатов по прыжкам в длину для четырех спортсменов, фамилии которых заранее известны. Пользователь вводит данные и в выпадающем списке выбирает «Сортировать по убыванию» или «Сортировать по возрастанию». После нажатия кнопки «ГОТОВО» вызывается php-скрипт, который вносит данные в ассоциированный массив (имена полей – фамилии спортсменов) и выводит массив в отсортированном виде (по убыванию или по возрастанию). Перед сортировкой предусмотрите преобразование введенных данных в вещественные числа.

Задача 18. В HTML-форме предусмотрите поле для ввода фамилии пользователя и список книг в виде checkbox. Пользователю предложите отметить те из них, которые он читал, и далее, используя php-скрипт, выведите их в виде отсортированного списка. Дополнительно предусмотрите в форме выпадающий список с этими же книгами, в котором пользователь укажет одну любимую. При выводе списка книг любимая должна быть выделена жирным шрифтом.

6. ФУНКЦИИ ДЛЯ РАБОТЫ С ДАТОЙ И ВРЕМЕНЕМ

Для работы со временем в языке PHP есть несколько встроенных функций.

Функция checkdate проверяет правильность даты/времени:

```
int checkdate (int month, int day, int year);
```

Возвращает true, если данная дата правильна (год между 1900 и 32767 включительно, месяц между 1 и 12 включительно, день находится в диапазоне разрешенных дней данного месяца с учетом високосных лет), иначе false.

Функция date определяет формат локального времени/даты:

```
string date (string format, int timestamp);
```

В форматной строке должны использоваться следующие символы:

a – «am» или «pm»;
A – «AM» или «PM»;
d – день месяца, цифровой, 2 цифры (на первом месте ноль);
D – день недели, текстовый, 3 буквы; т. е. «Fri»;
F – месяц, текстовый, длинный; т. е. «January»;
h – час, цифровой, 12-часовой формат;
H – час, цифровой, 24-часовой формат;
i – минуты, цифровой;
j – день месяца, цифровой, без начальных нулей;
l (строчная «L» – день недели, текстовый, длинный; т. е. «Friday» ;
m – месяц, цифровой;
M – месяц, текстовый, 3 буквы; т. е. «Jan»;
s – секунды, цифровой;
S – английский порядковый суффикс, текстовый, 2 символа; т. е. «th», «nd»;
U – секунды с начала века;
Y – год, цифровой, 4 цифры;
w – день недели, цифровой, 0 означает воскресенье;
y – год, цифровой, 2 цифры;
z – день года, цифровой, т. е. «299».

Нераспознанные символы в форматной строке будут печататься как есть.

Функция getdate получает информацию о дате/времени:

array getdate (int timestamp);

Данная функция возвращает ассоциативный массив, содержащий информацию о дате со следующими элементами:

"seconds" – секунды;
"minutes" – минуты;
"hours" – часы;
"mday" – день месяца;
"wday" – день недели, цифровой;
"mon" – месяц, цифровой;
"year" – год, цифровой;
"yday" – день года, цифровой; т. е. «299»;
"weekday" – день недели, текстовый, полный; т. е. «Friday»;
"month" – месяц, текстовый, полный, т. е. «January».

7. ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ

В php используются следующие функции для работы со строками:

AddSlashes – выделяет строку обратной чертой, возвращает строку с обратной чертой (/) перед символами: ('), двойные кавычки («), (\) и NUL (нулевой байт), которые должны быть выделены в запросах к базам данных и т. п.:

string addslashes(string str);

• **Chop** – удаляет повторяющиеся пробелы, возвращает строку без повторяющихся пробелов:

string chop(string str);

• **explode** – разбивает строку на строки, возвращает массив строк, содержащий элементы, разделенные строкой separator:

array explode(string separator, string string);

• **implode** – объединяет массив элементов в строку, возвращает строку, содержащую совокупность всех элементов массива в том же порядке, со строкой glue между каждым элементом:

string implode(array pieces, string glue);

• **ltrim** – удаляет пробелы из начала строки и возвращает обрезанную строку:

string ltrim(string str);

• **nl2br** – переводит символы новой строки в HTML-тег разрыва строки, возвращает string с '
', вставленными перед каждой новой строкой:

string nl2br(string string);

• **sprintf** – возвращает строку, обрабатываемую в соответствии с форматирующей строкой format:

sprintf(string format, mixed [args]...);

Идентификатор типа format говорит о том, как тип данных аргумента должен трактоваться:

% – символ процента. Аргумент не требуется;

b – аргумент трактуется как integer и представляется как двоичное число;

c – аргумент трактуется как integer и представляется как символ с ASCII значением;

d – аргумент трактуется как integer и представляется как десятичное число;

f – аргумент трактуется как double и представляется как число с плавающей точкой;

o – аргумент трактуется как integer и представляется как восьмеричное число;

s – аргумент трактуется и представляется как строка;

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);

• **strchr** – находит первое появление символа:

string strchr(string haystack, string needle);

• **strcmp** – двоичное сравнение строк (безопасное), возвращает одно из значений < 0, если str1 меньше, чем str2; > 0, если str1 больше, чем str2; 0, если они равны. Следует отметить, что это сравнение чуствительно к регистру:

int strcmp(string str1, string str2);

• **StripSlashes** – удаляет символы \ из строки, а двойные символы \\ заменяет на одинарные \:

string stripslashes(string str);

• **strlen** – возвращает длину строки:

int strlen(string str);

• **strrev** – переворачивает строку, возвращает перевернутую строку string:

string strrev(string string);

• **strtolower** – переводит строку в нижний регистр:

string strtolower(string str);

• **strtoupper** – переводит строку в верхний регистр:

string strtoupper(string string);

• **substr** – возвращает часть строки string, определяемую параметрами start (начало) и length (длина):

string substr(string string, int start, int [length]);

• **trim** – обрезает пробелы с начала и с конца строки:

string trim(string str);

8. ФАЙЛЫ И ДИРЕКТОРИИ

В php есть функции для работы с файловой системой:

• **fopen** – служит для открытия файла:

```
int fopen(string filename, string mode)
```

Если при открытии файла происходит ошибка, функция возвращает false.

Параметр *mode* выбирается из табл. 2.

