ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Для выполнения каждой лабораторной работы выполните следующие действия.

1. Внимательно и полностью прочтите не только задание, но и весь текст лабораторной работы. Повторите соответствующие ее теме материалы лекций. При необходимости изучите дополнительную литературу.

2. При необходимости разработайте и создайте требуемую для выполнения задания базу данных:
   2.1. разработайте модель БД согласно полученному заданию;
   2.2. подключитесь к серверу БД с помощью панели администратора *phpMyAdmin* или ее аналога;
   2.3. создайте в БД необходимые таблицы;
   2.4. проверьте работоспособность таблиц, добавив в них несколько записей;
   2.5. убедитесь, что Вам известны адрес сервера БД, имя базы данных, имя пользователя и его пароль для доступа к БД. Эти данные необходимо использовать в PHP для подключения к серверу БД.

3. Создайте на сервере отдельную папку для файлов лабораторной работы. Например, для лабораторной работы № B-1 это может быть "lab-b1". Убедитесь, что созданная папка находится в рабочей папке на сервере, т.е. сервер предоставляет доступ к файлам вашей папки через протокол *http*.

4. Скопируйте в созданную папку файлы шаблона (при его наличии). Убедитесь, что шаблон нормально и корректно открывается в разных браузерах.

5. Внесите в статический файл шаблона и файл его CSS-стилей необходимые изменения или дополнения.

6. Убедитесь в работоспособности статического сайта в различных браузерах.

7. Внедрите в статические файлы приведенный в описании лабораторной работы PHP-код. Если необходимо доработайте его так, чтобы он успешно выполнял реализуемые в нем функции: напишите недостающие функции, скорректируйте алгоритмы и т.д. Не забудьте поменять расширение статических файлов с *PHP*-кодом на "*.php*". При необходимости создайте новые файлы с *PHP*-кодом.

8. Проверьте работоспособность реализованного функционала. При необходимости поправьте код страниц так, чтобы реализованная часть функций работала нормально. Полностью уясните работу *PHP*-программы и принципы ее разработки на данном этапе.

9. Самостоятельно доработайте *PHP*-код таким образом, чтобы он полностью соответствовал всем требованиям лабораторной работы.

10. Проверьте и протестируйте работоспособность сайта. При необходимости поправьте код страниц. Убедитесь, что результат точно соответствует описанному в задании.

11. Подготовьте ответы на контрольные вопросы.

12. Сдайте и защитите лабораторную работу.

ДЛЯ УСПЕШНОГО И СВОЕВРЕМЕННОГО ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ КРАЙНЕ ВАЖНА ПОДГОТОВКА К НЕЙ. ПОМНИТЕ, ЧТО НА САМОСТОЯТЕЛЬНУЮ РАБОТУ ВЫДЕЛЯЕТСЯ СТОЛЬКО ЖЕ ВРЕМЕНИ, КАК И НЕПОСРЕДСТВЕННО НА РАБОТУ В АУДИТОРИЯХ. ИСХОДЯ ИЗ ЭТОГО, ВРЕМЯ РАБОТЫ СТУДЕНТОВ РАССЧИТАНО ТАКИМ ОБРАЗОМ, ЧТО ЕГО ВПОЛНЕ ДОСТАТОЧНО ДЛЯ УСПЕШНОГО ВЫПОЛНЕНИЯ ВСЕГО КУРСА ПРИ УСЛОВИИ ДОМАШНЕЙ ПОДГОТОВКИ В ВИДЕ ИЗУЧЕНИЯ ТЕОРИИ, УЯСНЕНИЯ ЗАДАНИЯ, РАЗБОРА РЕКОМЕНДАЦИЙ К ВЫПОЛНЕНИЮ И Т.Д.

Простейшая программа на PHP.
Конвертация статического контента в динамический.

Лабораторная работа № А-1.

ЦЕЛЬ РАБОТЫ

Ознакомление с основами языка программирования *PHP*, его назначением и возможностями, спектра решаемых задач, ограничениями. Получение навыков работы со средой программирования, обучение работы с FTP-сервером.

Основная задача языка *PHP* – динамически формировать *HTML*-код страницы или содержимое документов другого типа. При этом в *PHP* нет команд, которые бы добавляли на страницу таблицы, заголовки, блоки, формы или любые другие теги. Нет, все что может сделать с помощью PHP программист для формирования страниц сайта – это вывести в HTML-код строку (текст), которая уже потом будет интерпретироваться браузером как HTML-документ. И если эта строка содержит какие-либо HTML-теги, они будут обработаны браузером. Практически, программный код выступает в роли HTML-верстальщика, который динамически, непосредственно перед загрузкой, верстает страницы сайта.

Поэтому крайне важно знать и уметь самостоятельно работать со статическим HTML-кодом. Если Вы не знаете, что именно необходимо вывести в качестве правильного HTML-кода средствами PHP, то даже зная в совершенстве теорию алгоритмов, сам язык и имея многолетнюю практику прикладного программирования – результат Вашей работы, а именно динамически формируемый HTML-код, будет некачественным и ошибочным.

Самое простое что можно сделать на PHP – это преобразовать сразу весь HTML-код в динамический. Действительно, если создать переменную и записать в нее сразу весь код – то вывод этой переменной создаст требуемую динамическую страницу. Конечно, такое действие вряд ли имеет смысл, но формирование 100% кода средствами PHP – нормальный и часто применяемый подход, например, при программировании CMS. Но на данном этапе этого не требуется, сначала необходимо научиться работать с PHP, использовать его для небольших задач, формировать HTML-код лишь частично.

Итак, смысл данной лабораторной работы – в замене статического HTML-кода на PHP-программу, которая формирует этот же код в виде текста и выводит его. Поэтому, во время работы над заданием, просто ищите в статической верстке те фрагменты, которые следует заменить. Смело внедряйте вместо них PHP-программы, которые выводят эти фрагменты динамически. В случае дополнительных заданий – выводите не только формируемый контент, но и дополнительный текст согласно условиям лабораторной работы.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа (2 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу http документы (страницы сайта) с частично динамически формирующимся контентом.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Тема статического сайта выбирается студентом самостоятельно, информация – копируется из открытых источников или формируется самостоятельно. Дизайн должен быть лаконичным, цвета – сочетаемыми, размер шрифта (кегль) – читаемым, но не излишне громоздким.

Статический сайт	
Страниц	Три, макет страниц одинаковый.
Меню	В виде ссылок в шапке страницы. Ссылка на текущую страницу выделена цветом шрифта или фона с помощью стиля CSS. Ссылки реагируют на курсор мыши.
Заголовок страницы (title)	ФИО и группа студента, номер и название лабораторной работы.
Шапка (header)	Фиксированной высоты, приклеена к верху страницы и не реагирует на скроллинг. Темно-зеленый цвет фона, буквы белые.
Подвал (footer)	Фиксированной высоты, приклеен к низу страницы и не реагирует на скроллинг. Темно-серый цвет фона, буквы светло-серые.
Заголовки	Один H1, не менее двух H2.
Текст	Не менее 1Кб на страницу, не менее одной таблицы с двумя строками и тремя колоками на страницу.
Фотографии	Не менее двух на странице.

После внедрения в статический HTML-код элементов PHP-скриптов (частичного конвертирования контента в динамический вид) сайт должен вести себя следующим образом.

Динамический сайт	
Название страницы (TITLE)	Название страницы сохраняется в переменной, которая затем выводится с помощью PHP.
Подвал	В подвале страницы размещается надпись: "*Сформировано 15.02.2016 в 12-57:18*" с актуальной датой и временем.
Таблицы	HTML-код первой строки таблицы (включая теги `<tr>...</tr>`) полностью выводится средствами PHP. В HTML-коде второй строки таблицы динамически формируется только содержание ячеек (между тегами `<td>...</td>`, не включая сами теги). Для вывода данных промежуточное сохранение строк переменные не используется.
Меню	Адрес, текст ссылок и класс пункта меню формируется в виде **ДВУХ** включений PHP-кода. При этом они должны быть одинаковы для всех пунктов на каждой странице, за исключением присваиваемым переменным значений.
Фотографии	В зависимости от секунды (четная или нечетная) на одно и тоже место загружаются разные фотографии (разные имена файлов).

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Первая лабораторная работа носит ознакомительный характер, поэтому ее содержание достаточно просто. Замена заголовка или других фрагментов HTML-кода вряд ли вызовет затруднение, формирование строки с текущей датой – тоже. Могут вызывать затруднения два вопроса – это загрузка разных фотографий в зависимости от секунды и вывод пункта меню. Рассмотрим первую задачу более подробно.

Пусть имеется две фотографии одинакового размера: "*fotos/foto1.jpg*" "*fotos/foto2.jpg*". Тогда словесное описание алгоритма можно сформулировать следующим образом.

1. Определяем текущую секунду и сохраняем ее в переменной `$s`.
2. Вычисляем остаток от деления переменной `$s` на число 2 и сохраняем его в переменной `$os`.
3. Если в переменной `$os` хранится ноль – в переменную $name записываем строку "*fotos/foto1.jpg*".
4. Иначе в переменную `$name` записываем строку "*fotos/foto2.jpg*".
5. Формируем строку, содержащую тег `<img>` с вычисленным именем файла и другими необходимыми параметрами и выводим ее в HTML-код браузера.

Предварительно формулировать реализуемый алгоритм на естественном языке очень важно. Любая программа выполняется именно так, как она написана; делает только то, что ей указали делать, а не то что хочется. Поэтому, если вы не понимаете логику работы программы, не можете выразить ее алгоритм простым человеческим языком – добиться ее правильного функционирования будет очень тяжело. Конечно, со временем и с опытом письменное выражения алгоритма для вас не будет необходимостью. И это будет ознаменовать новый этап развития вас как разработчика программ – язык программирования PHP станет для вас естественным. Вы научитесь выражать на нем свои мысли и читать его также просто, как сейчас читаете эти слова. Но пока этого не произошло, формулируйте алгоритм словами, а лишь затем переводите его в программный код. Фактически, это действительно перевод с одного языка, в данном случае русского, на PHP. Сделаем это прямо сейчас!

Листинг А-1. 1

```
$s = date('s');            // определяем текущую секунду
$os = $s % 2;         // вычисляем остаток от деления

if( $os === 0 )            // если в переменной $os хранится ноль
     $name='fotos/foto1.jpg';   // сохраняем имя первого файла
else                            // иначе
     $name='fotos/foto2.jpg';   // сохраняем имя второго файла

echo '<img src="'.$name.'" alt="Меняющаяся фотография">'; // выводим сформированный HTML-код
```

Ура, мы сделали это! Заметьте – при правильном переводе в программе пункты нашего словесного описания преобразуются в комментарии. Но, честно говоря, данный код хоть и решает задачу, но далек от идеала. Оптимизируем его, убрав переменные, которые используются только один раз, а также по-другому формируя HTML-код.

Листинг А-1. 2

```
if( date('s') % 2 === 0 )  // если секунда четная
     $name='1';            // имя файла содержит "1"
else                       // иначе
     $name='2';            // имя файла содержит "2"

// формируем и выводим HTML-код
echo '<img src="fotos/foto'.$name.'.jpg" alt="Меняющаяся фотография">';
```

Результат работы программы будет абсолютно одинаков с первым примером, но смотрится он более красиво. Не так ли? Для примера сформируем и третий вариант кода, который также может иллюстрировать возможности PHP и наличие у одной задачи разных алгоритмов решения.

Листинг А-1. 3

```
echo '<img src="fotos/foto';    // выводим первую часть формируемого HTML-кода

echo date('s') % 2 + 1;         // вычисляем переменную часть имени и выводим ее

echo '.jpg" alt="Меняющаяся фотография">';      // выводим вторую часть HTML-кода
```

Или, если сократить запись еще больше, можно просто написать такой код.

Листинг А-1. 4

```
// формируем HTML-код и сразу его выводим
echo '<img src="fotos/foto'.(date('s') % 2+1).'.jpg" alt="Меняющаяся фотография">';
```

Попробуйте все варианты кода, убедитесь, что результат их работы одинаков. Разберитесь и убедитесь, что вы понимаете каждый СИМВОЛ программы. Если вы не сделаете этого – разбирать более сложные примеры будет просто невозможно. Ведь Вы не умеете "говорить" на PHP, не

знаете его слов, не сможете перевести естественно-языковое описание алгоритм на язык программирования.

Аналогично поступим и со второй задачей. Путь в HTML-коде пункт меню формируется следующим тегом: "Вторая страница". Ясно, что различные пункты меню отличаются адресом, в данном случае "page2.php", и текстом ссылки. Стиль "selected_menu" будет определять выделенный пункт меню (ссылка без класса – обычный пункт меню). Тогда, для вывода пункта меню можно предложить такую программу.

Листинг А-1. 5

```
<?php      // начинаем PHP скрипт
             // формируем и выводим строку с ссылкой
       echo '<a href="page2.php" class="selected_menu">Вторая страница</a>';
?>
```

Сейчас мы просто выводим строку с кодом ссылки с помощью PHP. Но в задании лабораторной работы необходимо делать это с помощью двух фрагментов кода. Нет ничего проще!

Листинг А-1. 6

```
<?php      // начинаем первый PHP скрипт
       echo '<a href="page2.php"';
?>
<?php      // начинаем второй PHP скрипт
       echo ' class="selected_menu">Вторая страница</a>';
?>
```

Обратите внимание: между окончанием первого фрагмента PHP-программы и началом второго в данном примере нет ни пробелов, ни переводов строки. Если они будут, то и в HTML-коде ссылки появится пробел или перенос строки, что будет совершенно лишним. Продолжим приближать наш код к заданию лабораторной работы: сделаем так, что PHP выводит ТОЛЬКО адрес и текст ссылки, а также ее класс.

Листинг А-1. 7

```
<a href="<?php       // начинаем первый PHP скрипт
       echo 'page2.php';
?>" class="<?php                 // начинаем второй PHP скрипт
       echo 'selected_menu">Вторая страница';
?></a>
```

Формирование HTML-кода ссылки будет происходить следующим образом (для краткости уберем комментарии и необязательные переносы строк).

Шаг	**1**
Анализируемый фрагмент кода PHP-страницы	`<a href="`
Результат	`<a href="`
Сформированный HTML-код	`<a href="`

Первая часть исходного кода страницы – статическая. Соответственно PHP-его не обрабатывает, и она передается в браузер без изменений.

Шаг	**2**
Анализируемый фрагмент кода PHP-страницы	`<?php echo 'page2.php'; ?>`
Результат	`page2.php`

Сформированный HTML-код	`<a href="page2.php`

На втором шаге встречается PHP-скрипт, при выполнении которого выводится адрес ссылки.

Шаг	**3**
Анализируемый фрагмент кода PHP-страницы	`" class="`
Результат	`" class="`
Сформированный HTML-код	`<a href="page2.php" class="`

Третий фрагмент – опять статический HTML-код. Обратите внимание: он ежит между двумя скриптами, и если в прошлый раз мы специально оговаривали что меду ними ничего не должно быть, чтобы структура тега ссылки не нарушалась, то сейчас мы намеренно добавили этот фрагмент.

Шаг	**4**
Анализируемый фрагмент кода PHP-страницы	`<?php echo 'selected_menu">Вторая страница'; ?>`
Результат	`selected_menu">Вторая страница`
Сформированный HTML-код	`<a href="page2.php" class="selected_menu">Вторая страница`

Четвертый фрагмент – снова PHP скрипт. Он формирует продолжение тега ссылки с классом и текстом ссылки.

Шаг	**5**
Анализируемый фрагмент кода PHP-страницы	`</a>`
Результат	`</a>`
Сформированный HTML-код	`<a href="page2.php" class="selected_menu">Вторая страница</a>`

Заключительная часть фрагмента – снова статическая, она завершает формирование HTML-кода ссылки.

Такой последовательный разбор действия, с последовательной записью формируемого HTML-кода – хороший способ отладки работающего PHP-скрипта, когда он формирует "странный" HTML-код. Но вернемся к тексту программы: он должен быть универсален для всех пунктов меню на всех страницах. Ясно, что сейчас это далеко не так.

Листинг А-1. 8

```
<a href="<?php       // начинаем первый PHP скрипт

      $name='Вторая страница'; // переменная с текстом ссылки
      $link='page2.php';       // переменная с адресом ссылки
      $current_page=true;      // переменная, определяющая активность пункта меню

      echo $link;              // выводим адрес ссылки

?>"><?php            // начинаем второй PHP скрипт

     if( $current_page )       // если пункт меню активный
          echo ' class="selected_menu"';  // выводим соответствующий класс
```

```
echo $name;                    // выводим текст ссылки
?></a>
```

Меняя значения переменных в начале первого фрагмента кода можно формировать ссылку с необходимым адресом, текстом и оформлением.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

<table>
<tr><td>Оформление начала и окончания PHP-кода</td><td><pre><? .... ?></pre></td></tr>
<tr><td>Комментарии</td><td><pre>.... // комментарий до конца строки
/* многострочный комментарий
....
*/</pre></td></tr>
<tr><td>Присвоение переменной строкового значения</td><td><pre>$A='строка';</pre></td></tr>
<tr><td>Функция вывода даты и некоторые ее параметры</td><td><pre>date($f);</pre>
<table>
<tr><th>Символ в шаблоне $f</th><th>Значение</th></tr>
<tr><td>H</td><td>Часы в 24-часовом формате с ведущими нулями</td></tr>
<tr><td>i</td><td>Минуты с ведущими нулями</td></tr>
<tr><td>s</td><td>Секунды с ведущими нулями</td></tr>
<tr><td>d</td><td>День месяца, 2 цифры с ведущими нулями</td></tr>
<tr><td>m</td><td>Порядковый номер месяца с ведущими нулями</td></tr>
<tr><td>Y</td><td>Порядковый номер года, 4 цифры</td></tr>
</table></td></tr>
<tr><td>Вывод значения переменной</td><td><pre>echo $A;</pre></td></tr>
<tr><td>Вывод строки</td><td><pre>echo 'Строка';</pre></td></tr>
<tr><td>Условный оператор</td><td><pre>if(<условие>) // если условие выполняется
    ...   // выполняется этот оператор
else   // иначе
    ...   // выполняется этот оператор</pre></td></tr>
</table>

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Что такое PHP?
2. Какой основное назначение PHP?
3. В чем отличие статического и динамического контента?
4. Как внедрить PHP-код на статическую страницу?
5. Какие общепринятые требования к страницам с PHP кодам?
6. Как трансформировать код вывода ссылки А-1.8 так, чтобы он не содержал статических фрагментов?
7. Как будет выглядеть словесное описание алгоритмов, реализуемых в листингах А-1.5 … А-1.8?

Циклические алгоритмы. Условия в алгоритмах. Табулирование функций.

Лабораторная работа № А-2.

ЦЕЛЬ РАБОТЫ

Изучение структуры и синтаксиса языка PHP, получение базовых навыков построения программ, использования основных операторов. Закрепление знаний использования циклов и условных операторов, как основы построения алгоритмов.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа (2 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу http документ (страница сайта) с результатами вычислений значений математической функции.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Перед написанием программного кода PHP, необходимо подготовить статическую HTML-страницу, которая будет использоваться как шаблон для этой и следующих лабораторных работа. Шаблон должен удовлетворять следующим требованиям.

1. Использование стандарта HTML5 и CSS3. Прохождение страницей валидации сервисом "http://validator.w3.org/" без ошибок и предупреждений.
2. Наличие тегов "header", "main" и "footer".
3. В шапке должен размещаться логотип университета.
4. В названии страницы (TITLE) и в ее шапке должны быть указаны:
    A. ФИО выполнившего работу;
    B. группа;
    C. номер лабораторной работы и номер варианта (если он есть).
5. Подвал (footer) должен быть "приклеен" к низу экрана и не реагировать на скроллинг.
6. Цвета заливки и текста шапки и соответствующие цвета подвала должны отличаться от основного цвета страницы.
7. Стили прописаны в отдельном файле "styles.css".
8. Основной блок страницы должен быть светло-серого фона с шрифтом черного цвета.
9. Страница сайта должна корректно отображаться в любом современном браузере.