Таблица 2

Значения параметра mode

Значение mode	Выполняемая операция
r	Открыть только для чтения, помещает указатель на начало файла
r+	Открыть для чтения и для записи, помещает указатель на начало файла
w	Открыть только для записи, помещает указатель на начало файла и очищает все содержимое файла. Если файл не существует, создается новый файл
w+	Открыть для чтения и для записи, помещает указатель на начало файла и очищает все содержимое файла. Если файл не существует, создается новый файл
a	Открыть только для записи, помещает указатель на конец файла. Если файл не существует, создается новый файл
a+	Открыть для чтения и для записи, помещает указатель на конец файла. Если файл не существует, создается новый файл

На платформе Windows используются передние слеши:

```
$fp = fopen("c:\\data\\info.txt", "r");
```

• **fclose** – служит для закрытия файлов и возвращает размер файла или false в случае ошибки:

```
int fclose(int fp);
```

Параметр fp является указателем на файл, который надо закрыть. Функция возвращает true при удачной операции и false при ошибке.

• **basename** – возвращает из полного пути имя файла:

```
string basename(string path);
```

• **copy** – создает копию файла, возвращает true при успешном завершении, в противном случае – false:

```
int copy(string source, string dest);
```

• **dirname** – получив строку, содержащую путь у файлу, данная функция возвращает директорию, содержащую файл:

```
string dirname(string path);
```

• **feof** – проверяет достижение указателем конца файла, возвращает true, если указатель файла равен EOF, в противном случае возвращает false:

```
int feof(int fp);
```

• **fgetc** – получает символ из файла, возвращает строку, содержащую один символ, прочитанный по файловому указателю fp. При EOF возвращается false:

string fgetc(int fp);

• **fgets** – получает строку по указателю на файл:

string fgets(int fp, int length);

Возвращает строку до length – читается по одному байту из файла, указанного в fp. Чтение заканчивается, если прочитано length символов (1 байт прочитается в любом случае – или до символов перевода строки и возврата каретки, или до EOF). При ошибке возвращается false;

• **file** – прочитать файл в массив:

array file(string filename);

Функция идентична readfile(), но file() выдает файл в массив. Каждый элемент массива соответствует строке файла (вместе с символом возврата строки).

• **file_exists** – проверяет существование искомого файла:

int file_exists(string filename);

Возвращает true, если файл, определенный в filename, существует; иначе – false.

• **fileperms** – выделяет разрешения для файла:

int fileperms(string filename);

Возвращает разрешения, установленные для файла, или false в случае ошибки.

• **filesize** – возвращает размер файла, или false в случае ошибки:

int filesize(string filename);

• **fpassthru** – выводит все данные с позиции указателя файла:

int fpassthru(int fp);

Читает до EOF и записывает результат на стандартное устройство вывода. При возникновении ошибки fpassthru() возвращает false.

• **ftell** – определяет текущую позицию указателя в файле:

int ftell(int fp);

Возвращает позицию указателя в файле, на которую ссылается fp, т. е. на смещение в потоке файла. При возникновении ошибки возвращается false;

• **fseek** – устанавливает указатель на компоненту файла с заданным номером:

int fseek(int fp, int offset);

offset – количество байт (символов), на которое нужно сместиться. Возвращает 1, если произошла ошибка, и 0, если операция прошла успешно;

• **fwrite** – осуществляет бинарную запись в файл:

int fwrite(int fp, string string, int [length]);

Записывает содержимое string в файловый поток, указанный fp. Если аргумент length присутствует, запись останавливается после записи length байта, или после записи всей строки string;

• **filesize** – возвращает размер файла:

int filesize(string filename);

• **is_dir** – возвращает true, если filename существует и это директория:

bool is_dir(string filename);

• **is_file** – возвращает true, если filename существует и является обычным файлом:

bool is_file(string filename);

• **mkdir** – создает директорию:

int mkdir(string pathname);

Пытается создать директорию, указанную в pathname. Возвращает true при успешном выполнении и false при ошибке;

• **rename** – переименовывает файл:

int rename(string oldname, string newname);

Пытается переименовать oldname в newname. Возвращает true при успешном выполнениии и false при сбое;

• **rewind** – позиционирует файловый указатель для fp на начало потока файла:

int rewind(int fp);

При возникновении ошибки возвращается 0;

• **rmdir** – удаляет директорию:

int rmdir(string dirname);

Пытается удалить директорию, указанную путем. Директория должна быть пустой, и релевантные разрешения должны допустить это. При возникновении ошибки возвращается 0;

• **unlink** – удаляет файл:

int unlink(string filename);

Удаляет filename. Возвращает 0 или false при ошибке;

• **chdir** – осуществляет смену каталога:

int chdir(string directory);

Возвращает false, если не может изменить, и true, если смена произошла;

• **closedir** – закрывает дескриптор каталога:

void closedir(int dir_handle);

Закрывает поток каталога, обозначенный как dir_handle;

• **opendir** – открывает дескриптор каталога:

int opendir(string path);

Возвращает дескриптор каталога, который в последующем используется в closedir(), readdir(), и rewinddir() обращениях;

• **readdir** – читает данные из каталога по дескриптору (handle):

string readdir(int dir_handle);

Возвращает имя следующего файла из каталога. Имена возвращаются в любом специфическом порядке:

```
$handle=opendir('.');
```

```
while ($file = readdir($handle)) {
   echo "$file\n";
}
closedir($handle);
```

• **rewinddir** – возвращает к началу данных каталога по дескриптору:
void rewinddir(int dir_handle);

Сбрасывает поток каталога, обозначенный как dir_handle, в начало данных.

Задачи

Задача 19. Используя php-скрипт и форму в одном документе, создайте сценарий, в котором пользователь вводит в текстовом поле имя файла и после нажатия на кнопку «ГОТОВО» проверяется, существует ли этот файл. Если он не существует, выводится сообщение вида «файл name.txt не существует», а если файл существует, выводятся данные о его имени (полный путь), размере, времени создания и последней модификации. Если файл существует, включите его содержимое внутрь тегов <textarea>, т. е. дайте возможность увидеть его пользователю. В случае отсутствия файла теги <textarea> в документ включены быть не должны.

Задача 20. В текстовом файле в первой строке вписать тег (без скобок <>), во второй – его описание, в третьей – второй тег, в четвертой – описание, и далее до 5-6 тегов. Далее в php-скрипте прочитайте файл построчно и выведите в виде таблицы.

 	Разрывает строку
<td>	Открывает ячейку в таблице

Скрипт должен сосчитать, сколько всего тегов описаны в файле и вывести ответ ниже таблицы. Например, «всего описано 7 тегов».