Не забудьте скопировать шаблон перед использованием в лабораторной работе – в дальнейшем он потребуется вам в начальном виде.

После создания шаблона и его проверки необходимо внедрить внутри тега "`<main> … </main>`" PHP-код, который бы:

1. в явном виде инициализировал числовые переменные, хранящие начальное значение аргумента функции, количество вычисляемых значений функции, шаг изменения значения аргумента. максимальное и минимальное значение функции, после которых вычисления останавливаются;
2. в явном виде инициализировал строковую переменную, хранящую тип формируемой верстки;
3. вычислял значения функции для соответствующего количества аргументов, начиная от начального значения аргумента с заданным выше шагом;
4. в подвале выводил название типа верстки.
5. в зависимости от заданного типа верстки ('A', 'B', 'C', 'D', 'E') выводил аргументы функции и ее значения в следующем виде.
    A. Простая верстка текстом, без таблиц и блоков. Выводятся строки (разделитель – тег "`<br>`") вида: "*f(x)=y*", где *x* – текущий аргумент, *y* – значении функции от этого аргумента (например, "*f(8)=15.8*").

B. Маркированный список. Каждая строка "*f(x)=y*" выводится как элемент маркированного списка (тег "<ul>" и "<li>").
C. Нумерованный список. Каждая строка "*f(x)=y*" выводится как элемент нумерованного списка (тег "<ol>" и "<li>").
D. Табличная верстка. Выводится таблица (границы ячеек одинарные, толщина 1*px*, черный цвет) в первой колонке которой размещается номер строки таблицы, во второй – значения аргументов, в третьей – значения функций от этих аргументов.
E. Блочная верстка. Каждая строка "*f(x)=y*" выводится внутри блока (тег " "), причем все блоки располагаются по горизонтали (соответствующий режим обтекания текстом), имеют красную рамку толщиной 2*px*, отступ друг от друга на 8*px*.

6. вычислял и выводил максимальное, минимальное, среднее арифметическое всех значений функции, а также их сумму.

ВАРИАНТЫ ЗАДАНИЯ

Задания лабораторной работы одинаковы для всех вариантов, за исключением вычисляемых функций. Номер задания соответствует номеру студента в списке группы (его необходимо уточнить у преподавателя перед началом выполнения лабораторной работы), для 1*x* и 2*x* номеров берется вариант №*x*.

1.	$f(x)=\begin{cases}10\cdot x-5, \text{при } x\le 10\\ (x+3)\cdot x^2, \text{при } x>10 \text{ и } x<20\\ \frac{3}{x-25}+2, \text{при } x\ge 20\end{cases}$
2.	$f(x)=\begin{cases}\frac{10+x}{x}, \text{при } x\le 10\\ (x/7)\cdot(x-2), \text{при } x>10 \text{ и } x<20\\ x\cdot 8+2, \text{при } x\ge 20\end{cases}$
3.	$f(x)=\begin{cases}3\cdot x^3+2, \text{при } x\le 10\\ 5\cdot x+7, \text{при } x>10 \text{ и } x<20\\ \frac{x}{22-x}-x, \text{при } x\ge 20\end{cases}$
4.	$f(x)=\begin{cases}(5-x)/(1-\frac{x}{5}), \text{при } x\le 10\\ x^2/4+7, \text{при } x>10 \text{ и } x<20\\ 2\cdot x-21, \text{при } x\ge 20\end{cases}$
5.	$f(x)=\begin{cases}3\cdot x+9, \text{при } x\le 10\\ \frac{x+3}{x^2-121}, \text{при } x>10 \text{ и } x<20\\ x^2\cdot 4-11, \text{при } x\ge 20\end{cases}$
6.	$f(x)=\begin{cases}x^2\cdot 0.33+4, \text{при } x\le 10\\ 18\cdot x-3, \text{при } x>10 \text{ и } x<20\\ \frac{1}{x*0.1-2}+3, \text{при } x\ge 20\end{cases}$
7.	$f(x)=\begin{cases}\frac{6}{x-5}\cdot x-5, \text{при } x\le 10\\ (x^2-1)\cdot x+7, \text{при } x>10 \text{ и } x<20\\ 4\cdot x+5, \text{при } x\ge 20\end{cases}$
8.	$f(x)=\begin{cases}7\cdot x+18, \text{при } x\le 10\\ \frac{(x-17)}{8-x\cdot 0.5}, \text{при } x>10 \text{ и } x<20\\ (x+4)\cdot(x-7), \text{при } x\ge 20\end{cases}$
9.	$f(x)=\begin{cases}x^2\cdot(x-2)+4, \text{при } x\le 10\\ 11\cdot x-55, \text{при } x>10 \text{ и } x<20\\ \frac{x-100}{100-x}-{}^{x}/_{10}+2, \text{при } x\ge 20\end{cases}$
10.	$f(x)=\begin{cases}\frac{3}{x}+\frac{x}{3}-5, \text{при } x\le 10\\ (x-7)\cdot(x/8), \text{при } x>10 \text{ и } x<20\\ 3\cdot x+2, \text{при } x\ge 20\end{cases}$

Все вычисления значения функции должны производиться по правилам математики с округлением конечного результата до 3-х знаков после запятой. В случае невозможности реализовать вычисление функций для заданной области определения, это должно быть обосновано студеном при защите. При попытке деления на ноль в качестве значения функции выводится строка "*error*".

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Естественно-языковое описание алгоритма PHP-программы достаточно точно пересекается с заданием. Поэтому сразу приступим к разбору программы непосредственно на PHP.

Листинг А-2. 1

```
<?php
	$start_value = -10;		// начальное значение аргумента
	$encounting = 10000;		// количество вычисляемых значений
	$step = 2;			// шаг изменения аргумента
	$type = 'A';			// тип верстки

	$x=$start_value;		// текущее значение аргумента. равно начальному
```

```
        for( $i=0; $i < $encounting; $i++ ) // цикл с заданным количеством итераций
        {
            if( $x <= 10 )        // если аргумент меньше или равен 10
                 $f = 32*x / 21;       // вычисляем функцию
            else                  // иначе
            if( $x <20 )               // если аргумент меньше 20
                 $f = x*x/3 + 7/(x-4);           // вычисляем функцию
            else                       // иначе
                 $f = ( 1 / (x-11) )*2 + x;      // вычисляем функцию

            echo 'f('.$x.')='.$f.'<br>';    // выводим значение функции

            $x += $step;              // вычисляем следующий аргумент
        }
?>
```

Согласно требованиям лабораторной работы, в начале программы мы инициализируем четыре переменных, в которых будут храниться соответствующие значения – про ограничение диапазона значений пока забудем. При изменении этих переменных выводимые данные будут меняться. В следующей строке мы определяем текущее значение аргумента функции, которое берется из соответствующей переменной.

Основа программы – цикл со счетчиком, который совершает заданное число итераций. При этом с помощью условного оператора определяется по какой формуле вычисляется и сохраняется в переменной `$f` значение функции. Вывод осуществляется согласно типу верстки "A" – с построчной разбивкой данных. В конце каждой итерации цикла вычисляется новое значение аргумента: к его старому значению прибавляется шаг инкремент из переменной `$step`.

Рассмотренный скрипт содержит сразу несколько погрешностей, а именно:

- не учитывает тип верстки;
- не оптимален;
- содержит потенциальную ошибку при делении на ноль.

Последовательно улучшим программу так, чтобы она удовлетворяла требованиям лабораторной работы.

Листинг А-2. 2

```
<?php
        $x = -10;      // начальное значение аргумента
        $encounting = 10000;      // количество вычисляемых значений
        $step = 2;                // шаг изменения аргумента
        $type = 'A';              // тип верстки

        // цикл с заданным количеством итераций
        for( $i=0; $i < $encounting; $i++, $x+=$step )
        {
            if( $x <= 10 )        // если аргумент меньше или равен 10
                 $f = 32*x / 21;       // вычисляем функцию
            else                  // иначе
            if( $x <20 )               // если аргумент меньше 20
                 $f = x*x/3 + 7/(x-4);      // вычисляем функцию
            else                  // иначе
            {
                 if( x == 22 )             // если аргумент равен 22
                      $f= 'error';         // не вычисляем функцию
                 else                 // иначе
                      $f = ( 1 / (x-22) )*2 + x;      // вычисляем функцию
            }

            echo 'f('.$x.')='.$f.'<br>';    // выводим значение функции
        }
?>
```

Первое что мы сделали – убрали лишнюю переменную *$start_value*. Она используется только один раз как источник значения при инициализации переменной *$x* – гораздо проще сразу инициализировать *$x* значением -10. Кроме того, увеличение аргумента на каждой итерации цикла можно также перенести в оператор `for`.

Далее, там, где возможно деление на ноль, следует предусмотреть безопасную обработку этой ситуации. В примере это возможно при *x*=4 и *x*=22 – т.к. в программе присутствуют дроби, в знаменателях которых есть выражения (*x*-4) и (*x*-22). Для второго значения аргумента следует проверить это в соответствующей ветке программы: если аргумент принимает значение 22, то вычисления не выполняются, а функция принимает значение "*error*".

Листинг А-2. 3

```
<?php
       $x = -10;       // начальное значение аргумента
       $encounting = 10000;      // количество вычисляемых значений
       $step = 2;                // шаг изменения аргумента
       $type = 'A';              // тип верстки

       if( $type == 'B' )        // если тип верстки B
           echo '<ul>';          // начинаем список

       // цикл с заданным количеством итераций
       for( $i=0; $i < $encounting; $i++, $x+=$step )
       {
           if( $x <= 10 )        // если аргумент меньше или равен 10
                $f = 32*x / 21;       // вычисляем функцию
           else                  // иначе
           if( $x <20 )                // если аргумент меньше 20
                $f = x*x/3 + 7/(x-4);      // вычисляем функцию
           else                  // иначе
           {
                if( x == 22 )              // если аргумент равен 22
                     $f= 'error';          // не вычисляем функцию
                else                  // иначе
                     $f = ( 1 / (x-22) )*2 + x;      // вычисляем функцию
           }

             if( $type == 'A' )                    // если тип верстки A
             {
                echo 'f('.$x.')='.$f;      // выводим аргумент и значение функции
                if( $i < $encounting-1 )   // если это не последняя итерация цикла
                           echo '<br>';          // выводим знак перевода строки
             }
             else                                  // иначе
             if( $type == 'B' )                    // если тип верстки B
             {
                    // выводим данные как пункт списка
                    echo '<li>f('. $x.')='.$f.'</li>';
             }
       }

       if( $type == 'B' )        // если тип верстки B
           echo '</ul>';         // закрываем тег списка
?>
```

В листинге А-2.2 остался неизменным тип верстки – "A", что не позволяет считать задание лабораторной работы полностью выполненным. Поэтому программа несколько усложниться. В зависимости от типа верстки в теле цикла формируется разный HTML-код: либо строки разделяются символами перевода строки <br>, либо оформляются как элементы списка.

В первом случае мы дополнительно отслеживаем итерацию цикла – если она последняя, то выводить
 не нужно, т.к. строк далее просто нет. Это часто необходимо в программировании, особенно если разделение производится более сложными объектами и элементами дизайна. Признаком того, что итерация не последняя является то, то ее номер меньше общего количества итераций мину один. При этом можно пойти другим путем: выводить символ
 не в конце

строки, а в начале. Тогда необходимо проверять итерацию не на признак последней, а на признак первой – часто это гораздо проще, да и такой код выглядит компактнее и элегантнее.

Вывод строки с аргументом и значением при типе верстки "B" немного проще – строка просто заключается в соответствующий тег. Но этого недостаточно для построения валидного HTML-кода – для списка необходимо использовать тег "<ul>", который в цикле не выводится. Конечно, мы можем прямо в цикле, использовав условные операторы с проверкой на первую и последнюю итерации цикла, вывести этот тег. Но гораздо проще просто вынести вывод тега за пределы цикла, с соответствующей проверкой типа верстки.

Теперь добавим в программу ограничения минимального и максимального значений функции, при появлении которых вычисления останавливаются. Это можно сделать тремя путями: использовать цикл с предусловием, цикл с постусловием или оператор break. Рассмотрим все варианты.

Листинг А-2. 4

```
<?php
       $min_value=10;      // минимальное значение, останавливающее вычисления
       $max_value=20;      // максимальное значение, останавливающее вычисления

       ...                 // другой код PHP

       // цикл с заданным количеством итераций
       for( $i=0; $i < $encounting; $i++, $x+=$step )
       {
           ...                   // другой код PHP

           if( $f>=$max_value || $f<$min_value )      // если вышли за рамки диапазона
                break;                                // закончить цикл досрочно

       }
       ...                 // другой код PHP
?>
```

В начале скрипта во всех трех вариантах добавляются две искомые переменные $min_value и $max_value. Для краткости не будем включать в листинг уже разобранный код, заменив его многоточием. Единственным изменением программы, от варианта А-2.3 является наличие в конце условного оператора, который при превышении вычисленного значения функции $f значения из переменной $max_value или приуменьшении $f значения из $min_value выполняет оператор break, который прекращает работу цикла. Таким образом при достижении указанных условий вычисления прекращаются, что и требовалось по условиям лабораторной работы.

В отличии от цикла со счетчиком, цикл с предусловием не использует счетчик, а работает до тех пор, пока выполняется указанное условие. В листинге А-2.5 это условие интерпретируется так: выполнять цикл до тех пор, пока переменная счетчик итераций $i меньше заданного значения И значение функции в указанных пределах ИЛИ это первая итерация. Обратите внимание – операция "И" имеет приоритет перед операцией "ИЛИ", поэтому второе логическое выражение взято в скобки. Кроме того, признаком первой итерации является нулевое значение переменной $i, что соответствует логическому значению false – поэтому взято логическое отрицание переменной $i (т.е. если $i и всегда true, кроме нулевого значения, то !$i – всегда ложно, кроме нулевого значения).

Листинг А-2. 5

```
<?php

       ...                 // другой код PHP

       $i=0; $f=0;
       // цикл с предусловием
       while( $i<$encounting && ($f>=$max_value || $f<$min_value || !$i) )
       {
```

```
        ...                     // другой код PHP
           $i++; $x+=$step;         // принудительно увеличиваем значения счетчиков
    }
    ...                 // другой код PHP
?>
```

В цикле с предусловием удобно то, что условие полностью записывается в начале цикла и он может выполняться не фиксированное, а заранее неизвестное число итераций. Платой за это является необходимость явно указывать ПЕРЕД циклом начальные значения всех используемых в условии переменных и самостоятельно увеличивать значения необходимых переменных в его конце.

Цикл с постусловием работает аналогично, но условие проверяется не до, а после выполнения первой итерации. Т.е. такой цикл гарантированно выполнится хотя бы один раз, нет необходимости инициализировать все переменные из условия.

Листинг А-2. 6

```
<?php

    ...                 // другой код PHP

    $i=0;                   // начальное значение счетчика итераций
    do  // цикл с постусловием
    {
        ...                 // другой код PHP
           $i++; $x+=$step;         // принудительно увеличиваем значения счетчиков
    }
    while ($i<$encounting && ($f>=$max_value || $f<$min_value) );
    ...                 // другой код PHP
?>
```

В рамках лабораторной работы исследуйте и представьте все три варианта цикла. Выберете и обоснуйте наиболее оптимальный тип цикла для сформулированной задачи.

Представленная программа уже хорошо выполняет задание лабораторной работы. Но необходимо доработать ее так, чтобы она учитывала возможность остальных типов верстки. Для этого удобно, а в рамках этой работы необходимо, использовать конструкцию выбора. Это, а также другие требования задания необходимо сделать самостоятельно.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Цикл со счетчиком	`for( $i=0; $i<100; $i++ ) { ... }`
Цикл с предусловием	`$x=0; while ($x<10) { ...; $x++; }`
Цикл с постусловием	`$x = 0; do { ...; } while ($x++<10);`
Оператор `break`	Прекращает выполнение цикла
Оператор `continue`	Прекращает выполнение текущей ИТЕРАЦИИ цикла

Конструкция выбора	`switch ($x) {`
	`case 0:`
	`  ...`
	`  break;`
	`case 1:`
	`  ...`
	`  break;`
	`default:`
	`  ...`
	`}`

Округление с указанной точностью	`round( $x, $precison);` `round(3.4)` → `3`

round(3.5)	4
round(3.6)	4
round(3.6, 0)	4
round(1.95583, 2)	1.96
round(1241757, -3)	1242000
round(5.045, 2)	5.05
round(5.055, 2)	5.06

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Что такое цикл?
2. Что такое условный оператор?
3. Какие виды циклов бывают?
4. Чем цикл с предусловием отличается от цикла с постусловием?
5. Чем отличаются операторы break и continue?
6. Можно ли решить задачу листингов А-2.4 – 6 четвертым путем? Если да, то каким?
7. Зачем в листинге А-2.5 проверяется условие первой итерации цикла?
8. Что такое итерация?
9. Какой тип данных определяет условие?
10. Что такое булева алгебра?
11. Какие операции булевой алгебры (логические операции) Вы знаете?
12. Как в PHP округлить значение до заданной точности?
13. Как в PHP преобразовать вещественный тип к целому?
14. Какие виды округления используются в PHP?
15. Что такое конструкция выбора?
16. Почему в примере А-2.2 и других не надо обрабатывать возможность деления на ноли при x=4?
17. Как изменить код А-2.3 так, чтобы выводить символ перевода строки "<br>" не в конце сформированной строки с данными, а в ее начале?

Использование GET-параметров в ссылках. Виртуальная клавиатура.

Лабораторная работа № А-3.

ЦЕЛЬ РАБОТЫ

Изучение принципов работы с GET-параметрами, возможностей по вводу и хранению данных с помощью параметров, понимание принципов построения многостраничных веб-сервисов и динамических страниц сайта.

ПРОДОЛЖИТЕЛЬНОСТЬ

2 академических часа (1 занятие)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу http документ (страница сайта) с кнопками цифр от 0 до 9, кнопкой сброса и окном просмотра результата.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Вся функциональность реализуется исключительно с помощью PHP. Кнопки формируются в виде ссылок с необходимыми стилями. Окно просмотра результата формируется из блока (тег `<div>`), выключка текста в блоке – по центру. Внешний вид кнопок и окна просмотра результата соответствует рисунку. Скрипт должен функционировать следующим образом.

- При первой загрузке в браузер окно просмотра результата пусто.
- При нажатии на кнопку "СБРОС" страница перезагружается, в окне просмотра результата ничего нет.
- При нажатии кнопки с цифрой страница перезагружается, к строке в окне просмотра результата добавляется соответствующая цифра.
- В подвале отображается общее число нажатий любых кнопок с момента первой загрузки страницы.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Основной задачей данной лабораторной работы является такая программа, которая бы формировала нужный результат без промежуточного хранения данных. Если бы такое хранилище у нас имелось, то описание алгоритма выглядело бы следующим образом.

- Если хранилища еще нет (первая загрузка страницы) – создаем пустое хранилище.
- Если нажата кнопка с цифрой – добавить в хранилище соответствующую цифру.
- Если нажата кнопка сброс – очистить хранилище.
- Вывести содержимое хранилища.

Предположим, что переменная `$STORE` как раз и является нашим хранилищем. Тогда программу можно представить следующим образом.