Задача 21. Составьте следующий сценарий. Пользователь входит на страницу, в текстовом поле вводит свой логин, а в выпадающем списке выбирает режим «регистрация» или «вход». В случае выбора «вход» проверяется, существует ли файл с именем, аналогичным логину, и если да, в страницу добавляется поле для ввода пароля. Далее после ввода пароля проверяется, соответствует ли он содержимому файла, и если да, выводится сообщение «Добро пожаловать, зарегистрированный пользователь Имярек!», в противном случае – сообщение «Пароль не верен». Если файла с именем пользователя, входящего как зарегистрированный, не обнаружено, выведите сообщение «Такой пользователь не зарегистрирован». Если

пользователь входит в режиме «Регистрация», проверьте, не занят ли логин, и если он свободен, создайте файл с именем логина и после ввода пароля напишите его в этот файл. Если логин занят, предложите сменить логин.

Задача 22. Напишите скрипт, считывающий из выбранного текстового файла строки и сортирующий их. Отсортированный результат запишите в файл sortline.txt.

Задача 23. Напишите скрипт, выводящий на экран дерево каталога и пропускающий файлы, в которые запрещена запись. Подсказка: воспользуйтесь материалами урока 8 (работа с файлами) из [1].

9. ЗАГРУЗКА КЛИЕНТОМ ФАЙЛОВ НА СЕРВЕР

В языке PHP реализована возможность загрузки файлов на сервер. Клиент может загрузить на сервер любой файл. Для того чтобы файлы можно было отправлять из браузера, необходимо создать элемент управления <input> типа file, позволяющий пользователю выбрать произвольный файл, нажав на клавишу «Обзор…». Кроме того, для загрузки файлов необходимо установить атрибут формы enctype равным «multipart/form-data» и, как обычно, атрибут action равным php-сценарию, который будет обрабатывать загрузку файла. В результате должно получится, как в примере 25.

Пример 25. Построение

```
<form action="имя_php_сценария" method="post"
enctype="multipart/form-data">
    Введите имя файла: <input type="file" name="userfile"><br>
    <input type="submit"><br>
 </form>
```

В результате при нажатии на кнопку «Подача запроса» файл автоматически будет загружен на сервер. На сервере он может быть размещен в директории TEMP или в директории, где находится php-интерпретатор. Автоматически файлу присваивается имя *phpX*, где X – значение по порядку.

Если файл в дальнейшем не скопировать в другую директорию, то он будет автоматически уничтожен при выходе из выполняемого php-сценария. Но это еще не все, что можно сделать. Еще можно наложить ограничения на загрузку файлов на сервер при помощи элемента управления **INPUT** с атрибутом **NAME**, равным **MAX_FILE_SIZE** и **VALUE**, равному верхней границе для загружаемого файла, но с числом, не превышающим размер, определенный в файле *php.ini* как upload_max_filesize (по умолчанию 2 Мбайт). При этом помимо переменной, отвечающий за файл, в php-сценарии появляются еще три переменные:

$userfile_name – путь и имя файла на стороне клиента;
$userfile_size – размер файла в байтах;
$userfile_type – тип файла.

Обработать загруженный файл на сервере тоже просто. Файл сохраняется как phpx (где x является наращиваемым целым числом) во временном каталоге (этот каталог можно задать с помощью переменной окружения TEMPDIR). В конце запроса файл автоматически уничтожается, поэтому при необходимости дальнейшего использования его нужно скопировать из той же страницы. Доступ к имени файла осуществляется тем же

способом, что и ко всем данным формы: с помощью имени, указанного в элементе ввода как переменная php (в данном случае – $userfile).

Файл можно скопировать в место постоянного хранения с помощью функции copy(), рассмотренной ранее в этом разделе. Хотя временный загруженный файл автоматически уничтожается в конце запроса, лучше после копирования уничтожить его явным образом с помощью функции unlink (). Поэтому, например, чтобы скопировать наш загруженный файл в C:\upload. txt (в Windows), а затем удалить его, можно использовать такой код:

```
<HTML>
<!-- upload.php —> <?
// Копировать файл в C:\upload.txt. Не забудьте преобразовать обратную косую черту!
if (copy($userfile, "C:\\upload.txc")) { echo("<B>File successfully copied!</B>"); }
else { echo("<B>Error: failed to copy file...</B>"); }
// Уничтожить файл после копирования
unlink($userfile) ;
?></HTML>
```

10. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Регулярное выражение – механизм, позволяющий задать шаблон для строки и осуществить поиск данных, соответствующих этому шаблону в заданном тексте. Кроме того, дополнительные функции по работе с регулярными выражениями позволяют результаты поиска получить в виде массива строк, произвести замену в тексте по шаблону, разбиение строки по шаблону и т. п. Однако главной их функцией, на которой основаны все остальные, является именно функция поиска в тексте данных, соответствующих шаблону, описанному в синтаксисе регулярных выражений.

Регулярные выражения пришли к нам из Unix и Perl. В php существует два различных механизма для обработки регулярных выражений: POSIX-совместимые и Perl-совместимые. Их синтаксис во многом похож, однако Perl-совместимые регулярные выражения более мощные и работают намного (в некоторых случаях до 10 раз) быстрее. Поэтому здесь будем рассматривать только Perl-совместимые регулярные выражения.

Сутью механизма регулярных выражений является то, что они позволяют задать шаблон для нечеткого поиска по тексту. Например, если стоит задача найти в тексте определенное слово, то с этой задачей хорошо справляются и обычные функции работы со строками. Например, если необходимо найти в тексте информацию, а про нее известно только то, что это «3 или 4 цифры, после которых через пробел идет 5 заглавных латинских букв», то это можно сделать, воспользовавшись следующим регулярным выражением:

```
/\d{3,4}\s[A-Z]{5}/.
```

Регулярные выражения, как уже было сказано выше, представляют собой строку. Строка всегда начинается с символа разделителя, за которым следует непосредственно регулярное выражение, затем еще один символ разделителя и потом необязательный список модификаторов. В качестве символа разделителя обычно используется слэш ('/'). Таким образом, в регулярном выражении

```
/\d{3}-\d{2}/m
```

символ '/' является разделителем, строка '\d{3}-\d{2}' – непосредственно регулярным выражением, а символ 'm', расположенный после второго разделителя, – модификатором.

Основой синтаксиса регулярных выражений является тот факт, что некоторые символы, встречающиеся в строке, рассматриваются не как обычные символы (литералы), а как имеющие специальное значение (*метасимволы*). Именно это решение позволяет работать механизму регулярных выражений. Каждый метасимвол играет свою роль в синтаксисе регулярных выражений.

Метасимволы

Одним из самых важных метасимволов является символ обратного слэша ('\'). Если в регулярном выражении встречается этот символ, то парсер рассматривает символ, непосредственно следующий за ним двояко:

- если следующий символ имеет какое-либо специальное значение, то он теряет это свое специальное значение и рассматривается как обычный символ. Это совершенно необходимо для того, чтобы иметь возможность вставлять в строку специальные символы, как обычные. Например метасимвол '.', в обычном режиме означает «*любой единичный символ*», а '\.' означает просто точку. Также можно лишить специального значения и сам этот символ: '\\'.
- если следующий символ не имеет никакого специального значения, то он может получить такое значение, будучи соединенным с символом '\'. К примеру символ 'd' в обычном режиме воспринимается просто как буква, однако, будучи соединенной с обратным слэшем ('\d') становится метасимволом, означающим «*любая цифра*».

Существует множество символов, которые образуют метасимволы в паре с обратным слэшем. Как правило, подобные пары используются для того, чтобы показать, что на этом месте в строке должен находиться символ, с кодом, который не имеет соответствующего ему изображения или же символ, принадлежащий какой-то определенной группе символов. Некоторые наиболее употребительные метасимволы приведены в табл. 3.

Таблица 3

Наиболее употребительные метасимволы

Метасимвол	Значение
Метасимволы для задания символов, не имеющих изображения	
\n	Символ перевода строки (код 0x0A)
\r	Символ возврата каретки (код 0x0D)
\t	Символ табуляции (код 0x09)
\xhh	Вставка символа с шестнадцатиричным кодом 0xhh, например \x41 вставит латинскую букву 'A'
Метасимволы для задания групп символов	
\d	Цифра (0–9)
\D	Не цифра (любой символ, кроме символов 0–9)
\s	Пустой символ (обычно пробел и символ табуляции)
\S	Непустой символ (все, кроме символов, определяемых метасимволом \s)
\w	«Словесный» символ (символ, который используется в словах, обычно все буквы, все цифры и знак подчеркивания)
\W	Все, кроме символов, определяемых метасимволом \w

Приведем несколько простейших примеров для понимания того, о чем идет речь. Сразу оговоримся, что примеры несколько громоздки и некрасивы, но лишь потому, что в них не использованы метасимволы, о которых еще речь впереди и которые сделали бы примеры намного проще.

Пример 26. Регулярные выражения

/\d\d\d/ – любое трехзначное число ('123', '719', '001')
/\w\s\d\d/ – буква, пробел (или табуляция) и двузначное число ('A 01', 'z 45', 'S 18')
/\d and \d/ – любая из следующих строк: '1 and 2', '9 and 5', '3 and 4'.

Синтаксис регулярных выражений имеет средства для определения собственных подмножеств символов. Например, может понадобиться задать условие, что в этом месте строки должна находиться шестнадцатеричная цифра или еще что-то подобное. Для описания таких подмножеств применяются символы *квадратных скобок* ′[ ]′. Квадратные скобки, встреченные внутри регулярного выражения, считаются одним символом, который может принимать значения, перечисленные внутри этих скобок.

Есть небольшая тонкость в том, как работают метасимволы внутри квадратных скобок. Дело в том, что в синтаксисе регулярных выражений существует еще множество метасимволов, но практически все они работают только **вне** секций описаний подмножеств. Метасимволы, которые работают внутри этих секций, это:

- обратный слэш «\», т. е. все метасимволы из табл. 3 будут работать;
- минус «–», используется для задания набора символов из одного промежутка (например, все цифры могут быть заданы как «0–9»);
- символ «^». Если этот символ стоит **первым** в секции задания подмножества символов (и только в этом случае!), он будет рассматриваться как символ отрицания. Таким образом можно задать все символы, которые **не описаны** в данной секции.

Приведем несколько примеров, чтобы было понятно, как это работает (табл. 4).

Таблица 4

Примеры регулярных выражений

Regexp	Комментарии
[0-9A-Fa-f]	Цифра в шестнадцатеричной системе счисления
[\dA-Fa-f]	То же самое, но с использованием метасимвола
[02468]	Четная цифра
[^\d]	Все, кроме цифр (аналог метасимвола \D)
[a^b]	Любой из символов «a», «b», «^». Заметьте, что здесь символ «^» не имеет какого-либо специального значения, потому что стоит не на первой позиции внутри квадратных скобок

Теперь необходимо рассмотреть еще несколько метасимволов. Как уже было сказано ранее, все они работают только вне секций описаний подмножеств символов (вне квадратных скобок).

Символы «^» и «$».** Они используются для того, чтобы указать парсеру регулярных выражений на то, чтобы он обратил внимание на положение искомого текста в строке. **Символ «^»** указывает, что искомый текст должен находиться в начале строки, **символ «$», наоборот, указывает, что искомый текст должен находиться в конце строки. Посмотрим, как это работает на примере.

Допустим, у нас есть текст:

12 aaa bbb
aaa 27 ccc
aaa aaa 45

и регулярное выражение для поиска чисел в этом тексте: **/\d\d/m** (не обращайте пока внимания на модификатор). Поиск по этому регулярному выражению вернет нам 3 значения: 12, 27, 45. Теперь ограничим поиск, указав, где именно внутри строки должен располагаться текст: **/^\d\d/m**. Здесь результат будет только один – 12, потому что только это число располагается в начале строки. Аналогично, регулярное выражение **/\d\d$/m** вернет результат 45.

Символ точки «.». Этот метасимвол указывает, что на данном месте в строке может находиться любой символ (за исключением символа перевода строки). Очень удобно использовать его, если вам нужно «пропустить» какую-нибудь букву в слове при проверке. Например, регулярное выражение /.bc/ найдет в тексте abc, и Abc, и Zbc, и 5bc.

Символ вертикальной черты «|». Используется для задания списка альтернатив. Например, регулярное выражение

/(красное |зеленое) яблоко/

позволит найти в тексте все словосочетания «красное яблоко» и «зеленое яблоко».

Символы круглых скобок «(» и «)». Эти символы позволяют получить из искомой строки дополнительную информацию. Обычно, если парсер регулярных выражений ищет в тексте информацию по заданному выражению и находит ее, он просто возвращает найденную строку. Однако если он встречает внутри регулярного выражения круглые скобки, то он рассматривает содержимое этих скобок как еще одно регулярное выражение, по которому необходимо произвести поиск. Парсер рекурсивно вызывает сам себя для поиска по новому регулярному выражению и использует результаты поиска для дальнейшей обработки основного регулярного вы-

ражения. При этом, если поиск хотя бы по одному из внутренних регулярных выражений не увенчался успехом, поиск по всему регулярному выражению считается безуспешным.