Листинг А-3. 1

```
<?php                                   // начало PHP-программы
      if( !isset($STORE) ) $STORE = '';   // если хранилища еще нет – создаем его

      if( isset($_GET['num']) )               // если была нажата кнопка с цифрой
          $STORE .= $_GET['num'];             // сохранить цифру в хранилище

      if( isset($_GET['reset']) )         // если была нажата кнопка СБРОС
          $STORE = '';                    // очистить хранилище

      echo '<div class="result">'.$STORE.'</div>';  // выводим содержимое хранилища
?>
<a href="/?num=1">1</a>
...
<a href="/?num=0">0</a>
<a href="/?reset=Y">СБРОС</a>
```

Кнопки реализованы в виде ссылок, как и требуется в задании. Причем в адресе каждой ссылки в качестве GET-параметра передается значение кнопки (цифра или "*reset*"). При нажатии кнопки ее значение попадает в PHP и может быть прочитано в суперглобальном массиве `$_GET` с соответствующим ключом: если в адресе ссылки указан параметр "`some_param_name=...`", то его значение будет доступно в `$_GET[some_param_name]`. В данном примере кнопки с цифрами используют один и тот же параметр *num*, но с разными значениями: если нажата одна из них, то в PHP появляется элемент массива *`$_GET['num']`*, хранящий переданной значение. Кнопка "СБРОС" использует параметр *reset* – если нажали ее, то появляется элемент массива *`$_GET['reset']`*.

Таким образом, обработка нажатия кнопки с цифрой скриптом очень проста (обработка нажатия кнопки СБРОС производится абсолютно аналогично):

- если в массиве есть элемент с ключом *num*, значит был передан параметр с именем *num*;
- если передан параметр *num*, значит был осуществлен переход по ссылке, которая его содержит;
- если был осуществлен переход по такой ссылке – значит была нажата кнопка с цифрой;
- если была нажата кнопка с цифрой – значит значение параметра и есть цифра на кнопке;
- цифру на нажатой кнопке необходимо добавить в хранилище.

В данном примере используются два параметра: *num* и *reset*. Но исходя из логики программы, если передан один параметр, то другой не будет передан никогда. Поэтому, имеет смысл заменить два параметра одним, назовем его *key*.

Листинг А-3. 2

```
<?php                                   // начало PHP-программы
      if( isset($_GET['key']) )               // если кнопка была нажата
      {
          if( $_GET['key'] == 'reset' )   // если нажата кнопка СБРОС
               $STORE = '';                   // очистить хранилище
          else                            // если нажата другая кнопка
               $STORE .= $_GET['key'];        // сохранить цифру в хранилище
      }
      else
          $STORE = '';    // если хранилища еще не – создаем его
echo '<div class="result">'.$STORE.'</div>';    // выводим содержимое хранилища
?>
<a href="/?key=1">1</a>
...
<a href="/?key=0">0</a>
<a href="/?key=reset">СБРОС</a>
```

В отличии от предыдущего примера, получение параметра *key* (наличие элемента массива *`$_GET['key']`*) означает что была нажата либо цифра, либо кнопка "СБРОС". Поэтому после проверки наличия параметра, необходимо проверить его значение: если это "reset" – была

нажата кнопка "СБРОС", иначе – была нажата кнопка с цифрой. Кроме того, если параметр не передан вовсе – это означает что кнопка не была нажата вовсе, что возможно только в одном случае: первая загрузка страницы. Поэтому создание параметра переносится из начала программы. Получившийся код гораздо более оптимален и понятен, а значит вероятность логической ошибки в программе уменьшается.

К несчастью, в данной лабораторной работе мы не можем пользоваться хранилищами данных, поэтому придется "накапливать" информацию в параметрах. Итак, при каждом вызове страницы, перед тем как обработать переданные параметры, нам необходимо знать результат предыдущей обработки, т.е. ту строку, которая выводилась до этого. В листингах А-3.1 и А-3.2 для этого использовалось хранилище, но оно отсутствует. Единственный выход – ПЕРЕДАВАТЬ В ПРОГРАММУ РЕЗУЛЬТАТ ПРЕДЫДУЩЕЙ ОБРАБОТКИ КАК ЕЩЕ ОДИН ПАРАМЕТР.

Листинг А-3. 3

```
<?php                                  // начало PHP-программы
      $STORE='';                            // создаем пустое хранилище
      if( isset($_GET['store']) )           // если передано предыдущее значение
            $STORE= $_GET['store'];         // сохраняем его в хранилище

      if( isset($_GET['key']) )                 // если кнопка была нажата
      {
          if( $_GET['key'] == 'reset' )   // если нажата кнопка СБРОС
               $STORE = '';                     // очистить хранилище
          else                          // если нажата другая кнопка
               $STORE .= $_GET['key'];          // сохранить цифру в хранилище
      }

      echo '<div class="result">'.$STORE.'</div>';  // выводим содержимое хранилища
?>
<a href="/?key=1&store=<?php echo $STORE; ?>">1</a>
...
<a href="/?key=0&store=<?php echo $STORE; ?>">0</a>
<a href="/?key=reset&store=<?php echo $STORE; ?>">СБРОС</a>
```

Это и делается в начале программы А-3.3: создается пустое хранилище и, если был передан параметр *store* с информации об его предыдущем содержимом – оно переносится в него. Интересно, что, если информация передана – значит была нажата какая-либо кнопка. Следовательно, если информация не была передана, то кнопка не была нажата, т.е. была осуществлена первая загрузка страницы. В этом случае хранилище все равно будет проинициализировано, а значит создание пустого хранилища после проверки на отсутствие параметра *key* – дублирование функционала и уже не нужно.

Для собственно передачи предыдущего значения в ссылки кнопок добавляется второй параметр, который и хранит значение хранилища. Для этого адрес ссылок частично формируется динамически: в ссылку добавляется текущее вычисленное значение, которое для следующей загруженной страницы станет предыдущим.

Оптимизируем рассмотренный код, сократив по возможности передаваемые параметры, используемые переменные и т.д.

Листинг А-3. 4

```
<?php                                  // начало PHP-программы
      if( !isset($_GET['store']) )          // если НЕ передано предыдущее значение
          $_GET['store'] = '';              // создаем пустое хранилище
     else                               // иначе
      if( isset($_GET['key']) )                 // если кнопка была нажата
          $_GET['store'].= $_GET['key'];        // сохранить цифру в хранилище

     // выводим содержимое хранилища
      echo '<div class="result">'.$_GET['store'].'</div>';
?>
```

```
<a href="/?key=1&store=<?php echo $_GET['store']; ?>">1</a>
...
<a href="/?key=0&store=<?php echo $_GET['store']; ?>">0</a>
<a href="/">СБРОС</a>
```

Первое что следует сделать – убрать переменную `$STORAGE`, которая в программе полностью дублирует элемент массива `$_GET['store']`. Это упрощает начальную инициализацию хранилища: если параметр не передан, то мы инициализируем его пустой строкой. Кроме того, нажатие кнопки "СБРОС" на данном этапе полностью соответствует начальной загрузке страницы, поэтому эта ссылка не передает никаких параметров вовсе. Следовательно, и обработку значения *reset* тоже можно убрать.

Убедитесь, что если вручную в URL страницы передать любое значение в параметре *key*, то оно добавится в окно просмотра результата. Самостоятельно доработайте программу согласно требованиям лабораторной работы.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Передача одного параметра в ссылке	`<a href="?param=value"></a>` `<?php echo $_GET['param'];  // выводит value  ?>`
Передача двух параметров в ссылке	`<a href="?p1=v1&p2=v2"></a>` `<?php echo $_GET['p1'].'_'.$_GET['p2'];  // v1_v2  ?>`
Передача трех параметров в ссылке	`<a href="?p1=v1&p2=&p3=3"></a>` В качестве значения параметра p2 передана пустая строка
Существование переменной	`isset( $v ); // true если переменная была инициализирована`
Наличие в массиве элемента с ключом	`array_key_exists( $key, $array )`

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Как передать GET-параметр через ссылку?
2. Что такое $_GET?
3. Как передать в ссылке несколько параметров?
4. Как проверить существование переменной?
5. Как узнать, был ли передан параметр в скрипт?
6. Как узнать значение переданного в сценарий параметра?
7. Изменится ли работа программы А-3.1, если между двумя условными операторами поставить else? Если да – то как?
8. Изменится ли работа программы А-3.4, если между двумя условными операторами убрать else? Если да – то как?
9. Изменится ли работа программы А-3.4, если убрать второй условный оператор, оставив else? Если да – то как?

Пользовательские функции. Вывод таблиц.

Лабораторная работа № А-4.

ЦЕЛЬ РАБОТЫ

Изучение преимуществ и особенностей использования пользовательских функций в языке PHP, закрепление навыков построения простейших алгоритмов обработки информации.

ПРОДОЛЖИТЕЛЬНОСТЬ

2 академических часа (1 занятие)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу http документ (страница сайта) с несколькими таблицами.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

В начале программы задается массив с не менее 10 разными элементами, содержащие строки формата: "*C1*C2*C3#C4*C5*C6# … Cn*", где C*x* – произвольный текст, символ "*" – разделитель колонок таблицы, символ "#" – разделитель строк. На странице необходимо вывести соответствующие заданным в переменных структурам таблицы. При этом необходимо разработать и использовать функции для:

- вывода HTML-кода таблицы исходя из ее структуры указанного выше формата;
- формирования содержимого отдельной строки таблицы

Кроме того, при выводе таблиц необходимо учитывать следующее.

- Перед выводом каждой новой таблицы необходимо добавить заголовок второго уровня (тег `<h2>`) формата "Таблица №*x*", где *x* – номер выводимой таблицы.
- В начале программы необходимо инициализировать переменную, которая бы содержала число колонок таблиц. Таблицы всегда должны выводится с требуемым числом колонок (возможно с пустыми ячейками в них), вне зависимости от заданной структуры. Если задано нулевое число колонок – таблицы не выводятся, выводится надпись: "Неправильное число колонок".
- Если в строке нет ячеек – ее HTML-код не выводится.
- Если в структуре таблицы нет строк с ячейками – таблица не выводится, выводится надпись: "В таблице нет строк с ячейками".
- Если в структуре таблицы нет строк – таблица не выводится, выводится надпись: "В таблице нет строк".

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Допустим у нас есть только одна переменная со структурой таблицы. Тогда для ее вывода в форме HTML-кода можно использовать следующий укрупненный алгоритм.

- Разбить структуру на элементы, соответствующие строкам таблицы.
- Если строк одна или более, то:
    - начать вывод таблицы;
    - вывести все строки таблицы по одной;
    - закончить вывод таблицы.

Видно, что часто повторяющееся действие – вывод строки таблицы. Это признак того, что его можно оформить в виде отдельной функции, что значительно улучшит наглядность кода, позволит более просто искать ошибки и изменять его. Напишем такую функцию и назовем ее `getTR()` – в качестве аргумента в нее передается строка вида: "*C1*C2*C3*".

Листинг А-4. 1

```
function getTR( $data )                          // объявление функции
{
     $arr = explode( '*', $data );               // разбиваем строку в массив
     $ret = '<tr>';                         // начинаем тег строки таблицы

     for($i=0; $i<count($arr); $i++)             // цикл по всем ячейкам таблицы
          $ret .= '<td>'.$arr[$i].'</td>';           // добавляем ячейкам тег

     return $ret.'</tr>';                   // возвращаем строку таблицы
}
```

В качестве аргумента функции выступает переменная `$data` – при обращении к ней в теле функции, мы будем получать его значение. Важно помнить, что нет никаких правил для имен переменных аргументов функций – мы вполне могли назвать такую переменную к примеру `$val` или по-другому.

В теле функции переданная строка со структурой разбивается на соответствующие ячейкам таблицы элементы (разделитель элементов – символ "*"), которые сохраняются в массив $arr. Затем последовательно в цикле эти элементы "оборачиваются" тегом `<td>` и добавляются к переменной, содержащей результат работы функции *`$ret`*. Эта переменная инициализируется в самом начале работы функции и содержит открывающийся тег `<tr>`, затем к ней добавляется необходимое количество тегов `<td>`. Закрывающийся тег `</tr>` в переменную не добавляется – он комбинируется с ней непосредственно при возврате результата работы функции.

Теперь, можно реализовать описанный выше алгоритм, используя построенную функцию как его элемент.

Листинг А-4. 2

```
$structure = 'C1*C2*C3#C4*C5*C6';          // структура таблицы

$strings = explode( '#', $structire );          // разбиваем структуру на строки

$datas='';                              // итоговый HTML-код строк
for($i=0; $i<count($strings); $i++ )       // цикл для всех строк из структуры
     $datas .= getTR( $strings[$i] );           // добавляем код строки в итоговый

if( $datas )                                    // если код строк определен
     echo '<table>'.$datas.'</table>';     // выводим таблицу
else                                    // иначе
     echo 'В таблице нет строк ';          // выводим предупреждение
```

Структура определяется в переменной *`$structure`*. Затем, с помощью стандартной функции `explode()`, структура разбивается на отдельные элементы, соответствующие строкам таблицы (признаком окончания такого элемента является символ "#"). В цикле последовательно для каждого из них вызывается функция `getTR()` , которая возвращает соответствующий строке HTML-код. Код накапливается в переменной `$datas` , а затем, если в итоге обработки всех строк, переменная содержит что-нибудь (хотя бы одна строка успешно распознана и преобразована) – выводится тег таблицы <table> и HTML-код строки. Если переменная пуста – выводится соответствующее сообщение.

Код успешно работает, но что делать если надо вывести 5 таких таблиц? Скопировать его и вставить несколько раз, изменив значение переменной *`$structure`*? Да, это сработает – но если таблиц 500? Чтобы избежать такой проблемы, одинаковые фрагменты кода также оформляют в виде отдельных функций. Сделаем это и мы.

Листинг А-4. 3

```
function outTable($structure)                    // объявление функции
{
      $strings = explode( '#', $structure );        // разбиваем структуру на строки

      $datas='';                                 // итоговый HTML-код строк
      for($i=0; $i<count($strings); $i++ )           // цикл для всех строк
          $datas .= getTR( $strings[$i] );          // добавляем код строки в итоговый

      if( $datas )                               // если код строк определен
          echo '<table>'.$datas.'</table>';      // выводим таблицу
      else                                       // иначе
          echo 'В таблице нет строк ';           // выводим предупреждение
}

outTable('C1*C2*C3#C4*C5*C6');

$structure='C7*C8*C9#C10*C11*C12';
outTable($structure);

$data='C13*C14*C15#C16*C17*C18';
outTable($data);
```

Обратите внимание: эта функция, в отличии от `getTR()`, не возвращает значения, она сама выводит результат. Переменную со структурой таблицы мы перенесли в аргументы функции, что позволило минимизировать правку кода программы. Тогда, для вывода нескольких таблиц не надо копировать весь код – достаточно только скопировать вызовы функции. Кроме того, в примере показана равнозначность передачи аргументов в функцию: можно передавать сразу значение, можно использовать промежуточную переменную, как совпадающую с именем в объявлении функции, так и нет.

Теперь еще больше минимизируем код, сохранив структуры разных таблиц в массиве и вызывая функцию в цикле.

Листинг А-4. 4

```
$structure=array(    'C1*C2*C3#C4*C5*C6', 'C7*C8*C9#C10*C11*C12',
                'C13*C14*C15#C16*C17*C18' );    // массив со структурами таблиц

for($i=0; $i<count($structure); $i++)      // для всех элементов массива
      outTable($structure[$i]);            // выводим соответствующую структуре таблицу
```

Таким образом, использование функций значительно упрощает программирование, уменьшает размер кода, повышает его наглядность.

Самостоятельно доработайте программу так, чтобы она полностью удовлетворяла требованиям лабораторной работы.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Создание массива	`array();`
	Создает массив из указанных элементов. Например:
	`$list_1 = array('A'); // список из одного элемента`
	`$list_2 = array(1, 2, 3); // список из 3-х элементов`
	`$list_3 = array(); // пустой массив`
	`// ассоциированный массив из двух элементов`
	`$assoc_arr = array('A'=>12, 'D'=>'ff');`

Функция без аргументов	`function f() { ... }  // объявление функции` `f();  // вызов функции`
Функция с одним аргументом	`function f($a) { ... }` `f(10);`
Функция с двумя обычными аргументами	`function f($a, $b) { ... }` `f(10, 'ADC');`

Функция с двумя обычными и двумя аргументами со значением по умолчанию	`function f($a, $b, $c=10, $d=true) { ... }`
	`f(10, 'ADC');  // варианты вызова функции`
	`f(10, 'ADC', 17);`
	`f(10, 'ADC', 17, false);`

Возврат значения функции	`function f($a) { return $a+2; }` `echo f(10); // выводит "12"`

Синтез возврата значения и вывода строки в функции	`function f($a) { echo 'Summa'; return $a+2; }`
	`// выводит "Start Summa 12 End"`
	`echo 'Start '.f(10).' End';`

Разбиение строки на элементы массива с использованием символа-разделителя	`// в массиве элементы [a], [b], [d]`
	`$arr = explode( 's', 'asbsd' );`
	`// в массиве элементы [a], [b], [d], []`
	`$arr = explode( 's', 'asbsds' );`
	`// в массиве элемент [abd]`
	`$arr = explode( 's', 'abd' );`
	`// в массиве элемент [], [ab], [d]`
	`$arr = explode( 's', 'sabsd' );`

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Что такое пользовательская функция?
2. Как функция возвращает значение?
3. Можно ли вызвать из функции другую функцию?
4. Можно ли вызвать из функции эту же функцию?
5. Продолжит ли функция свою работу после выполнения инструкции `return`?
6. Сколько раз инструкция `return` может быть использована в теле функции?
7. Сколько аргументов может быть передано функции?
8. Что-такое аргументы "по умолчанию"?
9. Может ли функция вообще не иметь аргументов?
10. Должны ли совпадать имена переменных-аргументов при объявлении и при вызове функции?
11. Если в теле функции объявлена переменная $P – изменится ли эта переменная с таким же именем в основной программе? В другой функции? В этой же функции, но вызванной другой раз?
12. В каких случаях имеет смысл использовать пользовательские функции?
13. Может ли функция выводить HTML-код?
14. Может ли функция вызываться непосредственно в операторе `echo`?
15. Может ли функция одновременно выводить текст в HTML-код браузера и возвращать значение?
16. В каком порядке формируется HTML-код, если функция не только вызывается непосредственно из оператора echo, но и выводит что-либо внутри себя?

Динамическое формирование контента и меню. Таблица умножения.

Лабораторная работа № А-5.

ЦЕЛЬ РАБОТЫ

Освоение навыков динамического формирования контента в зависимости от набора параметров. Закрепление знаний по основам программирования простейших алгоритмов на PHP. Получение навыков использования пользовательских функций.

ПРОДОЛЖИТЕЛЬНОСТЬ

6 академических часов (3 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу http документ (страница сайта) с динамически формируемой таблицей умножения.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Работа оформляется в виде одного HTML-документа с интегрированным PHP-кодом. При открытии страницы в браузере должен отображаться следующий контент.

1. Главное меню (в шапке страницы) из двух пунктов: "Табличная верстка"; "Блочная верстка".
   - Оба пункта меню реализуются с помощью тега `<a>` и передают один и тот-же GET-параметр, значение которого определяет тип HTML-верстки таблицы умножения.
   - По умолчанию (при первой загрузке, если параметр не выбран) используется табличная верстка.
   - Ни один из пунктов меню по умолчанию (при первой загрузке) не выделен!
   - При переходе по ссылке этот пункт меню выделяется каким-либо образом.
   - Внешний вид страницы при обоих типах верстки должен совпадать.
   - При изменении типа верстки содержание таблицы умножения не должно изменяться.
2. Основное меню (в левой части страницы) с пунктами: Всё, 2, 3, 4, 5, 6, 7, 8, 9.
   - При выборе первого пункта (выделен по умолчанию при первой загрузке) выводится вся таблица умножения: восемь колонок.
   - При выборе пункта с цифрой – выводится таблица умножения на соответствующую цифру (более крупно).
   - При выборе любого пункта меню – этот пункт выделяется любым способом.
   - При изменении содержания таблицы умножения тип верстки не должен изменяться.
3. Информация о содержании страницы (в подвале).
   - тип верстки;
   - название таблицы умножения (полная или одна колонка);
   - дата и время.
4. Таблица умножения (в основной части страницы).
   - Тип верстки и содержание определяется заданными параметрами (или их отсутствием).
   - Все цифры (и только цифры) должны быть ссылками на соответствующие таблицы умножения. Например, строка 2x3=6 должна содержать три ссылки: на таблицы умножения на 2, 3 и 6. Строка 3x5=15 – две ссылки: на 3 и 5. Указанные ссылки всегда «сбрасывают» тип верстки – соответствующий параметр не указывается.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Программу (скрипт) можно разделить на четыре основных блока:

1. главное меню (вверху, определяет тип верстки);
2. основное меню (сбоку, определяет содержание таблицы);
3. таблица умножения;
4. информация о таблице умножения.

Как и любой другой PHP-скрипт, программа должна в зависимости от переданных в нее параметров выполнять какие-либо действия, выводить какую-либо информацию. Исходя из условий данной лабораторной работы, необходимо и достаточно два таких параметра: *html_type* (тип верстки) и *content* (содержание таблицы умножения). В зависимости от их значения каждый блок принимает тот или иной вид. Сочетание параметров влияет на внешний вид каждого блока. Составим таблицы, в которой укажем все сочетания параметров и вид основных блоков. Блоки рекомендуется реализовывать последовательно, в указанном выше порядке.

ГЛАВНОЕ МЕНЮ

	html_type n/a	**html_type = TABLE**	**html_type = DIV**
content n/a	Оба пункта меню не выделены	Пункт «**Табличная верстка**» выделен. Пункт «**Блочная верстка**» – НЕ выделен.	Пункт «**Табличная верстка**» НЕ выделен. Пункт «**Блочная верстка**» – выделен.
content=2			
content=3			
content=4			
content=5			
content=6			
content=7			
content=8			
content=9			

Из таблицы видно, что на главное меню оказывает влияние только один параметр. Тогда реализация блока «Главное меню» может быть представлена следующим кодом.

Листинг А-5. 1

```
<div id=”main_menu”><?php
       echo '<a href=”?html_type=TABLE”'; // начало ссылки ТАБЛИЧНАЯ ФОРМА

       // если в скрипт был передан параметр html_type и параметр равен TABLE
       if( array_key_exists('html_type', $_GET) && $_GET['html_type']== 'TABLE' )
              echo ' class=”selected”'; // ссылка выделяется через CSS-класс
       echo '>Табличная форма</a>'; // конец ссылки ТАБЛИЧНАЯ ФОРМА

       echo '<a href=”?html_type=DIV”'; // начало ссылки БЛОКОВАЯ ФОРМА

       // если в скрипт был передан параметр html_type и параметр равен DIV
       if( array_key_exists('html_type', $_GET) && $_GET['html_type']== 'DIV' )
              echo ' class=”selected”'; // ссылка выделяется через CSS-класс

       echo '>Табличная форма</a>'; // конец ссылки БЛОКОВАЯ ФОРМА
?></div>
```

Для оформления главного меню в соответствии с дизайном используется тег `<div>` – т.к. он не изменяется в зависимости от переданных параметров, его вывод осуществляется не PHP-скриптом, а статически из HTML-кода страницы. В программе осуществляется вывод двух пунктов меню в виде ссылок. Их оформление в форме кнопок или иной внешний вид осуществляется CSS. Вывод производится в три этапа.

1. Выводится начало HTML-кода ссылки (вывод не зависит от переданного в скрипт параметра *html_type*). Здесь полностью указан адрес ссылки, в котором закодирован передаваемый параметр *html_type* и его значение.
2. Выводится выделение ссылки, если это необходимо. Для этого проверяется наличие имени параметра *html_type* в списке переданных программе методом GET параметров (массив `$_GET`) и его соответствие требуемому значению. Обратите внимание – параметр, который передает ссылка, и параметра от которого зависит ее внешний вид, совпадает. Действительно – внешний вид пункта меню зависит от того параметра, который он и передает (т.е. признаком необходимости выделения пункта меню является его нажатие, а значит передача указанного в нем параметра скрипту).
3. Выводится окончание ссылки с его текстом. Этот этап также не зависит от переданных в программу параметров.

ОСНОВНОЕ МЕНЮ

	html_type n/a	**html_type = TABLE**	**html_type = DIV**
content n/a	Выделен пункт меню «Вся таблица умножения»		
content=2	Выделен пункт меню «Таблица умножения на 2»		
content=3	Выделен пункт меню «Таблица умножения на 3»		
content=4	Выделен пункт меню «Таблица умножения на 4»		
content=5	Выделен пункт меню «Таблица умножения на 5»		
content=6	Выделен пункт меню «Таблица умножения на 6»		
content=7	Выделен пункт меню «Таблица умножения на 7»		
content=8	Выделен пункт меню «Таблица умножения на 8»		
content=9	Выделен пункт меню «Таблица умножения на 9»		

Принцип работы программы для основного меню аналогичен блоку главного меню. Однако в его реализации, в силу его большого размера (а на практике возможны случаи меню из нескольких сотен пунктов, описание которого будет чрезвычайно трудоемко) предполагает использование цикла.

Листинг А-5. 2

```
<div id="product_menu"><?php

      echo '<a href="/"'; // начало ссылки ВСЯ ТАБЛИЦА УМНОЖНЕНИЯ
      if( !isset($_GET['content']) ) // если в скрипт НЕ был передан параметр content
            echo ' class="selected"'; // ссылка выделяется через CSS-класс
      echo '>Вся таблица умножения</a>'; // конец ссылки

      for( $i=2; $i<=9; $i++ ) // цикл со счетчиком от 2 до 9 включительно
      {
            echo '<a href="?content='.$i. '" '; // начало ссылки
            // если в скрипт был передан параметр content
            // и параметр равен значению счетчика
            if( isset($_GET['content']) && $_GET['content']==$i )
                  echo ' class="selected"'; // ссылка выделяется через CSS-класс
            echo '>Таблица умножения на '.$i.'</a>'; // конец ссылки
}
?></div>
```

Первая ссылка выводится вне цикла, т.к. она не передает никакой параметр и признаком ее выделения служит именно отсутствие переданного параметра *content* (функции `array_key_exists('content',$_GET)` и `isset($_GET['content'])` в данном случае эквивалентны).

Цикл последовательно совершает восемь итераций, выводя восемь однотипных ссылок. При этом значение передаваемого в ссылке параметра *content* и ее текст определяются номером итерации (значением переменной `$i`). Фактически такое использование циклов позволит вывести сколько угодно ссылок – иначе пришлось бы копировать и исправлять код для каждой из них отдельно,

что неизбежно привело бы к увеличению объема кода, ухудшению читаемости, усложнению внесения изменений и росту вероятности ошибок.

На каждой итерации происходит:

- вывод адрема ссылки (со значением параметра *content* равному номеру итерации `$i`);
- проверка передачи в программу параметра *content* и равенства его значения текущей итерации (т.е. проверка, не была ли эта ссылка нажата для перехода на эту страницу) с выделением ссылки при ее прохождении;
- вывод текста ссылки (с использованием номера итерации).

СОПРЯЖЕНИЕ ГЛАВНОГО И ОСНОВНОГО МЕНЮ

Реализация приведенного выше кода главного и основного меню не предполагает сохранение выделенных пунктов меню при переходе по ссылке другого меню. Действительно, если мы перешли по ссылке «Таблица умножения на 4» и нажали на пункт «Блочная верстка», то выделение пункта меню «Таблица умножения на 4» пропадет. А это не соответствует условиям лабораторной работы.

Для решения задачи сопряжения меню и сохранения выделенных пунктов добавим в код ссылок второй параметр. Т.е. ссылки главного меню должны передавать в программу не только параметр *html_type*, но и текущее значение параметра *content* (при условии его существования). Ссылки основного меню – аналогично. Поэтому PHP-код необходимо модифицировать следующим образом (для примера возьмем ссылку главного меню «Табличная форма»).

Листинг А-5. 3

```
echo '<a href="?html_type=TABLE'; // начало ссылки ТАБЛИЧНАЯ ФОРМА

if( !isset($_GET['content']) ) // если параметр content был передан в скрипт
      echo '&content='.$_GET['content']; // добавляем в ссылку второй параметр

echo '"'; // завершаем формирование адреса ссылки и закрываем кавычку

// если в скрипт был передан параметр html_type и параметр равен TABLE
if( array_key_exists('html_type', $_GET) && $_GET['html_type']== 'TABLE' )
      echo ' class="selected"'; // ссылка выделяется через CSS-класс

echo '>Табличная форма</a>'; // конец ссылки ТАБЛИЧНАЯ ФОРМА
```

Начало ссылки слегка отличается от исходного варианта – нет закрывающей ссылку кавычки. Ее нельзя использовать сразу т.к. пока не известно закончено формирование ссылки, или же в ней надо передать второй параметр. Проверка этого осуществляется на второй строке: если второй параметр был передан в скрипт ранее, то его необходимо сохранить и добавить в формируемую ссылку. И только после этого осуществляется вывод закрывающей кавычки. Таким образом, хотя внешний вид меню зависит только от одного параметра – его HTML-код, а именно код ссылок, зависит от обоих параметров!

Модификация остальных пунктов главного и основного меню осуществляется аналогичным образом в соответствии с логикой работы программы и здравым смыслом.

ТАБЛИЦА УМНОЖЕНИЯ

	html_type n/a	**html_type = TABLE**	**html_type = DIV**
content n/a	Верстка табличная. Вся таблица умножения.		Верстка блочная. Вся таблица умножения.
content=2	Верстка табличная. Таблица умножения на 2		Верстка блочная. Таблица умножения на 2.
content=3	Верстка табличная. Таблица умножения на 3		Верстка блочная. Таблица умножения на 3.

content=4	Верстка табличная. Таблица умножения на 4	Верстка блочная. Таблица умножения на 4.
content=5	Верстка табличная. Таблица умножения на 5	Верстка блочная. Таблица умножения на 5.
content=6	Верстка табличная. Таблица умножения на 6	Верстка блочная. Таблица умножения на 6.
content=7	Верстка табличная. Таблица умножения на 7	Верстка блочная. Таблица умножения на 7.
content=8	Верстка табличная. Таблица умножения на 8	Верстка блочная. Таблица умножения на 8.
content=9	Верстка табличная. Таблица умножения на 9	Верстка блочная. Таблица умножения на 9.

В отличие от любого из меню на блок таблицы умножения влияют оба параметра. Первый определяет тип верстки, второй – выводить ли всю таблицу или же только один ее столбец. Для реализации программы блока удобно использовать функции – это упрощает структуру программы, уменьшает размер кода, последующее внесение изменений и вероятность ошибок. Определим в программе две функции.

Листинг А-5. 4

```
// функция ВЫВОДИТ ТАБЛИЦУ УМНОЖЕНИЯ В ТАБЛИЧНОЙ ФОРМЕ
function outTableForm()
{
	// код тела функции
}

// функция ВЫВОДИТ ТАБЛИЦУ УМНОЖЕНИЯ В БЛОЧНОЙ ФОРМЕ
function outDivForm ()
{
	// код тела функции
}
```

Первая функция `outTableForm()` – осуществляет всю работу по выводу таблицы умножения в табличной форме, вторая функция `outDivForm()` – в блочной форме. Тогда для обработки первого параметра достаточно следующего кода.

Листинг А-5. 5

```
if (!isset($_GET['html_type']) && $_GET['html_type']== 'TABLE' )
	outTableForm();	// выводим таблицу умножения в табличной форме
else
	outDivForm();  // выводим таблицу умножения в блочной форме
```

В нем проверяется существование параметра *html_type* и его значение. Если он не существует (вывод по умолчанию) или его значение равно *TABLE* – вызывается функция, выводящая таблицу умножения в табличной форме, иначе – в блоковой.

Для примера рассмотрим работу функции `outDivForm()`, функция `outTableForm()` строится аналогичным образом, за исключением выводимых HTML-тегов для верстки результатов в виде таблицы.

Листинг А-5. 6

```
function outDivForm ()
{
	// если параметр content не был передан в программу
	if( !isset($_GET['content']) )
```

```
	{
		for($i=2; $i<10; $i++) // цикл со счетчиком от 2 до 9
		{
			echo '<div class="ttRow">'; // оформляем таблицу в блочной форме
			outRow( $i ); // вызывем функцию, формирующую содержание
				// столбца умножения на $i (на 4, если $i==4)
			echo '</div>';
		}
	}
	else
	{
		echo '<div class="ttSingleRow">'; // оформляем таблицу в блочной форме
		outRow( $_GET['content'] ); // выводим выбранный в меню столбец
		echo '</div>';
	}
}
```

Функция работает следующим образом. Если параметр *content* не был передан в программу, то это означает вывод всей таблицы умножения. Для этого последовательно в цикле восемь раз вызывается функция `outRow()` с параметром, соответствующим номеру итерации. Эта функция полностью формирует содержание указанного столбца таблицы умножения – например, если передан параметр 4, то и сформируется столбец таблицы умножения на 4. Поэтому, вызов этих функций в цикле и соответствующее «обрамление» их HTML-тегами формирует всю таблицу умножения.

Если же параметр *content* указан, то выводимый столбец хранится в нем. Тогда, достаточно вызвать описанную функцию `outRow()` только один раз, передав ей параметр *content*. Обратите внимание – внешний вид блока полной таблицы умножения и одиночного блока может отличаться за счет использования разных CSS-классов: `ttRow` и `ttSingleRow`.

Листинг А-5. 7

```
// функция ВЫВОДИТ СТОЛБЕЦ ТАБЛИЦЫ УМНОЖЕНИЯ
function outRow ( $n )
{
	for($i=2; $i<=9; $i++)		// цикл со счетчиком от 2 до 9
		echo $n.'x'.$i.'='.($i*$n).'<br>';	// выводим строку столбца с тегом
}
```

В теле функции `outRow()` также удобно использовать цикл. С его помощью за восемь итерации выводятся восемь строк требуемого столбца. Обратите внимание – арифметические операции в операторе `echo` рекомендуется заключать в скобки (во избежание путаницы при преобразовании типов переменных)! Небольшая модификация функции позволит выполнить и последнее требование в лабораторной работе – формирование для всех чисел в таблице ссылок.

Листинг А-5. 8

```
// функция ВЫВОДИТ СТОЛБЕЦ ТАБЛИЦЫ УМНОЖЕНИЯ
function outRow ( $n )
{
	for($i=2; $i<=9; $i++) // цикл со счетчиком от 2 до 9.
		echo outNumAsLink($n).'x'.outNumAsLink($i).'='.
			outNumAsLink($i*$n).'<br>';
}
```

Функция `outNumAsLink()` преобразует число в соответствующую ему ссылку (если это возможно) и возвращает ее. Обратите внимание – функция ничего не выводит, а лишь возвращает значение! Поэтому ее код, заданный в начале описания данной лабораторной работы, необходимо самостоятельно модифицировать соответствующим образом.

ИНФОРМАЦИЯ О ТАБЛИЦЕ УМНОЖЕНИЯ

Информация о таблице умножения и типе верстки формируется исходя из значений обеих переменных. Для вывода даты и времени используется функция `date()`.Программе проверяет тип верстки и сохраняет его в переменной `$s`. Содержание таблицы добавляется к строке `$s` с помощью конкатенации (символ "."). Сформированная строка выводится вместе с текущей датой и временем.

Листинг А-5. 9

```
if( !isset($_GET['html_type']) || $_GET['html_type']== 'TABLE' )
      $s='Табличная верстка. '; // строка с информацией
else
      $s='Блочная верстка. ';

if( !isset($_GET['content']) )
      $s.='Таблица умножения полностью. ';
else
      $s='Столбец таблицы умножения на '.$_GET['content']. '. ';

echo $s.date('d.Y.M h:i:s');
```

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Для реализации меню используются ссылки, само меню оформляется в виде одного блока. Текущий (выбранный) пункт меню оформляется в виде специального класса. В зависимости от описания блоков в CSS меню будет выглядеть как вертикальное или горизонтальное.

Листинг А-5. 10

```
<div id=”main_menu”>
      <a href=”goTo1.html”></a>
      <a href=” goTo2.html” class=”selectedMenu”></a>
      <a href=”goTo3.html”></a>
</div>
```

Листинг А-5. 11

```
#main_menu a { display: block; width:25%; color: #F00; } /* вертикальное меню */

/* горизонтальное меню */
#main_menu a { display: block; width:25%; float: left; color: #F00; }
#main_menu { clear: all; }
#main_menu a:hover { color: #FF0; }
#main_menu .selectedMenu, #main_menu .selectedMenu:hover { color: #0FF; }
```

Если активность пункта меню определяется наличием параметра, то его можно проверить, используя функцию array_key_exists().

Листинг А-5. 12

```
<a href=”?myparam=10” <?php
// ссылка активна если параметр не указан или он равен значению ссылки
if( !array_key_exists('myparam', $_GET) || $_GET['myparam']==10 )
      echo 'class=”selected”';
?>>10</a>
<a href=”?myparam=20” <?php
// ссылка активна только(!) если параметр равен значению ссылки
if( array_key_exists('myparam', $_GET) && $_GET['myparam']==20 )
      echo 'class=”selected”';
?>>20</a>
```

При формировании ссылки постоянную часть удобнее генерировать один раз, сохраняя ее в переменной.

Листинг А-5. 13

```
$link='?type=table';
if( array_key_exists('type', $_GET) && $_GET['type']== 'block' )
	$link='?type=block';
for( $i=1; $i<=9; $i++)
{
	echo '<a href="'.$link.'&src='.$i. '">'.$i.'</a>';
}
```

Активное использование пользовательских функций упрощает структуру программы. Для часто повторяющихся действий (в рамках данной работы это оформление ссылки из числа, вывод столбца таблицы умножения) рекомендуется использовать функции.

Листинг А-5. 14

```
function outNumAsLink( $x )  // функция ВЫВОДИТ ЧИСЛО КАК ССЫЛКУ
{
	if( $x<=9 )
		echo '<a href="?content='.$x. '"> '.$x.'</a>';
	else
		echo $x;
}
```

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Можно ли в ссылке использовать три параметра?
2. Можно ли передавать в ссылку параметр без значения?
3. Какое начальное значение может быть у цикла со счетчиком?
4. Что такое URL?
5. Что такое URI?
6. Что такое конкатенация?
7. Можно ли применить операцию конкатенации к строке и функции?

Использование форм для передачи данных в программу PHP. Тест математических знаний.

Лабораторная работа № А-6.

ЦЕЛЬ РАБОТЫ

Закрепление базовых знаний использования *PHP*; приобретение навыков работы с формами, как с основным инструментом получения входных данных для ВЕБ-приложений. Обзор возможностей языка *PHP* для ввода/вывода данных, в том числе с помощью почтовых сообщений.

ПРОДОЛЖИТЕЛЬНОСТЬ

2 академических часа (1 занятие)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу *http* документ (одна страница сайта) с формой, позволяющей определить математическую задачу и ввести предполагаемый ответ. При отправке формы осуществляется автоматическое решение задачи, сравнение переданного и полученного результата, вывод и при необходимости отправка по электронной почте результатов вычисления и сравнения.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Результата лабораторной работы – одна *html*-страница: вывод формы, обработка данных и вывод выполняется одной *PHP*-программой. При первой загрузке на странице сайта должна отображаться форма, содержащая следующие элементы.

- Кнопка "Проверить".
- Однострочные поля для ввода текста:
  - ФИО;
  - номер группы;
  - значение A;
  - значение B;
  - значение C;
  - Ваш ответ;
  - Ваш е-майл.
- Многострочные поля для ввода текста:
  - немного о себе.
- Селектор со следующими опциями:
  - площадь треугольника;
  - периметр треугольника;
  - объем параллелепипеда;
  - среднее арифметическое;
  - другие, самостоятельно придуманные опции (общее число опций – не менее 6).
- Флажок:
  - отправить результат теста по е-майл.
- Селектор со следующими опциями:
  - версия для просмотра в браузере;
  - версия для печати.

Математическая задача определяется с помощью селектора. Входными данными для нее являются три введенных значения: *A*, *B* и *C*. Например, если выбрана задача среднее арифметическое, то ее решением является (*A*+*B*+*C*)/3. При загрузке страницы в качестве значений берутся и выводятся как начальные значения полей произвольные числа от 0 до 100. В качестве

значений могут выступать как целые числа, так и десятичные дроби (допустимо использовать в дробях как точку, так и запятую).