Приведем пример, когда получение результатов внутренних регулярных выражений может быть полезным. Допустим, нам необходимо проверить, является ли строка семизначным телефонным номером с указанием кода города, и получить из нее код города и номер телефона:

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

Некоторые из примененных здесь метасимволов вам еще не известны и будут рассмотрены чуть позднее. Давайте рассмотрим этот regexp подробнее.

Первая круглая скобка здесь теряет свое специальное значение и будет рассматриваться как обычный символ:

/**(**\(\d{3,5}\)\s+(\d{3}-\d{2}-\d{2})/

Далее идет регулярное выражение в скобках (проверка кода города):

/\(**(\d{3,5})**\)\s+(\d{3}-\d{2}-\d{2})/

После этого идет закрывающая круглая скобка, которая также лишена своего специального значения из-за символа обратного слэша, стоящего перед ней:

/\((\d{3,5})**\)**\s+(\d{3}-\d{2}-\d{2})/

Затем идет пропуск пустого места:

/\((\d{3,5})\)**\s+**(\d{3}-\d{2}-\d{2})/

И еще одно регулярное выражение в скобках, которое проверяет номер телефона:

/\((\d{3,5})\)\s+**(\d{3}-\d{2}-\d{2})**/

Как видите, здесь есть 3 регулярных выражения – основное и два внутренних. При этом основное выражение позволяет нам проверить, имеет ли строка необходимый нам формат, а два внутренних выражения позволяют получить соответственно код города и номер телефона. Таким образом, одним регулярным выражением можно решить сразу несколько задач!

Посмотрим, как работает это регулярное выражение. Пусть у нас есть строка: «My phone is (095) 123-45-67». Результатами поиска будут 3 строки: '(095) 123-45-67', '095' и '123-45-67'.

Осталось рассмотреть еще одну группу метасимволов, определяющих количественные показатели – ***quantifiers***. Как вы уже могли заметить ранее, очень часто бывает необходимо указать, что какой-то символ должен повторяться определенное количество раз. Конечно, можно просто указать

его необходимое количество раз непосредственно в строке, но это, естественно, не выход. Тем более что очень часто встречаются ситуации, когда точное количество символов неизвестно. Поэтому синтаксис регулярных выражений содержит набор метасимволов, предназначенных именно для решения подобных задач. Каждый из описанных ниже метасимволов определяет количественную характеристику символа, который находится **непосредственно** перед ним.

Звездочка «*». Указывает, что символ должен быть повторен 0 или более раз (т. е. символ может отсутствовать или присутствовать в любых количествах). Пример: выражение /ab*c/ найдет строки ac, abc, abbc и т. д.

Плюс «+». Указывает, что символ должен быть повторен 1 или более раз (т. е. символ обязан присутствовать и может присутствовать в любых количествах). Пример: выражение /ab+c/ найдет строки abc, abbc, abbbc и т. д., но не найдет строку ac.

Знак вопроса «?». Указывает, что символ может как присутствовать, так и отсутствовать, но при этом не может повторяться более одного раза. Пример: выражение /ab?c/ найдет строки ac и abc, но не найдет строку abbc.

Фигурные скобки «{»и «}». Определяют количественную характеристику символа. Внутри скобок через запятую перечисляются минимальное и максимальное количество повторений символа. При этом любой из параметров может быть опущен, а кроме того, можно задать точное количество повторений, указав только одно число. Примеры:

{2,4} – символ должен повториться минимум 2 раза, но не более 4;

{,5} – символ может отсутствовать (так как не задано минимальное количество повторений), но если присутствует, то не должен повторяться более 5 раз;

{3,} – символ должен повторяться минимум 3 раза, но может быть, и больше;

{4} – символ должен повторяться ровно 4 раза.

Есть еще одна тонкость в использовании метасимвола «?». Посмотрите на такое выражение: /.+a/. Ожидается, что оно вернет часть текста до первого вхождения символа «a» в этот текст. На самом деле оно будет работать несколько не так, как ожидается, и результатом поиска будет весь текст до **последнего** вхождения символа «a». Дело в том, что по умолчанию количественные метасимволы «жадничают» и пытаются захватить как можно больший кусок текста. Если это не нужно (как в нашем случае), то необходимо «отучить» их от жадности, указав знак «?» после количественного метасимвола: /.+?a/ . После этого выражение будет работать правильно.

Как уже было сказано ранее, механизм регулярных выражений позволяет добавлять модификаторы, влияющие на обработку регулярного выражения. В табл. 5 рассмотрены наиболее употребительные модификаторы.

Таблица 5

Модификаторы

Модификатор	Значение
i	Включение режима case-insensitive, т. е. большие и маленькие буквы в выражении не различаются
m	Указывает на то, что текст, по которому ведется поиск, должен рассматриваться как состоящий из нескольких строк. По умолчанию механизм регулярных выражений рассматривает текст как одну строку вне зависимости от того, чем она является на самом деле. Соответственно метасимволы ' ^ ' и ' $ ' указывают на начало и конец всего текста. Если же этот модификатор указан, то они будут указывать соответственно на начало и конец каждой строки текста
s	По умолчанию метасимвол «.» не включает в свое определение символ перевода строки, т. е. для многострочного текста выражение /.+/ вернет только первую строку, а не весь текст, как ожидается. Указание этого модификатора снимает это ограничение
U	Делает все количественные метасимволы «нежадными» по умолчанию

На этом мы заканчиваем рассмотрение синтаксиса регулярных выражений, но тема еще не закончена. В следующем выпуске мы рассмотрим вопросы того, как PHP работает с регулярными выражениями, а также рассмотрим некоторые практические примеры использования регулярных выражений в ваших программах.

Функции, которые поддерживают регулярные выражения:

- **ereg** – парное значение регулярного выражения:

```
int ereg(string pattern, string string, array [regs]);
```

Функция ищет в строке string шаблон, заданный в регулярном выражении, указанном в pattern. Если шаблон присутствует в строке, тогда функция возвращает значение, отличное от нуля. В третий параметр можно записывать части строки, соответствующие шаблону. Поиск чуствителен к регистру;

- **ereg_replace** – заменяет регулярное выражение:

```
string ereg_replace(string pattern, string replacement, string string);
```

Функция ищет по шаблону pattern в строке string, затем заменяет найденный текст на replacement и возвращает либо измененную строку в случае удачи, либо исходную строку. Поиск чувствителен к регистру;

• **eregi_replace** – заменяет регулярное выражение без учета регистра:

string eregi_replace(string pattern, string replacement, string string);

Эта функция идентична ereg_replace() за исключением того, что она игнорирует различие в регистре у букв;

• **split** – разбивает строку на массив:

array split(string pattern, string string, int [limit]);

Функция возвращает массив строк (каждая из которых является подстрокой строки), образованный разбитием этой строки на части, отделенные друг от друга разделителем, определенным в параметре pattern. Третий параметр определяет количество элементов в массиве. Если произойдет ошибка, функция вернет false.

В php-версии 4.0 и выше вместо описанных метасимволов используются следующие:

[[:alpha:]] – любая буква;
[[:digit:]] – любая цифра;
[[:alnumt:]] – любая буква или цифра;
[[:space:]] – любой пробельный символ;
[[:upper:]] – любая заглавная буква;
[[:lower:]] – любая маленькая буква;
[[:punct:]] – любой знак пунктуации;
[[:xdigit:]] – любая шестнадцатеричная буква.

11. COOKIES

Cookies были разработаны для решения проблемы сохранения состояния между последовательными посещениями сервера клиентами. Cookies позволяют веб-серверам записывать некоторые данные на жесткий диск каждого клиента и затем при последующих посещениях извлекать их.

Cookie – маленький текстовый файл, который содержит записи, сделанные сервером, и хранится у клиента в папке Cookies определенное время.

Существуют определенные ограничения, накладываемые на использование cookies. Во-первых, браузер не может иметь более 300 cookies и более 20 cookies на один сервер. Во-вторых, cookies посылаются только тем серверам, которым разрешено их получать.

В cookies задаются следующие параметры:

− *время хранения*. По умолчанию cookies существуют до момента закрытия браузера;

− *информация о пути.* Указывает каталоги на сервере, для которых действует cookie. По умолчанию указывает все каталоги сервера;

− *информация о домене.* В cookie можно настроить имена доменов, для которых они будут посылаться. По умолчанию устанавливается домен сервера, установившего cookie;

− *параметр защиты.* Cookies могут посылаться как по защищенным, так и по открытым каналам. По умолчанию – по открытому каналу, но если параметр защиты включен, будут отправляться по защищенным каналам (т. е. через протокол HTTPS).

Приведем пример использования Cookies на конкретном примере. Предположим, нам нужно написать счетчик посещения сайта каждым конкретным посетителем. Данную задачу можно решить двумя способами. Первый из них заключается в ведении учета IP-адресов пользователей. Для этого нужна база данных всего из одной таблицы, примерная структура которой такая:

IP-адрес	*Число посещений*
110.117.234.203	4
212.201.268.207	22
83.103.203.73	8

Когда пользователь заходит на сайт, нам нужно определить его IP-адрес, найти в базе данных информацию о его посещениях, увеличить счетчик и вывести его в браузер посетителя. Написать обработчик (скрипт) подобной процедуры несложно. Однако при использовании такого метода у нас появляются проблемы следующего характера:

•для каждого IP-адреса нужно вести учет в одной таблице, которая может быть очень большой, а из этого следует, что мы нерационально используем процессорное время и дисковое пространство;

•у большинства домашних пользователей IP-адреса являются динамическими, т. е. сегодня у пользователя адрес 212.218.78.124, а завтра – 212.218.78.137. Таким образом, велика вероятность идентифицировать одного пользователя несколько раз.

Можно использовать второй способ, который намного легче в реализации и более эффективен. Мы устанавливаем в Cookie переменную, которая будет храниться на диске удаленного пользователя. Эта переменная и будет хранить информацию о посещениях. Она будет считываться скриптом при обращении посетителя к серверу. Выгода такого метода идентификации очевидна. Во-первых, нам не нужно хранить множество ненужной информации о IP-адресах. Во-вторых, нас не интересуют динамические IP-адреса, поскольку данные о своих посещениях хранятся конкретно у каждого посетителя сайта.

Чтобы установить cookie, используется функция *setcookie*:

int setcookie (string name, string [value], int [expire], string [path], string [domain], int [secure]),

где первый параметр – имя cookie; *value* – значение, соответствующее этому cookie; третий параметр – время жизни cookie (устанавливается при помощи функций *time()* либо *mktime()*); четвертый параметр – информация о пути; *domain* – имя домена; *secure* – параметр защиты (если установлен в 1, cookie посылается по защищенному каналу).

Необходимо помнить, что функция *setcookie* должна находиться в самом начале страницы, так как любое содержание, предшествующее тегу PHP, приводит к возникновению ошибки при работе с cookies.

Пример 27. Cookies

```
<?php
   $cook++;
   setcookie("cook", $cook, mktime(18,0,0,10,31,2002));
   //истекает 31 ноября 2002 года в 18:00:00
   echo "Hello! You were here ".$cook;
?>
```

Пример 27 определяет, сколько раз пользователь посещал данный сайт. Важно помнить, что cookie и соответствующая переменная (в нашем случае «cook» и $cook) доступны только в том случае, если клиент принимает и возвращает cookie обратно серверу.

Доступ к cookie происходит через глобальную переменную с таким же именем, как и у cookie. Но это не единственный способ доступа к cookie.

Другой способ – это доступ через глобальный массив $HTTP_COOKIE_VARS, который хранит только переменные, связанные с cookies.

```
echo $HTTP_COOKIE_VARS["cook"];
```

Кроме того, у вас есть возможность в одном cookie сохранить несколько значений. Для этого cookie рассматривается как массив, и всем элементам этого массива присваиваются значения.

Пример 28. Cookies в виде массива

```
$name = "Vasya";
if(!isset($ArrCook[0])){
    setcookie("ArrCook[0]", $name);
}
if(!isset($ArrCook[1])){
    $ArrCook[1] = 0;
}
$ArrCook[1]++;
setcookie("ArrCook[1]", $ArrCook[1]);

echo "Hello, ".$ArrCook[0]." you were here ".$ArrCook[1]." times!";
```

12. СЕССИИ

Сессии и cookies предназначены для хранения сведений о пользователях при переходах между несколькими страницами. При использовании сессий данные сохраняются во временных файлах на сервере. Файлы с cookies хранятся на компьютере пользователя, и по запросу отсылаются браузером серверу.