Флажок "Отправить результат теста на е-майл" изначально не отмечен; поле "Е-майл" вместе с надписью скрыто. При отметке флажка поле и соответствующая надпись появляются в форме (реализуется с помощью JavaScript).

Все элементы формы должны быть сгруппированы так, чтобы их было удобно использовать. Все они должны быть выровнены (включая кнопку) по левому краю, быть одинакового размера (кроме флажка и кнопки). Подписи к элементам должны располагаться слева от них.

При нажатии кнопки "Проверить" страница перезагружается с тем-же URL что и прежде. В зависимости от выбранного в списке значения внешний вид страницы может быть простым (для печати) или более сложным (для просмотра в браузере). На странице выводятся отчет со следующими данными:

- ФИО и группа студента;
- сведения о студенте (немного о себе);
- тип задачи;
- входные данные (числа);
- предполагаемый тестируемым результат (вычисленный человеком результат решения задачи);
- вычисленный программой результат;
- выводы об успешности теста ("Тест пройден" или "Ошибка: тест не пройден").

В случае, если был установлен флажок "Отправить результат теста по е-майл" – должна быть выведена надпись: "Результаты теста были автоматически отправлены на e-mail" с указанием введенного тестируемым адреса.

Если был выбран пункт списка "Версия для просмотра в браузере", то внизу результатов теста должна выводиться ссылка "Повторить тест", при переходе по которой должна вновь загрузиться форма (произвольные числа в полях *A*, *B* и *C* должны быть другими). Поля "*ФИО*" и "*Номер группы*" должны быть заполнены ранее введенными значениями. Кнопка оформляется в виде ссылки `<a>` (с фоном и рамкой), реагирует на курсор мыши.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Представленное в лабораторной работе задание является классическим обработчиком данных формы. Причем, исходя из задания, формировать элементы формы и выводить результаты ее обработки должна одна программа.

Рассмотрим сначала обработку данных. Обычно обработчики формы располагаются в самом начале html-страницы, т.к. от результата их работы зачастую зависит ее содержание.

Листинг А-6. 1

```
<?php
if( isset( $_POST['A'] ) ) // если из формы были переданы данные
{
     ... // обработчик переданных из формы данных
}
?><!doctype html>
...
```

Явным и единственным признаком того, что пользователем была заполнена форма и ее данные переданы в скрипт для обработки – это наличие элементов с именами тегов формы в массиве `$_POST` или `$_GET` (в зависимости от метода передачи данных). Т.к. в лабораторной из формы передается большой массив данных (включая многострочный текст), то разумно использовать *POST*-форму. Следовательно, ее данные будут размещены в массиве `$_POST`. Зная имена элементов формы, можно проверить наличие в массиве элементов с соответствующими ключами. Если они присутствуют – они были переданы из формы (им просто неоткуда взяться в массиве, кроме как быть переданными из формы), а значит пользователь заполнил ее и отправил для обработки, нажав кнопку "Ок".

В данном случае достаточно проверить наличие в массиве только одного элемента с индексом "*A*", т.к. страница обрабатывает только одну форму. Если в обработчик могут быть переданы данные из нескольких форм, необходимо так подбирать имена их элементов, чтобы можно было однозначно определить из какой формы были отправлены данные. Часто для этого используют невидимые текстовые поля в которых записывают имя формы или какой-либо идентификатор.

Листинг А-6. 2

```
<form name="form_1" method="post" action="/">
      <input type="hidden" name="PROCESS" value="1">
</form>

<form name="form_2" method="post" action="/">
      <input type="hidden" name="PROCESS" value="2">
</form>
```

Итак, наличие в массиве `$_POST` элемента с ключом "*A*" говорит о том, что форма была заполнена, данные переданы в программу, а значит в массиве присутствуют и другие элементы, соответствующие полям формы. Теперь их необходимо обработать.

Листинг А-6. 3

```
if( isset( $_POST['A'] ) ) // если из формы были переданы данные
{
     if( $_POST['TASK'] == 'mean' )  // если вычисляется среднее арифметическое
     {
          $result = round( ($_POST['A']+$_POST['B']+$_POST['C'])/3, 2 );
     }
     else
     if( $_POST['TASK'] == 'perimetr' )   // если вычисляется периметр
     {
          $result = $_POST['A']+$_POST['B']+$_POST['C'];
     }
}
```

Первым шагом обработки является автоматическое решение математической задачи. Тип задачи, т.е. алгоритм вычисления результата, зависит от выбранной опции в соответствующем селекторе. Исходя из анализа переданного значения селектора с помощью ряда условных операторов, задача успешно решается выбранным алгоритмом (одним или рядом арифметических действий), а результат сохраняется в переменной `$result`. В некоторых случаях разумно округлить результата с помощью функций PHP, в данном примере это функция `round()` с округлением до двух знаков после запятой. Самостоятельно добавьте в код обработку решения других математических задач.

Обратите внимание – переменная инициируется только при решении задачи, а значит если она определена, то в программе ниже это также является признаком передачи данных из формы. Исходя из этого выводится результат обработки данных.

Причем результатом обработки может являться и сама форма! Действительно, при ошибке обработки необходимо повторно запросить данные. Или же, как в данной лабораторной работе, отсутствие данных в массиве `$_POST` означает что форма не была заполнена, а значит ее необходимо вывести заново. Или можно сказать, что результатом обработки пустых входных данных является их повторный запрос, т.е. вывод формы.

Листинг А-6. 4

```
if( isset( $result )       // если форма была обработана
{
     ...              // выводим результаты обработки
}
else                  // если форма не обработана (данные не переданы в PHP)
{
     echo '<form name="form" method="post" action="/">';  // выводим форму
     ...              // выводим теги элементов формы
```

```
     echo '</form>';
}
```

Теперь, зная результат решения задачи и все переданные из формы данные, можно довольно просто вывести требуемый отчет.

Листинг А-6. 5

```
echo 'ФИО: '.$_POST['FIO'].'<br>';   // выводим ФИО студента
echo 'Группа: '.$_POST['GROUP'].'<br>';   // выводим группу студента

if($_POST['ABOUT'] ) // если сведения "немного о себе" заполнены
       echo '<br>'.$_POST['ABOUT'].'<br>'; // выводим их

echo 'Решаемая задача: ';       // выводим тип решаемой задачи
if( $_POST['TASK'] == 'mean' ) echo 'СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ'; else
if( $_POST['TASK'] == 'perimetr' ) echo 'ПЕРИМЕТР ТРЕУГОЛЬНИКА';

if( $result === $_POST['result'] )   // если вычисления человека и машины совпали
     echo '<br><b>ТЕСТ ПРОЕДЕН</b><br>';  // выводим поздавления
else                                 // если человек ошибся
     echo '<br><b>ОШИБКА: ТЕСТ НЕ ПРОЙДЕН!</b><br>'; // ругаемся
```

Программа формирует отчет и непосредственно в процессе его формирования выводит его в HTML-код браузера. Это вполне допустимо если формируемый текст нигде более не используется, но, по условиям лабораторной работы, он должен быть отправлен по электронной почте при установке соответствующего флажка. Т.е. результат используется дважды: для отправки по почте и для вывода в браузер. Поэтому сохраним отчет в промежуточно переменной `$out_text`, которую затем и будем использовать необходимое число раз.

Листинг А-6. 6

```
$out_text='ФИО: '.$_POST['FIO'].'<br>';   // подготавливаем содержимое отчета
$out_text.='Группа: '.$_POST['GROUP'].'<br>';

if($_POST['ABOUT'] ) $out_text.='<br>'.$_POST['ABOUT'].'<br>';

$out_text.='Решаемая задача: ';
if( $_POST['TASK'] == 'mean' ) $out_text.='СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ'; else
if( $_POST['TASK'] == 'perimetr' ) $out_text.='ПЕРИМЕТР ТРЕУГОЛЬНИКА';

if($result === $_POST['result']) $out_text.='<br><b>ТЕСТ ПРОЕДЕН</b><br>'; else
                          $out_text.='<br><b>ОШИБКА: ТЕСТ НЕ ПРОЙДЕН!</b><br>';

echo $out_text;      // выводим отчет в браузер

if( array_key_exists('send_mail', $_POST) )     // если нужно отправить результаты
{
     // отправляем результаты по почте простым письмом
     mail( $_POST['MAIL'], 'Результат тестирования',
            str_replace('<br>', "\r\n", $out_text),
            "From: auto@mami.ru\n"."Content-Type: text/plain; charset=utf-8\n" );

     // выводим соответствующее сообщение в браузер
     echo 'Результаты теста были автоматически отправлены на e-mail '.$_POST['MAIL'];
}
```

В представленной программе отчет сначала формируется, а выводится в браузер только после полного завершения этого процесса. Для отправки письма с отчетом необходимо проверить установку соответствующего флага. Напомним, что если флаг установлен, то он передается в обработчик формы, если нет – то не передается. Поэтому признаком установки флага является наличие элемента массива `$_POST` с совпадающим с именем флага ключом. В программе проверяется это условие и, при его выполнении, формируется и отправляется письмо.

В примере письмо отправляется обычным текстом, а не *HTML*-кодом, поэтому необходимо заменить все теги перевода строки `<br>` на соответствующие символы (символы "Возврат каретки" и "Перевод строки"). После отправки выводится требуемая надпись.

Для завершения основной части PHP кода также необходимо сформировать *HTML*-код кнопки, при нажатии которой будет повторно отображена форма, а тестирование можно будет провести заново.

Листинг А-6. 7

```
echo '<a href="?F='.$_POST['GROUP'].'&G='.$_POST['GROUP'].
     '" id="back_button">Повторить тест</a>';
```

Кнопка формируется в виде ссылки, оформление которой осуществляется в CSS-файле. Т.к. после перехода по ссылке будет проводиться повторное тестирование, то, согласно условиям лабораторной работы, поля формы с ФИО и группой студента уже должны быть заполнены: для этого в адрес ссылки передаются соответствующие GET-параметры. При выводе самой формы довольно просто проверить существование указанных ключей в массиве `$_GET` и, если они там есть, вывести элементы массива как значения полей.

Самостоятельно дополните программу обработкой других видов задач, дополнением отчета входными данными, вычисленным и переданным результатом вычисления. При этом, если предполагаемый результат не был передан (пустая строка) – должна формироваться надпись: "Задача самостоятельно решена не была".

Самостоятельно разделите способ предоставления данных "Для отображения в браузере" и "Для печати" так, чтобы они были одинаковы по содержанию, но разные по внешнему виду. Кнопку "Повторить тест" необходимо выводить только в первом случае.

Самостоятельно сформируйте форму и назовите ее элементы в соответствии с используемыми в рассмотренных выше примерах ключами массива `$_POST`. Дополните форму соответствующим кодом *JavaScript* для выполнения на стороне клиента (*JavaScript* выводится методами *PHP*, как и любой другой текст *HTML*-кода страницы). Добавьте в код формы вывод произвольных чисел как начальные значения полей *A*, *B* и *C*, а также вывод ФИО и группы студента, если страница была сформирована повторно.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Получение произвольного числа	`$some_val = mt_rand(5,100);`
	Функция `mt_rand()` возвращает случайное целое число в указанном диапазоне (в данном примере от 5 до 100). Если необходимо получить случайное вещественное число, можно использовать следующий код PHP:
	`$some_val =  mt_rand(500,10000)/100;`

Замена подстроки в строке	`$s = str_replace ( $search, $replace, $str );` Функция заменяет в строке `$str` все подстроки `$search` на строку `$replace`. Подробнее о функции – см. в справочнике.
Получение данных о переданных в документ GET- и POST-параметрах.	`$_GET['el_name']` – значение элемента *GET*-формы с именем "*el_name*". Такие параметры также можно передавать и через *URL*. `$_POST['el_name']` – значение элемента *POST*-формы с именем "*el_name*".
Отправка сообщения по электронной почте	`mail($adress, $subject, $message, $headers);` Отправляет по адресу `$adress` сообщение с темой `$subject` и текстом `$message`. Заголовки `$headers` указывают как именно письмо должно отображаться в почтовой программе, могут использоваться и для других целей. Подробнее о функции – см. в справочнике.
Скрыть и показать элемент в	`<input type="checkbox" name="send_mail" onClick="`

зависимости от состояния флажка (код JavaScript)	`obj=document.getElementById('object');`
	`if( this.checked )`
	`  obj.style.display='block';`
	`else`
	`  obj.style.display='none';">`
	
	`<div id="object">...</div>`
	Непосредственно в теге флажка определяем обработчик события нажатия на кнопку мыши. В событии проверяем установлен ли флажок. Если да, то отображаем объект с помощью изменения стиля. Если нет – то скрываем его.

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Что такое форма?
2. Какие типы форм бывают?
3. В чем отличие POST- и GET-форм?
4. Какие элементы формы Вы знаете?
5. Как передает свое значение каждый элемент формы в PHP-программу?
6. Как округлить вещественное число до определенного количества разрядов после запятой?
7. Сколько символов может максимально содержать строка в которой буферизуется HTML-код?
8. Как отправить сообщение по электронной почте средствами PHP?
9. Почему в листинге А-6.5 при сравнении переданного и вычисленного результата используется оператор эквивалентности?
10. На какую функцию можно заменить array_key_existst() в листинге А6. 6?
11. Можно ли в листинге А6. 6 буферизовать сообщение об отправки письма в переменной $out_text? Если да – то как это сделать и будет ли такой код более оптимальным?
12. Какие события могут обрабатываться в JavaScript? Какими способами можно задать обработчик события для объекта?

Основы использования массивов в программировании. Ввод данных и сортировка массивов.

Лабораторная работа № А-7.

ЦЕЛЬ РАБОТЫ

Закрепление знаний о работе с одномерными массивами в PHP, получение навыков алгоритмического мышления и способах отладки программы на PHP.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа (2 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу http документы, обеспечивающие возможность пользователю задать произвольной длинны массив чисел и отсортировать его различными алгоритмами с выводом состояния массива на каждом шаге работы.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Работа выполняется в виде двух *php*-файлов: для ввода массива и для отображения процесса его сортировки. Первый файл содержит статически заданную форму, содержащую следующие элементы.

- Одно поле для ввода значения элемента массива.
- Селектор с опциями:
  - сортировка выбором;
  - пузырьковый алгоритм;
  - алгоритм Шелла;
  - алгоритм садового гнома;
  - быстрая сортировка;
  - встроенная функция PHP для сортировки списков по значению.
- Кнопки:
  - добавить еще один элемент;
  - сортировать массив.

При нажатии кнопки "Добавить еще один элемент" на форме с помощью *JavaScript*, без перезагрузки страницы, добавляется еще одно поле для ввода элемента массива (с номером элемента слева от него). При нажатии кнопки "Сортировать массив" в отдельном окне (вкладке) загружается вторая страница, в которой данные формы обрабатываются.

В качестве результата обработки страница должна содержать название алгоритма сортировки, входные данные (введённый массив чисел), результат проверки валидности входных данных (все ли элементы массива числа). Если среди элементов массива есть не числа – сортировка не выполняется, вместо нее выводится соответствующее предупреждение. Если входных данных нет – также выводится предупреждение, сортировка не выполняется.

В процессе сортировки выводится информация о каждой итерации алгоритма:

- номер итерации (сквозная нумерация даже для вложенных циклов);
- текущее состояние массива.

В конце работы алгоритма выводится надпись: "Сортировка завершена, проведено *N* итераций. Сортировка заняла *T* секунд". (где *N* – количество проделанных итераций, *T* – время выполнения сортировки в секундах или долях секунды).

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Первый файл, несмотря на расширение ".*php*", не содержит *PHP*-кода, зато содержит код *JavaScript*. Он необходим для формирования произвольного количества полей с элементами массива. Такой прием часто используется в экранных формах, когда число строк (элементов) в форме заранее неизвестно. JavaScript позволяет реализовать такую функцию различными способами, рассмотрим в качестве примера один из них.

Листинг А-7. 1

```
<script>
       function addElement(table_name, amount)  // функция добавляет еще один элемент
       {
           var t = document.getElementById(table_name); // объект таблицы

           for(var i=0; i<amount; i++)
           {
                var index=t.rows.length;        // индекс новой строки
                var row=t.insertRow(index); // добавляем новую строку

                var cel = row.insertCell(0);      // добавляем в строку ячейку
                cel.className='element_row';      // определяем css-класс ячейки

                // формируем html-код содержимого ячейки
                var celcontent='<input type="text" name="element0">';

                // добавляем контент в ячеку таблицы
                setHTML(cel, celcontent);
           }
           // в скрытом поле записываем количество полей (строк таблицы)
           document.getElementById('arrLength').value=t.rows.length;
       }
</script>

<table id="elements">
     <tr><td class="element_row"><input type="text" name="element0"></td></tr>
</table>

<input type="hidden" id="arrLength">
<input type="button" value="Добавить еще один элемент"
       onClick="addElement('elements', 1);">
```

Статический HTML-код на представленном листинге формирует согласно требованиям лабораторной работы кнопку и одно поле для ввода элемента массива. Реализованный в представленной *JavaScript*-программе подход к решению задачи предполагает размещение этого поля в таблице. Тогда каждый раз при нажатии кнопки, с помощью разработанной функции `addElement()` в таблицу добавляется еще одна строка с еще одним полем. Рассмотрим работу указанной функции более подробно.

Обратите внимание, что в качестве аргумента в функцию была передана строка с *id* таблицы, а не соответствующий ей объект. Поэтому первая строка тела функции с помощью `getElementById()` определяет соответствующий таблице объект по ее имени. В дальнейшем работа JavaScript-программы происходит уже именно с этим объектом.

Вторым аргументом функции, введенным для универсальности, является `amount` – число добавляемых в таблицу строк (полей). Т.к. в лабораторной работе требуется при нажатии кнопки добавление только одной строки, то при вызове функции этот аргумент равняется одному. В теле функции этот аргумент используется как предел цикла со счетчиком – в этом цикле последовательно на каждой итерации в таблицу добавляется одна строка. Следовательно, в таблицу будет добавлено ровно столько строк, сколько указано в переменной `amount`.

Сам процесс добавления строки происходит следующим образом. Сначала определяется количество уже присутствующих в таблице строк и сохраняется в переменной `index`. Затем с помощью функции `insertCell()` на последнее место в таблице добавляется новая строка. Ее

номер будет соответствовать количеству строк в таблице до ее добавления (нумерация начинается с нуля), т.е. значению в переменной `index`. Обратите внимание: если в качестве аргумента функции `insertCell()` указать ноль – строка будет добавлена на первом месте, если `index-1` – то на предпоследнем.

Добавление в таблицу строки не означает автоматическое добавление в них ячеек – это необходимо сделать явно с помощью функции `insertCell()`. Получив таким образом объект, идентифицирующий ячейку, мы можем изменять ее свойства (например, присвоить имя *CSS*-класса) и определять ее содержимое. Контент ячейки предварительно формируется в переменной `celcontent`, а затем добавляется в нее с помощью пользовательской функции `setHTML()`.

Обратите внимание – каждая строка будет содержать поле с одним и тем же именем. Чтобы обработчик формы получил все элементы массива имена полей должны отличаться. Самостоятельно доработаете JavaScript-программу так, чтобы итоговые поля назывались "*element*X", где *X* – номер поля по порядку начиная с нуля.

В конце программы общее количество строк таблицы, а значит общее количество полей с элементами массива, а значит и длина массива будет присвоена значению скрытого поля с *id*="*arrLength*" – это несколько упростит дальнейшую обработку массива *PHP*-программой из второго файла.

Для завершения первой части лабораторной работы необходимо лишь разработать код функции `setHTML()`, которая бы для указанного объекта (в данном случае для ячейки таблицы) устанавливала бы внутреннее содержимое (например, для объекта-блока с *id* "*block*", заданного кодом `<div id="block">This is the innerContent</div>` внутренним содержимым будет строка "*This is the innerContent*"). В JavaScript у объектов есть свойство `innerHTML`, присвоение которому *HTML*-кода и означает требуемую операцию. Но, к сожалению, оно доступно не во всех браузерах, поэтому и есть необходимость определить подобную функцию.