Использование сессий и cookies очень удобно и оправдано в таких приложениях, как интернет-магазины, форумы, доски объявлений, когда, во-первых, необходимо сохранять информацию о пользователях на протяжении нескольких станиц, а во-вторых, своевременно предоставлять пользователю новую информацию.

Протокол HTTP является протоколом «без сохранения состояния». Это означает, что данный протокол не имеет встроенного способа сохранения состояния между двумя транзакциями, т. е. когда пользователь открывает сначала одну страницу сайта, а затем переходит на другую страницу этого же сайта, то, основываясь только на средствах, предоставляемых протоколом HTTP, невозможно установить, что оба запроса относятся к одному пользователю. Таким образом, необходим метод, при помощи которого можно было бы отслеживать информацию о пользователе в течение одного сеанса связи с Web-сайтов. Одним из таких методов является управление сеансами при помощи предназначенных для этого функций. Для нас важно то, что сеанс, по сути, представляет собой группу переменных, которые, в отличие от обычных переменных, сохраняются и после завершения выполнения PHP-сценария.

При работе с сессиями различают следующие этапы:

- открытие сессии;
- регистрация переменных сессии и их использование;
- закрытие сессии.

Самый простой способ открытия сессии заключается в использовании функции session_start(), которая вызывается в начале PHP-сценария. Эта функция проверяет, существует ли идентификатор сессии, и если нет, то создает его. Если идентификатор текущей сессии уже существует, то загружаются зарегистрированные переменные сессии. После инициализации сессии появляется возможность сохранять информацию в суперглобальном массиве $_SESSION.

Пример 29. Организация сессии (сеанса)

Пусть имеется файл index.php, в котором в массив $_SESSION сохраняется переменная и массив.

index.php

```
<?php
// Инициируем сессию
 session_start();
// Помещаем значение в сессию
 $_SESSION['name'] = "value";
    // Помещаем массив в сессию
 $arr = array("first", "second", "third");
 $_SESSION['arr'] = $arr;
// Выводим ссылку на другую страницу
 echo "<a href='other.php'>другая страница</a>";
?>
```

На страницах, где происходит вызов функции **session_start()**, значения данных переменных можно извлечь из суперглобального массива **$_SESSION**. В следующем листинге приводится содержимое страницы other.php, где извлекаются данные, ранее помещенные на странице index.php.

other.php

```
<?php
  // Инициируем сессию
  session_start();
  // Выводим содержимое суперглобального массива $_SESSION
  echo "<pre>";
  print_r($_SESSION);
  echo "</pre>";
?>
```

После завершения работы с сессией сначала нужно разрегистрировать все переменные сессии, а затем вызвать функцию unset($_SESSION [«username»]).

Рассмотрим пример простой сессии, работающей с тремя страницами. При посещении пользователем первой страницы открывается сессия и регистрируется переменная **$username**. Соответствующий код реализации приведен в примере 30.

Пример 30

```
<? php
  session_start();
  $_SESSION['username'] = "maksim";
  echo 'Привет, '.$_SESSION['username']."<br>";
?>

 <a href="page2.php">На следующую страницу </a>
```

Результат работы этого сценария показан на рис. 4.

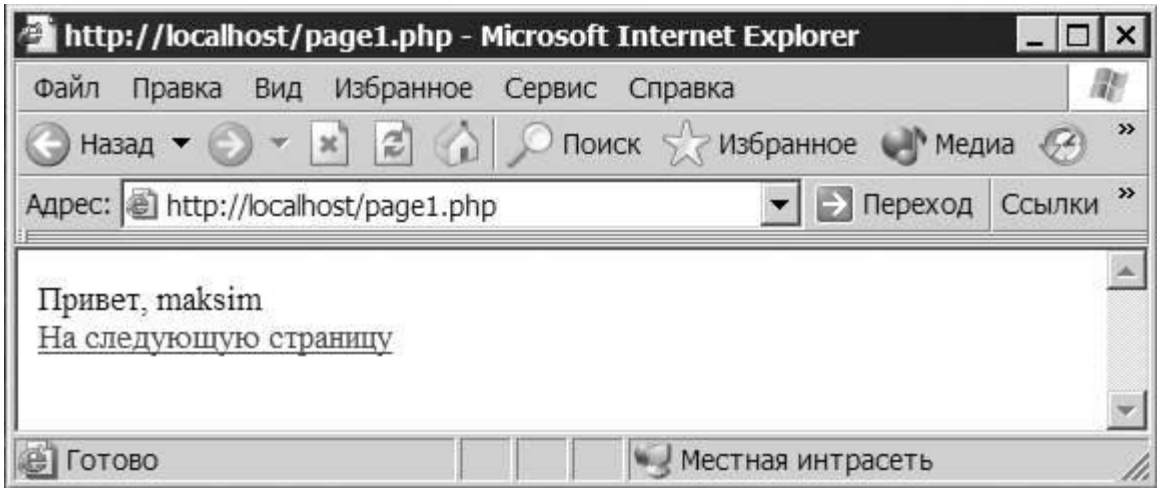

Рис. 4. Результат работы сценария примера 30

После этого пользователь maksim нажимает на ссылку и попадает на страницу page2.php, код которой приведен в примере 30.

Пример 31

```
<?php
  session_start();
  echo $_SESSION['username'].', ты пришел на другую страницу этого сайта!';
  echo("<br>");
?>

<a href="page3.php">На следующую страницу </a>
```

Результат работы этого скрипта показан на рис. 5.

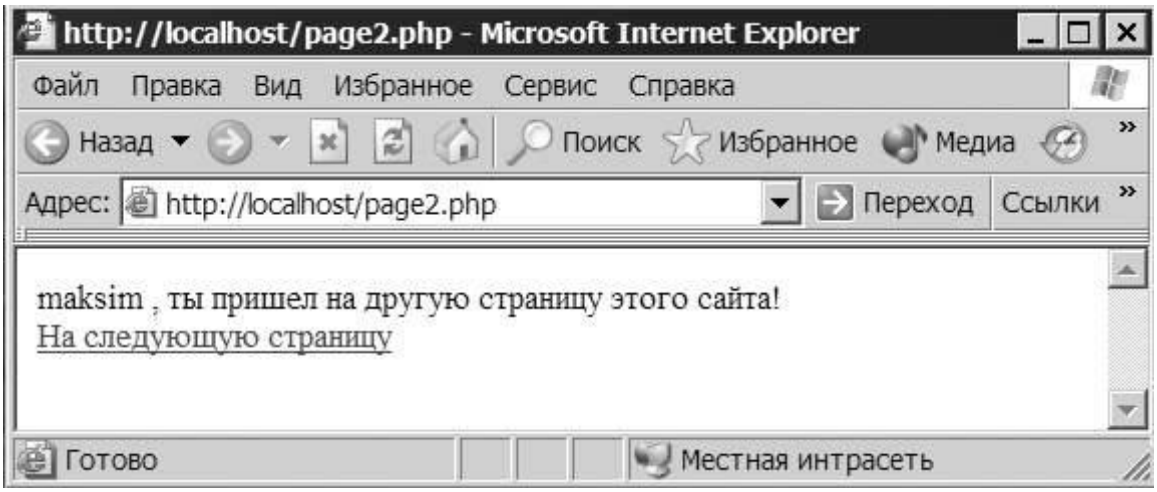

Рис. 5. Результат работы сценария примера 31

При нажатии на ссылку, пользователь попадает на страницу page3.php, при этом происходит разрегистрация сеансовой переменной и уничтожение сессии. Соответствующий код реализации приведен в листинге (пример 32).

Пример 32

```
<?php
  session_start();
  unset($_SESSION['username']); // разрегистрировали переменную
  echo 'Привет, '.$_SESSION['username'];
  /* теперь имя пользователя уже не выводится */
  session_destroy(); // разрушаем сессию
?>
```

Как видно из рис. 6, после разрегистрации сеансовой переменной значение массива **$_SESSION**['username'] уже недоступно.

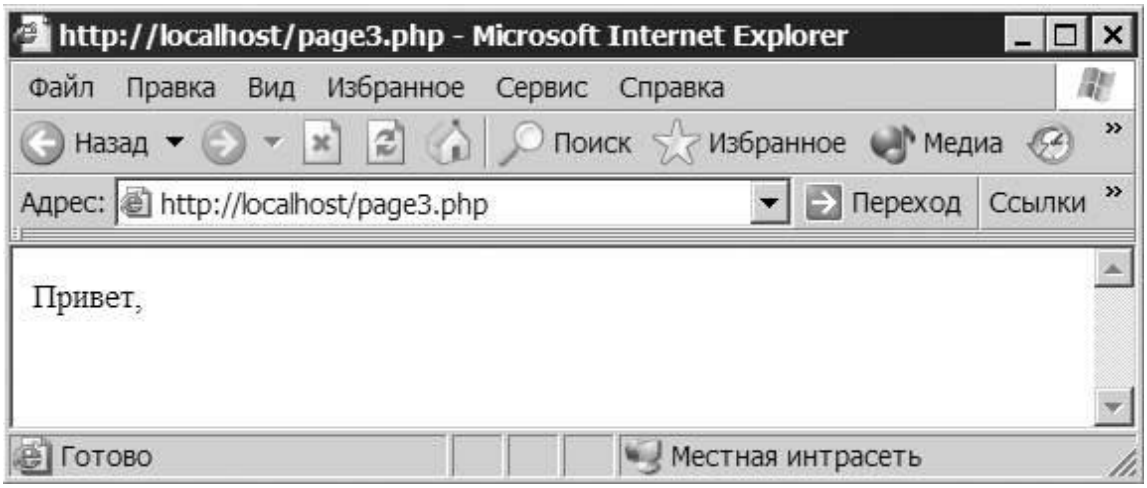

Рис. 6. Результат работы сценария примера 32

Задачи

Задача 24. Опишите форму для загрузки файлов. Используйте ее для загрузки. Если полученный файл не превышает 1 Мбайт и не было ошибок при передаче, создайте папку NewDir и поместите в нее полученный файл.

Задача 25. Проверьте, может ли заданная строка являться адресом электронной почты, расположенной на сайте www.mail.ru. Выведите соответствующее сообщение в браузер.

Задача 26. Дан текстовый файл и «слово». Выделите жирным слова в текстовом файле, которые содержат в себе «слово». Например, дано слово «мастер» и файл с текстом, где это слово выделено: «MasterWebs – Форум веб-**мастер**ов». Текстовый файл может выбираться произвольно, выделяемое слово вводится через форму.

Задача 27. Палиндромом называют последовательность символов, которая читается как слева направо, так и справа налево. Найдите во введенной строке подстроку-палиндром максимальной длины.

Задача 28. Осуществите задачу перевода числа из одной системы счисления (СС) в другую. Есть пользовательская форма с тремя полями (число, из какой СС, в какую СС). Пользователь заполняет все поля и отправляет данные на сервер. На экране должно появиться сообщение вида:

Старая система счисления – [СС]	Новая система счисления – [СС]
Число – [Число в старой СС]	Число – [Число в новой СС]

Задача 29. Создайте в сессии массив для хранения всех посещенных страниц и сохраните в качестве его очередного элемента путь к текущей странице. Выведите в цикле список всех посещенных пользователем страниц. Примечание: для решения задачи воспользуйтесь материалами из [2].

Задача 30. Инициализируйте переменную для подсчета количества посещений. Если соответствующие данные передавались через cookie, сохраняйте их в переменную. Нарастите счетчик посещений. Инициализируйте переменную для хранения значения последнего посещения страницы. Если соответствующие данные передавались из cookie, отфильтруйте их и сохраните в эту переменную. Установите соответствующие cookie. Примечание: для решения задачи воспользуйтесь материалами из [2].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://php-book.ru/
2. http://www.php.su/php/
3. http://www.magnoli.ru/zadachi.php
4. http://www.softtime.ru/bookphp/gl1_1.php
5. http://ru.html.net/tutorials/php/
6. http://php.net/manual/ru/tutorial.php
7. http://codebook.info/uchebnik_phpmysql/protocol-http.php
8. http://w-wb.com/cgi/cgi_05_01.php

Филиппов Феликс Васильевич
Губин Александр Николаевич

HTTP + PHP
В ПРИМЕРАХ И ЗАДАЧАХ

Учебное пособие

Редактор *И. И. Щенсняк*

Компьютерная верстка *Н. А. Ефремовой*

План издания 2015 г., доп. п. 2

Подписано к печати 10.12.2015
Объем 4,25 усл.-печ. л. Тираж 30 экз. Заказ 608

Редакционно-издательский отдел СПбГУТ
191186 СПб., наб. р. Мойки, 61

Отпечатано в СПбГУТ