Листинг А-7. 2

```
function setHTML(element, txt)
{
	if(element.innerHTML)
		element.innerHTML = txt;
	else
	{
		var range = document.createRange();
		range.selectNodeContents(element);
		range.deleteContents();
		var fragment = range.createContextualFragment(txt);
		element.appendChild(fragment);
	}
}
```

Представленный код первой страницы практически закончен. Но для выполнения условий лабораторной работы самостоятельно модифицируйте *HTML*-код и *JavaScript*-программу так, чтобы слева от поля в отдельной колонке выводился номер (ключ) элемента массива; также модифицируйте и упростите функцию `addElement()` так, чтобы она всегда добавляла в таблицу только одну строку; добавьте в статический *HTML*-код теги формы и ее не указанных элементов.

Второй файл, содержащий PHP-программу для обработки данных, также достаточно прост. Сформулируем словесное описание алгоритма следующим образом.

1. Если данные не переданы в обработчик – вывести сообщение, прекратить работу.
2. Если среди переданных элементов есть не число – вывести сообщение, прекратить работу.
3. Выбрать алгоритм сортировки.
4. Вывести название алгоритма, входные данные, сообщение о прохождении валидации элементов массива.
5. Сохранить текущее время.
6. Провести сортировку массива выбранным алгоритмом с выводом информации о ходе алгоритма.
7. Засечь разницу между текущим временем и сохраненным ранее.

8. Вывести сообщение о завершении алгоритма, количество итераций алгоритма, затраченное время.

Переведем описание на русском языке на язык *PHP*. Для простоты описания будем считать, что все алгоритмы сортировки уже известны и определяются номером от нуля до одного (для краткости рассмотрим вариант с двумя алгоритмами). Тогда функции `sort_0()`и `sort_1()` сортируют массив и выводят информацию о ходе процесса по соответствующим алгоритмам.

Листинг А-7. 3

```
if( !isset($_POST['element0']) )     // если данных нет
{
     echo 'Массив не задан, сортировка невозможна'; // сообщение
     exit();                                        // и завершение программы
}

for($i=0; $i<$_POST['arrLength']; $i++)              // для всех элементов массива
     if( arg_is_not_Num( $_POST['element'.$i] ) )    // если элемент массива не число
     {
          // выводим сообщение и завершаем программу
          echo 'Элемент массива "'. $_POST['element'.$i].'" - не число';
          exit();
     }

// определим реализуемый алгоритм и выведем его название
if( $_POST['algoritm']==='0' )
     echo '<h1>Алгоритм 0</h1>';
else
if( $_POST['algoritm']==='1' )
     echo '<h1>Алгоритм 1</h1>';

$arr = array(); // создаем пустой массив для формирования сортируемого списка
echo 'Исходный массив<br>----------------------------<br>';

for($i=0; $i<$_POST['arrLength']; $i++)         // для всех элементов массива
{
      echo '<div class="arr_element">'.$i.': '.
           $_POST['element'.$i].'</div>'; // выводим текущий элемент и его номер

     $arr[] = $_POST['element'.$i]; // добавляем элемент в массив для сортировки
}

// сообщение об успешной валидации
echo '<br>----------------------------<br>Массив проверен, сортировка возможна';

$time = microtime(true); // засекаем время начала сортировки
if( $_POST['algoritm']==='0' )
     $n = sort_0( $arr );       // запускаем сортировку по первому алгоритму
else
if( $_POST['algoritm']==='1' )
      $n = sort_1( $arr );          // запускаем сортировку по второму алгоритму

// выводим сообщение о завершении сортировки
echo 'Сортировка завершена, проведено '.$n.' итераций. ';

// считаем и выводим затраченое на сортировку время
echo 'Затрачено '.($time- microtime(true)).' микросекунд!';
```

Приведенную программу вполне можно оптимизировать, но в приведенном виде она довольно точно соответствует описанному алгоритму. Итак, в первых строках программы осуществляется проверка наличия в массиве с *POST*-параметрами первого (нулевого) элемента массива. Если его нет – это означает что данная страница была вызвана не как результат работы формы, а просто через *URL* без каких-либо *GET*- или *POST*-параметров. В этом случае обработка невозможно, о чем и выводится соответствующее предупреждение.

Далее необходимо проверить все переданные элементы массива на их соответствие целому числу. Использовать функцию `is_integer()` в этом случае нельзя – параметры передаются в массив как строки. Напишем собственную функцию `arg_is_not_Num()`, которая возвращает `true`

если аргумент не целое число (рассмотрим эту функцию более подробно ниже в листинге А-7.4). Тогда, перебирая в цикле все полученные параметры с элементами массива и проверяя их этой функцией мы осуществим валидацию входных данных.

Обратите внимание: мы точно знаем сколько элементов массива передано для обработки, т.к. передали их количество через скрытое поле с именем "*arrLength*". Кроме того, нет необходимости всегда перебирать все элементы – первый же невалидный элемент массива останавливает работу программы.

Далее, в программе определяется выбранный через селектор тип алгоритма, который и выводится в *HTML*-коде. Для вывода исходного массива (входных данных) также используется цикл: все элементы "оборачиваются" в блок `<div>` – определив *CSS*-стиль для этого блока набор элементов массива можно вывести очень компактно и наглядно (используйте свойства `float`, `margin`, `padding`). Параллельно в этом же цикле элементы массива вынимаются из ассоциированного массива `$_POST` с ключами в виде строк и добавляются в специально созданные ранее список `$arr`.

Теперь осталось собственно отсортировать массив, что и происходит с помощью вызова соответствующей функции. Предварительно в переменной `$time` засекается системное время (с точностью до микросекунды) до начала выполнения алгоритма. Разница между системным временем после выполнения алгоритма и значением переменной `$time` и будет определять длительность выполнения сортировки. Число итераций алгоритма будем брать как результат работы соответствующей функции.

Листинг А-7. 4

```
function arg_is_not_Num( $arg )
{
     if( $arg!=='' ) return true;    // передана пустая строка

     for($i=0; $i<strlen($arg); $i++)  // цикл по всем символам аргумента
          if( $arg[$i]!=='0' && $arg[$i]!=='1' && $arg[$i]!=='2' &&
              $arg[$i]!=='3' && $arg[$i]!=='4' && $arg[$i]!=='5' &&
              $arg[$i]!=='6' && $arg[$i]!=='7' && $arg[$i]!=='8' &&
              $arg[$i]!=='9' ) // если встретилась не цифра
                   return true; // возвращаем true

      // строка состоит из чисел - возвращаем false
      return false;
}
```

Работу используемой выше функции `arg_is_not_Num()` можно реализовать следующим образом. В начале происходит проверка на пустую строку, которая разумеется не является числом. Обратите внимание – в данном случае нельзя проверять условие (`$arg`) – необходимо явное сравнение аргумента с пустой строкой. Это следует делать из-за того, что PHP достаточно вольно преобразует типы. И уверенность в том, что строка "*0*" не будет преобразована сначала в число ноль (при передаче в качестве аргумента функции), а затем и в `false`, отсутствует.

Далее в цикле перебираются все символы строки: если среди них встретился отличный от цифры символ – работа функции завершается, возвращается `true` (аргумент не число). Если все символы аргумента прошли проверку – возвращается `false`. Функцию вполне можно упростить с помощью регулярных выражений, которые будут рассматриваться в последующих лабораторных работах.

Для завершения лабораторной работы самостоятельно переименуйте соответствующие сообщения и функции сортировки согласно названию алгоритмов и напишите их *PHP*-код (используйте раздел "Справочная информация" этой лабораторной работы). Добавьте в программу стандартную функцию сортировки массивов (без учета количества итераций). Сравните время работы реализованных вами функций и встроенной функции *PHP,* сделайте выводы.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Сортировка массива по значению элементов	`sort($arr); // в порядке возрастания`
	`rsort($arr); // в порядке убывания`
	`assort($arr); // по возрастанию с сохранением ключей элементов; для ассоциированных массивов`
	`arsort($arr); // по убыванию с сохранением ключей элементов;для ассоциированных массивов`
	Функции возвращают true в случае успешной сортировки и false в случае ошибки.

Досрочное прекращение выполнения PHP-программы	`exit();` Прекращает выполнение PHP и дальнейшую передачу статического кода в браузер. Фактически в HTML-коде браузера будет только то, что было передано в него (неважно, статически или с помощью PHP) до вызова функции. Удобно использовать для зашиты сайтов от попыток взлома и для прекращения формирования страницы при некорректных данных.
Получение системного времени	`$time = $microtime(true);` Функция `microtime()` возвращает текущую метку веремени *Unix*. Если необязательный аргумент указан и равен `true` – время возвращается в виде вещественного числа, обозначающего секунды с точностью до микросекунды. Если аргумент не указан или равен `false` – метка времени имеет формат "*msec sec*".

АЛГОРИТМ СОРТИРОВКИ МАССИВА ВЫБОРОМ

Сортировка выбором, пожалуй, самый простой алгоритм. Его суть заключается в том, что мы находим в массиве минимальный элемент и меняем его местами с первым элементом массива. Затем ищем минимальный элемент уже начиная со второго и меняем его местами со вторым. И так далее, пока элементы массива не закончатся.

Другими словами, можно представить себе отсортированную и неотсортированную часть массива. В начале работы отсортированной части нет, т.к. весь массив не отсортирован. После первой итерации, когда на первом месте размещен минимальный элемент, можно сказать что отсортированная часть – первый элемент массива, а неотсортированная – массив начиная со второго элемента. После второй, когда найден минимальный элемент неотсортированной части предыдущей итерации, отсортированная часть состоит уже из двух элементов массива. На третьей – из трех и т.д.

Обратите внимание, что нет необходимости проводить интеграции для всех элементов массива: когда неотсортированная часть будет состоять из одного элемента – она автоматически станет также отсортированной (массив из одного элемента отсортирован всегда). Тогда словесное описание такого алгоритма можно сформулировать следующим образом.

В цикле для всех элементов массива, кроме последнего делаем следующие итерации.

- Начиная текущего ищем минимальное значение элемента массива и запоминаем его индекс.
- Если найденный индекс больше текущего, т.е. минимальный элемент не располагается в самом начале неотсортированной части массива – меняем местами текущий элемент и элемент с найденным индексом.

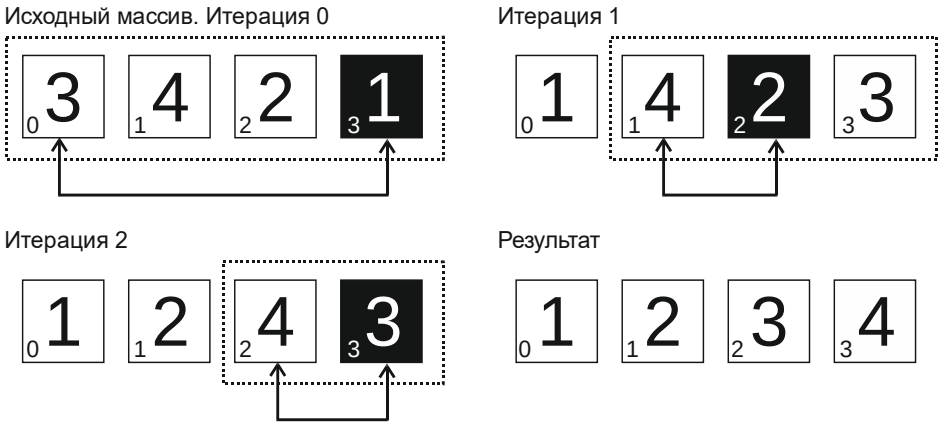
Запишем этот алгоритм с помощью PHP в виде отдельной функции `sorting_by_choice()`.

Листинг А-7. 4

```
function sorting_by_choice( $arr ) // функция сортировки выбором
{
	for($i=0; $i<count($arr)-1; $i++) // цикл для всех элементов списка
	{
		// неотсортированной частью массива считаем элементы начиная от текущего
	    $min=$i;

	    for($j=$i+1; $j<count($arr); $j++) // ищем минимальный элемент
	              if( $arr[$j]<$arr[$i] ) $min=$j; // в неотсортированной части

	    if( $min > $i+1 )     // если минимальный элемент не первый в
	    {                 // неотсортированной части массива
	         $element = $arr[$i];       // меняем его с первым
	         $arr[$i]=$arr[$min];
	         $arr[$min] = $element;
	    }
	}
	return $arr;
}
```

В качестве аргумента функции передается неотсортированный массив. В теле функции запускается цикл по всем элементам массива, кроме последнего. Если массив пустой или содержит только один элемент – ничего делать не надо, массив уже отсортирован. В противном случае на каждой итерации цикла определяется индекс минимального элемента массива начиная с текущего. Для этого в начале установим текущий элемент как минимальный, т.е. элемент с индексом `$i`. Затем в цикле переберём все последующие за ним элементы – если встретился меньший, то уже он становится минимальным, а его индекс сохраняется в переменной `$min`.

Если в неотсортированной части массива минимальный элемент уже стоит на первом месте – ничего делать не надо. Если же нет, т.е. если найденный индекс больше текущего индекса – два элемента меняются местами. Рассмотрим работу этого алгоритма для небольшого массива.



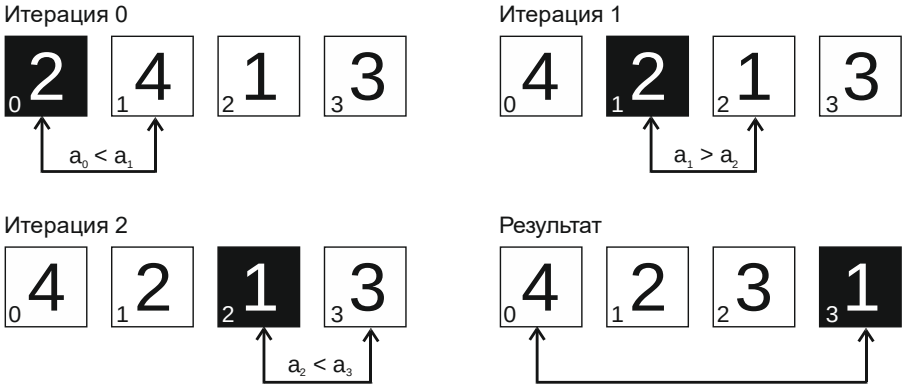
На рисунке пунктирной рамкой обведена неотсортированная часть массива. В начале первой итерации весь массив не отсортирован. Ищется минимальный элемент среди неотсортированной части – в данном случае это элемент с индексом 3 и значением 1. Т.к. индекс 3 больше текущего элемента 0 – они меняются местами. На второй итерации неотсортированная часть массива уже меньше, минимальный элемент расположен по ключ у 2. Меняем его с элементом с индексом 1. Аналогично проводим третью и последнюю итерацию и на выходе получаем полностью отсортированный массив.

Рассмотренный алгоритм очень хорошо, а порой и более эффективно чем его более сложные и продвинутые аналоги, работает для небольших массивов. Но для больших массивов алгоритм всегда проигрывает им.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Пузырьковый алгоритм сортировки массива

Самый известный алгоритм, изучаемый обычно еще в школе. Его суть заключается в том, что более "легкие" элементы "всплывают" наверх – отсюда и название по аналогии с пузырями в воде. Для этого все элементы последовательно в цикле сравниваются с соседним: если элемент меньше следующего, то они меняются местами.



Проведя эту операцию в цикле для всех элементов массива окажется, что самый "лёгкий" элемент находится на последнем месте. Например, нам необходимо отсортировать массив (2, 4, 1, 3). Рассмотрим работу алгоритма. На нулевой итерации берется элемент с индексом 0 и сравнивается со следующим, т.е. с индексом 1. Т.к. два меньше четырех, то элементы меняются местами – двойка "всплывает" вверх, а четверка "оседает" вниз.

На следующей итерации рассматриваем уже индексы 1 и 2. Обратите внимание: раньше элемент с индексом 1 имел значение 4, но после первой итерации оно поменялось на 2. Поэтому сравнивается элементы 2 и 1: единица "легче" (меньше) двойки, поэтому ничего не происходит. На второй итерации сравниваются значения 1 и 3, после нее элемент со значением 1 "всплывает" на самый верх массива: именно этого мы и добивались.

Таким образом на каждой итерации цикла рассматривается элемент с индексом номера итерации и следующий за ним, т.е. номер итерации плюс один. А для полного прохода по всему массиву необходимо *N*-1 итераций, где *N* – длинна массива. Поэтому для пустого массива или массива с одним элементом сортировка не производится – он уже априори отсортирован. Опишем приведенный алгоритм с помощью естественного языка следующим образом.

В цикле со счетчиком *i* от 0 до *N*-2 (где *N* – длина массива) делаем:

- если элемент *a*[*i*] меньше *a*[*i*+1] – меняем местами их значения.

Переведем описание алгоритма с русского языка на PHP.

Листинг А-7. 5

```
for($i=0; $i<=count($arr)-2; $i++)    // для элементов от первого до предпоследнего
     if( $arr[$i]<$a[$i+1] )          // если текущий элемент меньше следующего
     {
          $temp = $arr[ $i ];         // меняем их местами элементы i и i+1
          $arr[ $i ] = $arr[ $i+1 ];
          $arr[ $i+1 ] = $temp;
      }
}
```

В программе организован цикл со счетчиком от нуля до размера массива минус два. Т.к. нумерация элементов ведется от нуля, то последний рассматриваемый элемент будет иметь индекс *N*-2, т.е. будет предпоследним элементом массива. Сравниваться он будет с элементом (*N*-2)+1 = *N*-1, т.е. с последним элементом. Далее цикл прекращается, т.к. сравнивать последний элемент уже не с чем.

Обратите также внимание на условие в цикле: обычно используется сравнение "меньше", а не "меньше или равно". Действительно, сейчас условие было записано таким образом только для

того, чтобы дословно перевести с русского на PHP. После оптимизации программу можно записать так.

Листинг А-7. 6

```
for($i=0; $i<count($arr)-1; $i++)     // для элементов от первого до предпоследнего
      if( $arr[$i]<$a[$i+1] )         // если текущий элемент меньше следующего
      {
            $temp = $arr[ $i ];       // меняем их местами
            $arr[ $i ] = $arr[ $i+1 ];
            $arr[ $i+1 ] = $temp;
       }
}
```

Перестановка элементов осуществляется если элемент с индексом *i* меньше элемента с индексом *i*+1. Для этого вводится дополнительная переменная `$temp`, в которой сохраняется значение элемента *i*. Если этого не сделать, а сразу присвоить *i*-ому элементу значение *i*+1, то информация о его значении пропадет: оба элемента будут иметь одинаковое значение. А так для окончания перестановки элементу *i*+1 присваивается сохраненное в `$temp` значение *i*-ого.

Вернемся к массиву (2, 4, 1, 3): в результате выполнения разработанной программы он превратится в массив (4, 2, 3, 1) который все еще не до конца отсортирован. Но обратите внимание: после одной итерации один последний элемент массива находится на своем месте. Рассмотрим его и остальные элементы отдельно: (4, 2, 3) и (1). Левый массив содержит более "тяжелые" элементы, чем правый и не отсортирован. Логично предположить, что, если мы его отсортируем и соединим левую и правую часть обратно – то итоговый массив будет полностью отсортирован.

Поэтому применим к левой части уже описанный алгоритм – в результате элемент 2 "всплывет" наверх. Его можно присоединить к массиву (1), т.к. элемент 2 – самый легкий из всех элементов, которые "тяжелее" 1 и его место в начале массива: (2, 1). Остальные два элемента (4, 3) тоже надо провести через алгоритм – этот массив тоже надо упорядочить несмотря на то, что нам кажется очевидным что он упорядочен. На самом деле Вы только что выполнили этот алгоритм самостоятельно в уме: сравнили первый и следующий за ним элементы, убедились, что первый меньше второго и вышли из цикла.

В итоге за три вызова алгоритма, рассматривая при каждом вызове лишь часть массива на *j* меньшую его длины (где *j* – номер вызова), мы получаем полностью отсортированный массив. Функцию, которая выполняет эту операцию, можно представить следующим образом.

Листинг А-7. 7

```
function BubbleSort($arr)
{
      // число повторов: длина массива - 1
      for($j=0; $j<count($arr)-1; $j++)
      {
            // для всех элементов рассматриваемой части массива
               // от нулевого до предпоследнего
               for($i=0; $i<count($arr)-1-$j; $i++)
            if( $arr[$i]<$a[$i+1] )    // если текущий элемент меньше следующего
            {
                  $temp = $arr[ $i ];        // меняем их местами
                  $arr[ $i ] = $arr[ $i+1 ];
                  $arr[ $i+1 ] = $temp;
               }
       }
       return $arr;
}
```

Обычно при реализации для счетчиков циклов используется сначала переменная `$i`, а уже затем `$j` – с точки зрения PHP это не имеет значение. В примере сначала используется `$j`, а потом `$i` только исходя из очередности подачи материала для изучения.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Алгоритм шейкерной сортировки массива

Шейкерная сортировка – немного модифицированный вариант сортировки пузырьком. Если в нем перебор массива осуществляется только в одну сторону, то здесь последовательно идут обход с начала неотсортированной части массива с выбором максимального элемента и обход с конца неотсортированной части с выбором н элемента. Поэтому такой алгоритм называется также "двунаправленная пузырьковая сортировка.

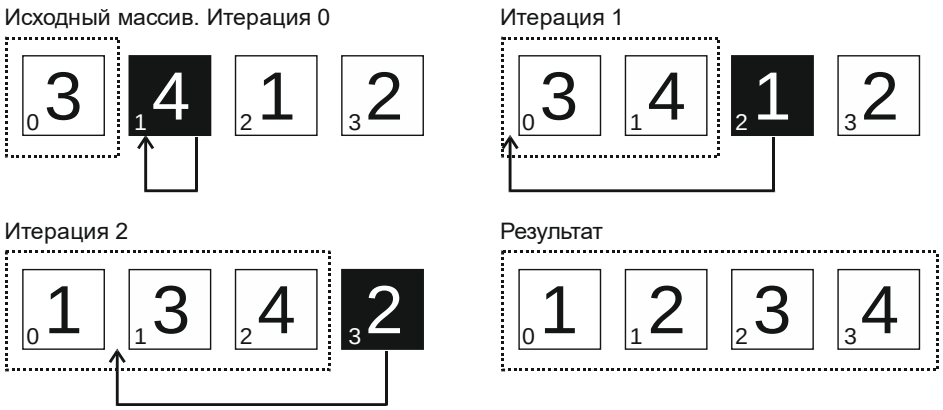
Листинг А-7. 8

```
function ShakerSort($arr)
{
      $left = 1;                  // слева-направо с первого элемента массива
      $right = count($arr)-1;          // справа-налево с последнего
      while( $left <= $right )         // пока границы не сойдутся
      {
            // обход аналогично пузырьковому алгоритму с правой границы до левой
            for($i=$right; $i>=$left; $i--)
                  if( $arr[$i-1]>$arr[$i] )  // если предыдущий элемент больше
                  {
                        $temp=$arr[$i-1];     // меняем его с текущим местами
                        $arr[$i-1]=$arr[$i];
                        $arr[$i]=$temp;
                  }
            $left++;   // сдвигаем левую границу вправо
            // обход аналогично пузырьковому алгоритму с левой границы до правой
            for($i=$left; $i<=$right; $i++)
                  if( $arr[$i-1]>$arr[$i] )  // если предыдущий элемент больше
                  {
                        $temp=$arr[$i-1];     // меняем его с текущим местами
                        $arr[$i-1]=$arr[$i];
                        $arr[$i]=$temp;
                  }
            $right--;  // сдвигаем правую границу влево
       }
       return $arr;
}
```

В начале алгоритма устанавливаются две границы с уже отсортированными частями массива: левая и правая. До тех пор, пока эти границы не сойдутся массив нуждается в сортировке: именно это условие используется в цикле с предусловием.

Первый вложенный цикл начинает обход от правой границы неотсортированной части массива и до его левой границы. В первый раз срабатывания этого цикла самый "легкий" элемент будет перемещен в самое начало массива. Во второй раз – самый легкий из оставшихся на второе место, т.е. левая отсортированная часть массива увеличивается. Поэтому левая граница неотсортированной части массива после окончания цикла сдвигается вправо. Аналогично работает и следующий цикл, проходящий массив слева на право.

Алгоритм сортировки массива вставками



Смысл алгоритма заключается во вставке каждого элемента массива на свое место в уже отсортированной части массива. На первой итерации такой частью является только первый элемент начального массива; текущим элементом – второй. Он сравнивается с первым и либо остается на своем месте, либо переставляется в начало массива. На втором шаге берется третий элемент и сравнивается со всеми предыдущими до тех пор, пока не подобран первый из них, который меньше его: тогда он вставляется сразу за ним.

Листинг А-7. 9

```
function InsertSort($arr)
{
      for($i=1; $i<count($arr); $i++) // для всех элементов начиная с первого
      {
            $val = $arr[$i];           // сохраняем значение текущего элемента
            $j = $i-1;            // начинаем перебор с предыдущего элемента

               // пока не найден элемент меньше текущего
            while( $j>=0 && $arr[$j]>$val )
            {
                  $arr[$j+1] = $arr[$j];     // сдвигаем элементы массива вправо
                  $j--;
            }
            $arr[$j+1] = $val;         // вставляем текущий элемент на свое место
      }
      return $arr;
}
```

В основном цикле осуществляется перебор всех элементов массива, начиная со второго. Если в массиве нет элементов или он только один – то цикл не выполняет ни одной итерации, т.е. такой массив уже отсортирован. Если же элементов два и более – то значение текущего запоминается в переменной `$val`. Далее в теле цикла осуществляется перебор всех предыдущих элементов, т.е. перебор элементов отсортированной части массива, до тех пор, пока не встретился элемент меньший текущего. Для этого удобно использовать цикл с предусловием; дополнительным условием цикла является существование в массиве элемента с рассматриваемым индексом, т.е. `$j>=0` – это важно, т.к. иначе существует риск бесконечного цикла.

На каждой итерации этого вложенного цикла элементы массива сдвигаются вправо, освобождая место для вставки текущего элемента из переменной `$val`. Рассмотрим пример: пусть текущим состоянием массива после двух итераций будет (1, 3, 4, 2), т.е. первые три элемента уже отсортированы. Тогда, в переменной `$val` сохраняется значение 2; перебор во вложенном цикле начинается с элемента со значением 4. Массив, во время работы этого цикла будет изменяться следующим образом: (1, 3, 4, 2) -> (1, 3, 4, 4) ->(1, 3, 3, 4), после чего цикл закончится на элементе со значением 1, т.к. оно меньше значения 2. Сразу после цикла после этого элемента будет вставлен текущий и массив будет полностью упорядочен: (1, 2, 3, 4).

Алгоритм садового гнома

По мнению голландского ученого Дика Груна обычные садовые гномы очень любят поддерживать порядок в их саду. Поэтому, если вечером расставить горшки с цветами, то утром они обязательно будут выстроены в идеальном порядке. При этом гномы, как очень умные и трудолюбивые существа, выработали свой собственный алгоритм сортировки: гном смотрит на предыдущий и текущий горшки. Если они в правильном порядке, то гном шагает вперед к следующему горшку, если нет – он меняет их местами и делает шаг назад. Начинает работу гном всегда со второго горшка; если предыдущего горшка нет – гном шагает вперед; если следующего горшка нет – гном заканчивает свою работу и таинственно исчезает.

Алгоритм очень прост в реализации и не содержит встроенных циклов, но вместе с тем очень перегружен. Левая, уже отсортированная часть массива, в определенный момент времени может оказаться неотсортированной и сравнения с перестановками придется повторять заново.

Листинг А-7. 10

```
function gnomeSort($arr)
{
      $i=1;                 // начинаем со второго элемента массива
      while( $i<count($arr) )    // пока не достигнут последний элемент - цикл
      {
            // если первый элемент массива (предыдущего нет)
               // или текущий элемент больше предыдущего
            if( !$i || $arr[$i-1]<=$arr[$i] )
                  $i++;                     // шагаем вперед
            else       // иначе
            {
                  $temp = $arr[$i];          // меняем элементы местами
                  $arr[$i] = $arr[$i-1];
                  $arr[$i-1] = $temp;
                  $i--;                // шагаем назад
            }
      }
      return $arr;
}
```

Приведённый алгоритм может быть оптимизирована за счет возврата к движению вперед с того места, откуда началось движение назад.

Листинг А-7. 11

```
function gnomeSort($arr)
{
      $i=1;                 // начинаем со второго элемента массива
      $j=2;
      while( $i<count($arr) )    // пока не достигнут последний элемент - цикл
      {
            // если первый элемент массива (предыдущего нет)
               // или текущий элемент больше предыдущего
            if( !$i || $arr[$i-1]<=$arr[$i] )
            {
                  $i=$j;                          // возвращаемся к месту
                  $j++;                     // до которого уже дошли
            }
            else       // иначе
            {
                  $temp = $arr[$i];          // меняем элементы местами
                  $arr[$i] = $arr[$i-1];
                  $arr[$i-1] = $temp;
                  $i--;                // шагаем назад
            }
      }
      return $arr;
}
```

Для этого вводится переменная `$j`, которая сохраняет индекс массива до начал движения назад. Когда вновь начинается движение вперед, то уже рассмотренные элементы массива не сравниваются: сравнение начинается с нерассмотренной части массива.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Алгоритм Шелла сортировки массива

Улучшенный вариант сортировки массива вставками, предложенный Дональдом Шеллом. Его идея заключается в том, что сравниваются не только соседние элементы, но элементы на некотором расстоянии: предварительно осуществляется грубый проход алгоритма, который постепенно становится все точнее и точнее. Для этого выбирается последовательность расстояний, которая заканчивается 1 (например, 5, 3, 1). Для каждого из них выполняется сортировка вставками с сравнением элементов с текущим расстоянием. Последняя итерация содержит классический алгоритм сортировки вставками. Существует различные способы подбора последовательности расстояний, рассмотрим один из самых простых: $D_1 = N/2$; $D_i = D_{i-1}/2$ … 1 – первой расстояние берется как половина длины массива; каждое последующее – как половина предыдущего. Последовательность заканчивается расстоянием 1.

Листинг А-7. 12

```
function ShellsSort($arr)
{
       // вычисляем в цикле последовательность расстояний
      for( $k=ceil(count($arr)/2); $k>=1; $k= ceil($k/2) )
      {
            for($i=$k; $i<count($arr); $i++)     // для всех элементов начиная с первого
            {
                  $val = $arr[$i];      // сохраняем значение текущего элемента
                  $j = $i-$k;     // начинаем перебор с элемента с k меньшим индексом
                      // пока не найден элемент меньше текущего
                  while( $j>=0 && $arr[$j]>$val )
                  {
                        $arr[$j+$k] = $arr[$j];    // сдвигаем элементы вправо
                        $j-=$k;
                  }
                  $arr[$j+$k] = $val;  // вставляем текущий элемент на свое место
            }
      }
      return $arr;
}
```

Первый цикл вычисляет последовательность расстояний `$k` между сравниваемыми элементами: первое расстояние берется как округленное в большую сторону половина длины массива; каждое последующее – как половина предыдущего. Цикл завершается алгоритмом сортировкой вставками, который и реализуется внутренними циклами при расстоянии равном 1.

Внутри цикла размещается уже рассмотренный ранее алгоритм сортировки вставками, отличие которого только в том, что сравниваются и сдвигаются не все элементы подряд, а элементы на расстоянии `$k`. Например, при `$k=3` будут сравниваться элементы (0, 3, 6, 9 …); (1, 4, 7, 10 …) и (2, 5, 8, 11 …). Такой подход позволяет в среднем уменьшить количество перестановок и сравнений по сравнению с обычным алгоритмом вставками.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Алгоритм быстрой сортировки массива

Один из самых эффективных алгоритмов сортировки массивов. Парадоксально, но факт в том, что это модернизация одного из самых медленных алгоритмов – пузырьковой сортировки.

Небольшие изменения позволили достичь поразительных результатов. Разработана английским ученым Чарльзом Хоаром в 1960 году.

Суть алгоритма заключается в разбиении массива на две части с помощью некоторого элемента, называемого "опорная точка". Все элементы меньше опорной точки перекидываются в левую часть массива, все элементы большие или равные опорной точке – в правую (или наоборот для другого направления сортировки). Для двух получившихся частей массивов описанные действия повторяются вновь: т.е. две части рассматриваются как отдельные массивы, которые сортируются тем же способом, с выбором своих опорных точек. Такое разбиение происходит до тех пор, пока оно возможно. В конце все массивы склеиваются в один.

Очень важно правильно выбрать опорную точку: в идеале это должна быть медиана, но для ускорения вычислений можно выбирать среднее арифметическое всех элементов массива. Также можно выбирать значение случайного элемента массива; последний элемент в массиве; значение "центрального" элемента в массиве и т.д.

Листинг А-7. 13

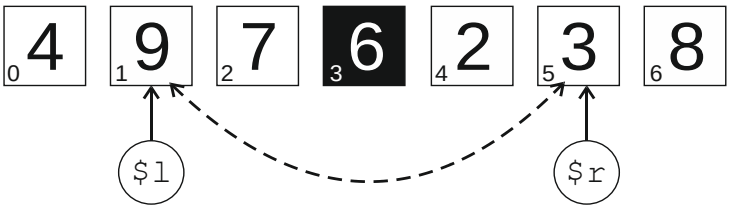
```
// входными параметрами функции являются массив,
// левая и правая граница сортируемой части
function quickSort(&$arr, $left, $right)
{
      $l=$left;       // копируем переменные для манипуляции
      $r=$right;

      $point = $arr[ floor(($left+$right)/2) ]; // вычисляем опорную точку

      do    // цикл сортировки массива
      {
            // сдвигаем левую границу вправо до тех пор,
               // пока не найден элемент массива равный опорному
            while( $arr[$l]<$point ) $l++;

               // сдвигаем правую границу влево до тех пор,
               // пока не найден элемент массива равный опорному
            while( $arr[$r]>$point ) $r++;

            if( $l <= $r ) // если левая и правая граница не пересекаются
            {
                  // меняем текущие элементы местами
                  $temp=$arr[$l]; $arr[$l]=$arr[$r]; $arr[$r]=$temp;
                      $l++;       // сдвигаем границы далее
                  $r--;
            }
      } while( $l<=$r )     // продолжаем цикл пока границы не пересекутся

      if( $r > $left )      // если присутствует левая част массива
            quickSort($arr, $left, $r);     // сортируем ее

      if( $l > $right )     // если присутствует правая часть массива
            quickSort($arr, $l, $right);    // сортируем ее
}

quickSort($arr, 0, count($arr)-1);   // вызов функции быстрой сортировки
```

Первое на что следует обратить внимание – функция не возвращает значение. Это возможно т.к. массив передается по ссылке (`&$arr`), в этом случае при его изменении внутри функции он изменяется и в остальной частим программы. Также входными параметрами являются первый и последний индексы массива для которых необходима сортировка. При первом вызове это `0` и `count($arr)-1`, т.е. индексы первого и последнего элемента массива.

В начале работы функция сохраняет границы сортируемой части массива в переменных `$l` и `$r` – теперь их можно изменять, не опасаясь потерять начальные значения. Опорная точка вычисляется как индекс посередине массива и сохраняется в переменной `$point`. Далее в двух циклах границы смещаются к центру массива до тех пор, пока не достигнут значения опорной точки. Так для

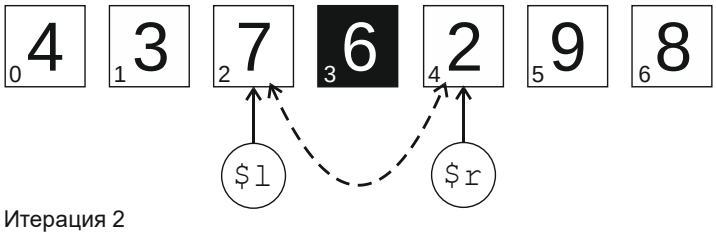
массива (4, 9, 7, 6, 2, 3, 8) опорной точкой будет служить значение 6, а левой и правой границей – 9 и 3.

Исходный массив. Итерация 0

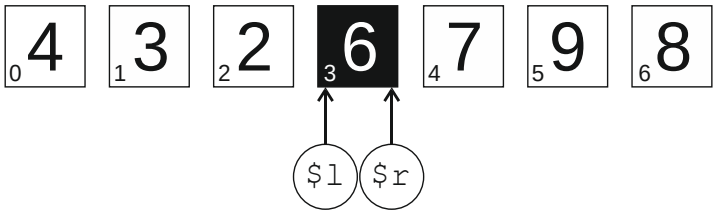


Если текущая левая граница `$l` не превышает правую границу `$r` – значит наступила ситуация, когда в левой части массива найден элемент которому место в правой, а в правой – элемент которому место в левой. А значит необходимо поменять эти элементы местами и анализировать уже следующие элементы. Далее еще раз проверяется рассмотренное условие о непересекаемости границ и, если оно выполняется, действия по сдвигу границ продолжаются с помощью цикла.

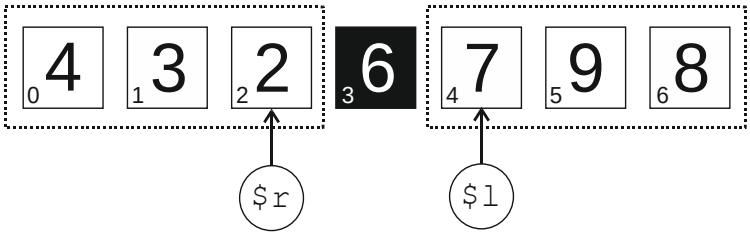
Итерация 1



Итерация 2



Итерация 3. Выбор частей массивов



В конце концов правая граница будет меньше левой: тогда все элементы он первого до правой границы будут меньше опорной точки, а элементы от левой границы до конца массива – больше. Тогда, если эти части содержат более одного элемента – они сортируются той же самой функцией, как и весь массив: в этом случает в качестве параметров передаются не границы массива, а границы этих частей.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно. Также сформируйте еще одну дополнительную функцию, которая бы сортировала массив без передачи в нее нуля и длины массива (эта функция будет вызывать уже сформированную). Для сортировки массива используйте ее.

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Какие изменения необходимо внести в программу, чтобы она корректно работала без использования информации из скрытого поля *arrLength*?
2. Как при нажатии кнопки "Добавить еще один элемент" добавлять не один, а несколько элементов сразу?
3. Как будет выглядеть обратная к *arg_is_not_Num*() функция (возвращает *true*, если аргумент число)?
4. Детально разберите и расскажите: как работают алгоритмы шейкерной сортировки массива, сортировки слиянием и вставками?
5. В каком случае на листинге 7.9 без условия $j>=0 будет бесконечный цикл?
6. Как изменится работа программы листинга 7.9, если в цикле с предусловием будет $j>0?
7. Какой метод сортировки использует стандартная функция PHP?
8. Как досрочно прекратить выполнения PHP-программы?
9. Если была вызвана функция exit() – будет ли в браузер передана статическая верстка, размещенная после последнего фрагмента кода PHP?
10. Как получить системное время с помощью PHP?
11. Какие параметру присутствуют у функции microtime() и для чего они используются?

Основы работы со строковыми данными в PHP. Кодировка. Анализ текста.

Лабораторная работа № А-8.

ЦЕЛЬ РАБОТЫ

Получение представлений об организации хранения текста в PHP, принципов работы с ним, особенностей обработки текстовой информации в различной кодировке.

ПРОДОЛЖИТЕЛЬНОСТЬ

2 академических часа (1 занятие)

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу http два документа:

- *index.html* – статический HTML-файл с формой ввода текста;
- *result.php* – анализатор введенного текста.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Первый документ содержит статический *HTML*-код в кодировке *UTF*-8 и включает в себя форму с полем для ввода многострочного текста. Кроме того, он содержит кнопку "*Анализировать*", при нажатии которой всегда открывается документ *result.php*, содержащий:

1. исходный текст;
2. информацию о тексте;
3. кнопку «Другой анализ».

Исходный текст выделяется цветом и курсивом. В случае если текста нет – выводится надпись: "Нет текста для анализа". Информация о тексте включает в себя:

1. количество символов в тексте (включая пробелы);
2. количество букв;
3. количество строчных и заглавных букв по отдельности;
4. количество знаков препинания;
5. количество цифр;
6. количество слов;
7. количество вхождений каждого символа текста (без различия верхнего и нижнего регистров);
8. список всех слов в тексте и количество их вхождений, отсортированный по алфавиту.

Результат анализа оформляется в виде таблицы с границами между ячейками. Кнопка "*Другой анализ*" реализуется с помощью тега `<a>`, при нажатии на нее вновь загружается первый документ *index.html*.

Корректно анализироваться должен как английский, так и русский текст.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Верстка документа *index.html* не представляется сложной задачей, поэтому остановимся на втором документе. Стоит отметить, что т.к. оба документа связаны, то они должны быть выполнены в одной кодировке *UTF*-8, иначе данные из первого документа будут неверно интерпретироваться во втором.

Листинг А-8. 1

```
if( isset($_POST['data']) && $_POST['data'] )   // если передан текст для анализа
{
     echo '<div class="src_text">'.$_POST['data'].'</div>';    // выводим текст
```

```
    test_it( $_POST['data'] );                  // анализируем текст
}
else       // если текста нет или он пустой
    echo '<div class="src_error">Нет текста для анализа</div>'; // выводим ошибку
```

Итак, второй документ содержит программный код PHP для анализа текста. Первое что необходимо сделать – это узнать доступен ли вообще этот текст для анализа. Если он не был передан, т.е. в документ не были переданы данные из формы, или же был передан пустой текст – анализ не производится, а выводится соответствующее сообщение.

Если же текст был передан, и он содержит символы – то он выводится в блоке `<div>`, что позволяет с помощью *CSS* оформить его согласно требованиям лабораторной работы. Затем вызывается функция `test_it()`, которая собственно и анализирует текст. Рассмотрим ее более подробно.

Листинг А-8. 2

```
function test_it( $text )
{
      // количество символов в тексте определяется функцией размера текста
      echo 'Количество символов: '.strlen($text).'<br>';
      // определяем ассоциированный массив с цифрами
      $cifra=array( '0'=>true, '1'=>true, '2'=>true, '3'=>true, '4'=>true,
                    '5'=>true, '6'=>true, '7'=>true, '8'=>true, '9'=>true );

      // вводим переменные для хранения информации о:
      $cifra_amount=0;    // количество цифр в тексте
      $word_amount=0;     // количество слов в тексте
      $word='';      // текущее слово
      $words=array();     // список всех слов

      for($i=0; $i<strlen($text); $i++)   // перебираем все символы текста
      {
            if( array_key_exists($text[$i], $cifra) )    // если встретилась цифра
                  $cifra_amount++;                  // увеличиваем счетчик цифр
            // если в тексте встретился пробел или текст закончился
            if( $text[$i]==' ' || $i==strlen($text)-1 )
            {
                  if( $word ) // если есть текущее слово
                  {
                        // если текущее слово сохранено в списке слов
                        if( isset($words[$word]) )
                              $words[ $word ]++; // увеличиваем число его повторов
                        else
                              $words[ $word ]=1; // первый повтор слова
                  }
                  $word=''; // сбрасываем текущее слово
            }
            else    // если слово продолжается
                  $word.=$text[$i]; //добавляем в текущее слово новый символ
      }
      // выводим количество цифр в тексте
      echo 'Количество цифр: '.$cifra_amount.'<br>';
      // выводим количество слов в тексте
      echo 'Количество слов: '.count($words).'<br>';
}
```

Для измерения длины текста в символах используется стандартная функция `strlen()`. Подсчет отдельной группы символов немного сложнее – в качестве примера подсчитаем количество цифр в тексте. Для этого необходимо найти способ, с помощью которого можно однозначно определить, принадлежит ли символ группе или нет. Это можно сделать несколькими путями. Первый и самый простой – с помощью условного оператора: `if( $symb=='0' || $symb=='1' || … $symb=='9' ) { … }`. Если в группе немного символов – такая запись вполне уместна. Если же их много – то можно использовать и другой подход, реализованный в примере листинге А-8. 2.

В начале производится формирование ассоциированного массива, ключами которого является группа символов – в данном случае все цифры. Тогда, проверяя наличие в этом массиве элемента с индексом равным проверяемому символу, можно однозначно сказать принадлежит ли этот символ группе или нет. Поэтому, последовательно в цикле перебирая все символы текста и проверяя каждый из них на вхождение в указанном массиве в качестве ключа, мы можем сосчитать количество цифр в тексте (переменная `$cifra_amount`).

Далее функция анализирует количество слов в тексте. Для этого используется признак окончания слова: наличие пробела, знака препинания или конец текста. Исходя из этого можно описать следующий алгоритм разбиения текста на слова.

1. Текущее слово инициализируется пустой строкой.
2. В цикле для всех символов текста:
   2.1. если символ пробел, знак препинания или он последний в тексте, то текущее слово добавляется в список слов и сбрасывается в пустую строку;
   2.2. иначе к текущему слову добавляется текущий символ.

В качестве примера в рассмотренном коде в качестве признака окончания слова используется только пробел. Перед циклом создается переменная `$word` инициализированная пустой строкой и пустой массив `$words`, в котором будет хранится информация о количестве вхождений слов в текст.

Тогда, перебирая в цикле все символы текста, в переменную `$word` добавляются символы до тех пор, пока не встретился пробел (или текст не закончился). Это значит, что в переменной найдено какое-то слово и необходимо учесть его присутствие в тексте. Чтобы продолжить искать следующее слово необходимо опять присвоить переменной `$word` пустую строку.

Например, для текста "*Привет всем*" в переменную по очереди будут добавлены символы "*П*", "*р*", "*и*", "*в*", "*е*" и "*т*", сформировав в ней строку "*Привет*". Далее `$word` опять будет пустой строкой и в нее опять-таки по очереди будут добавлены символы "*в*", "*с*", "*е*" и "*м*", сформировав следующее слово "*всем*".

Учет количества вхождений осуществляется с помощью ассоциированного массива `$words`: при нахождении слова проверяется наличие в массиве элемента с индексом в виде искомого слова. Если такой элемент найден, то его значение увеличивается на единицу; если нет – то элемент добавляется в массив и его значение инициализируется единицей.

Например, для текста "*привет всем привет*" будут последовательно определены три слова: "*привет*", "*всем*" и "*привет*". Перед анализом массив `$words` пуст. После нахождения первого слова осуществляется проверка на наличие в нем элемента с индексом `'привет'`, которого в пустом массиве естественно нет. Поэтому в массив добавляется первый элемент: `$words['привет']=1`. (значение элемента означает количество вхождений слова в тексте; при первой встрече слова оно естественно равно единице) Второе слов также отсутствует в массиве, поэтому после его нахождения в массив также добавляется элемент `$words['всем']=1`. Но третье слово уже было сформировано ранее, т.е. в массиве уже присутствует элемент с ключом `'привет'`, а, следовательно, его значение просто увеличивается на единицу, становясь равным количеству вхождений слова в текст. Если когда-либо слово "*привет*" опять встретится в тексте – значение элемента будет снова увеличено на 1, т.е. будет определено что это слово встретилось три раза.

Таким образом в массиве `$words` в качестве ключей выступают все слова текста, а в качестве значений – количество их повторений в нем.

Самостоятельно доработайте функцию для подсчета и вывода количества знаков препинания, строчных букв, заглавных букв и всех букв аналогичным цифрам способом; учета знаков препинания как признака окончания слова; вывода количества слов; упорядочивание массива слов и вывод информации о количестве слов в тексте.

Последним не рассмотренным заданием лабораторной работы является подсчет количества повторений каждого символа в тексте. Решим эту задачу аналогичным подсчету слов способом, для чего напишем отдельную функцию.

Листинг А-8. 3

```
function test_symbs( $text )
{
	$symbs=array(); // массив символов текста
	$l_text=strtolower( $text ); // переводим текст в нижний регистр

	// последовательно перебираем все символы текста
	for($i=0; $i<strlen($l_text); $i++)
	{
		if( isset($symbs[$l_text[$i]]) ) // если символ есть в массиве
			$symbs[$l_text[$i]]++;	// увеличиваем счетчик повторов
		else				// иначе
			$symbs[$l_text[$i]]=1; // добавляем символ в массив
	}
	return $symbs; // возвращаем массив с числом вхождений символов в тексте
}
```

Т.к. по условиям лабораторной работы число вхождений символов считается без учета регистра, то текст перед анализом целиком переводится в нижний регистр. Тогда в тексте все символы верхнего регистра перейдут в нижний, а число их вхождения в тексте будет автоматически добавлено к числу вхождений соответствующих им символов в нижнем регистре.

Подсчет числа вхождений символов аналогичен подсчету числа слов в тексте, за исключением того, что нет необходимости вводить дополнительную переменную для формирования текущего символа: он всегда известен. Поэтому, массив `$symbs`, аналогичный массиву `$words`, строится путем перебора всех символов текста. Если символ уже есть в списке (есть элемент массива с таким индексом) – число его повторов увеличивается на 1. Если нет – добавляем в массив новый элемент с ключом из этого символа.

Самостоятельно добавите вызов функции `test_symbs()` в функцию `test_it()` и напишите *PHP* код для вывода этой информации.

Кодировка и возможные затруднения в реализации лабораторной работы

Как известно при использовании кодировки *UTF*-8 кириллические символы занимают 2 байта, в то время как латинские – 1 байт. Из-за этого использование обычных функций для обработки строк будет работать некорректно – все они будут считать, что любой символ занимает 1 байт. Для исправления ситуации можно пойти двумя путями:

- использовать функции для *multybyte*-строк;
- перекодировать строки.

В первом случае функции автоматически учитывают сколько байт занимает символ и возвращают правильный результат. К сожалению, библиотека с ними не всегда подключена и корректно работает. И если подключение библиотеки легко проверить с помощью условного оператора (например: `if( function_exists('mb_strtolower') ) { ... }`), то корректность работы проверяется только на практике.

Второй способ более простой. Он заключается в том, что перед анализом текста он принудительно перекодируется из кодировки *UTF*-8 в *CP*1251, где каждому символу всегда соответствует 1 байт и все стандартные функции прекрасно работают.

Листинг А-8. 4

```
// перед анализом перекодируем текст из UTF-8 в CP1251
test_it( iconv("utf-8", "cp1251",$_POST['data']) );

function test_it( $text )
{
	...

	// непосредственно перед выводом перекодируем строку обратно в UTF-8
	echo iconv("cp1251", "utf-8", $words[$key]);
```

```
        ...
}
```

Но в этом случае необходимо помнить, что сайт использует кодировку *UTF*-8, поэтому непосредственно перед выводом результата анализа все строки должны перекодироваться обратно из *CP*1251 в *UTF*-8 – иначе символы будут отображаться неправильно.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Сортировка ассоциированного массива по ключам	`ksort(); // по возрастанию значений ключей`
	`krsort($arr); // по убыванию значений ключей`
	Функции возвращают *true* в случае успешной сортировки и *false* в случае ошибки.

Проверка существования функции	`function_exists('function_name');` Возвращает *true* в случае существования функции с указанным именем; *false* – в противном случае.

Перевод регистра для строки	`// преобразует строку в нижний регистр`
	`$s2=strtolower($s1);`
	`// преобразует строку в верхний регистр`
	`$s3=strtoupper($s4);`

Определение длины строки	`strtolower($str); // возвращает длину строки`
Функции для multybyte-строк	Синтаксис функций совпадает со стандартными функциями по обработке строк – перед именем добавляется префикс "mb_". Например: `mb_strtolower()`.
Смена кодировки у строки	`iconv( $in_charset, $out_charset, $str );` Функция возвращает результат перекодировки строки `$str` из кодировки `$in_charset` в `$out_charset`.

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Какие стандартные функции сортировки массивов есть в PHP и как они работают?
2. Как проверить была ли определена функция?
3. Как изменить регистр строки:
4. Что такое функции для multybyte-строк и в чем их отличие от стандартных?
5. Что такое кодировка?
6. Какие кодировки Вы знаете? В чем их отличия? Преимущества и недостатки?
7. Как перекодировать строку в PHP из одной кодировки в другую?

Основы баз данных и использования программных модулей. Записная книжка.

Лабораторная работа № В-1.

ЦЕЛЬ РАБОТЫ

Получение навыков оптимизации программы *PHP* с помощью использования программных модулей. Знакомство с основами использования Базы данных в *PHP* для реализации различных задач.

Очень часто абсолютно одинаковый программный код используется несколько раз в разных местах программы. Для *PHP*, который используется в веб-приложениях, это актуально при использовании нескольких страниц сайта со схожими функциями. Если просто "копировать" кусок кода и вставлять его на прочие страницы, да и на ту же самую страницу, то, во-первых, объем кода быстро разрастается, что приводит к ухудшению его восприятию программистом и увеличению числа ошибок в нем. Во-вторых, исправление или изменение одного фрагмента кода приводит к необходимости искать все остальные его включения, что не всегда удается – после нескольких таких модификаций код, который должен быть одинаковым и работать по одному алгоритму, довольно сильно отличается от своего аналога в других файлах проекта.

Решением этой задачи может быть применение модулей: отдельных файлов с *PHP*-кодом программы. При необходимости они "встраиваются" в любое место основной *PHP*-программы или в другой модуль. Это позволяет выделить часто повторяющие фрагменты, например, пользовательские функции, записать их в отдельный файл и использовать на разных страницах одного сайта.

Более того, многие функции или фрагменты кода без изменения могут быть использованы в разных проектах: например, для написания административной части сайта или для типовой обработки информации. По такому принципу строятся многочисленные библиотеки функций, которые позволяют *PHP*-программистам сразу использовать довольно мощный функционал без отрыва от основного задания. Так широко известна библиотека "*pclzip.lib.php*", позволяющая использовать *ZIP*-архивацию для файлов. Тогда, если необходимо исправить ошибку или реализовать новый алгоритм для того или иного действия, правка кода производится только один раз, после чего при необходимости файл с модулем просто копируется в другие проекты.

Использование модулей позволяет не только сократить код за счет использования одного и того же программного модуля на разных страницах, но и упростить сам процесс разработки. Нет необходимости работать с огромным файлом, содержащим тысячи строчек кода: разные его фрагменты очень просто реализовывать последовательно, модуль за модулем. Более того, этот процесс можно поручить сразу нескольким программистам, которые будут выполнять этот процесс параллельно.

Базы данных – не менее важная тема при Веб-разработке. Это оптимальный способ организации длительного хранения данных на сервере. Причем, в отличие от выполняющихся на локальных компьютерах пользовательских программ, он зачастую и единственный. Действительно, на локальном компьютере совсем не обязательно использовать специальные базы данных, можно просто сохранить промежуточный или окончательный результат в файл. Для веб-сайтов это не так: дело в том, что одну и ту же страницу могут одновременно просматривать, а значит и выполнять *PHP*-программу, десятки или даже сотни посетителей. Поэтому при использовании файлов неизбежно возникнет ситуация, когда к одному и тому же файлу будет обращение от разных системных процессов, что приведет к конфликту.

Делать специальную систему идентификации посетителей, привязанную к его *ip*-адресу, и выделять для каждого из них свой файл можно, но очень сложно: адреса могут меняться. База данных же значительно упрощают работу, позволяя не только согласовывать доступ и изменения

данных разными посетителями сайта, но и предоставляя современные средства доступа, организации и манипуляции данными.

Одним из самых распространенных серверов баз данных в настоящий момент времени является *MySQL*. Скорее всего именно он будет установлен на хостинге, который Вы решите использовать. Для его настройки, создания баз и таблиц используется специальная панель администратора *phpMyAdmin*. Как и большинство современных БД *MySQL* относится к реляционным базам данных, а значит для его использования применяется *SQL* – язык структурированных запросов. С его помощью осуществляется быстрый поиск требуемых данных, их фильтрация, добавление, удаление или редактирование. Знание и понимание принципов организации реляционных баз данных и *SQL* – важная и необходимая часть навыков успешного *PHP*-программиста.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа работы в аудитории, 4 академических часа – самостоятельно.

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу *HTTP* документы:

- *index.php* – единственный загружаемый в браузер документ, осуществляющий всю работу сайта;
- *menu.php* – формирующий меню и регламентирующий его работу модуль;
- *viewer.php* – модуль для вывода содержимого базы данных в браузер;
- *add.php* – модуль для добавления новой записи в базу данных;
- *edit*.php – модуль для редактирования существующей записи базы данных;
- *delete.php* – модуль для удаления записи из базы данных.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Сайт должен предоставлять удобный сервис для хранения, редактирования и доступа к данным о людях, т.е. фактически представлять из себя записную книжку с контактами. Каждый контакт характеризуется следующими атрибутами:

- фамилия;
- имя;
- отчество;
- пол;
- дата рождения;
- телефон;
- адрес;
- E-майл;
- комментарий.

Для работы сайта в качестве части URL используется только файл "*index.php*", все остальные файлы являются программными модулями и не доступны напрямую в браузере.

Модуль "*menu.php*" содержит функцию без параметров которая возвращает в виде строки *HTML*-код, содержащий основное меню сайта. Меню состоит из следующих пунктов:

- Просмотр;
- Добавление записи;
- Редактирование записи;
- Удаление записи.

Все пункты меню оформляются в виде созданных с помощью тега `<a>..</a>` кнопок синего цвета. При первой загрузке активным считается пункт меню "*Просмотр*", при переходе на другой пункт меню – тот по которому перешли. Активный пункт меню должен выделяться красным цветом.

В случае выбора пункта "*Просмотр*" ниже основных пунктов меню выводятся дополнительные пункты для определения вида сортировки (см. ниже). Визуально они должны быть меньше

основных, но оформлены том же стиле. При переходе по дополнительным пунктам выделение основного пункта меню не должно исчезать. У дополнительных пунктов меню также есть активный, выделенный красным цветом пункт меню (по умолчанию первый по порядку).

Вывод меню осуществляется в верхней части экрана с помощью подключения указанного модуля к файлу "*index.php*" и вызова сформированной функции. Вне зависимости от переданных на сайт *GET*- и *POST*-параметров один из пунктов меню всегда должен быть активным.

При первой загрузке сайта в браузере (либо при выборе пункта меню "*Просмотр*") выводится содержимое записной книжки в виде таблицы с не более чем 10 строками. Если записей в базе данных больше – под таблицей выводится пагинация (ссылки на другие страницы в виде их номеров; при наведении на ссылку курсора мыши она обводится в рамочку толщиной 2*px*), которая позволяет выводить другие фрагменты базы данных по 10 записей каждый. Для удобства использования должна быть предусмотрена возможность сортировки результата в порядке добавления записей в базу, а также по фамилии или по дате рождения (в порядке возрастания).

Для подготовки *HTML*-кода с таблицей и ссылками пагинации используется модуль "*viewer.php*", который должен содержать пользовательскую функцию с параметрами, определяющими тип сортировки и номер выводимой страницы пагинации. Функция вызывается из модуля "*index.php*" который и выводит в браузер возвращаемую ей строку с контентом.

При переходе по ссылке "*Добавление записи*" на месте таблицы выводится форма для добавления новой записи в базу данных. После ее заполнения и отправки страница перезагружается и выводится та же форма с надписью: "*Запись добавлена*" (зеленым цветом) или "*Ошибка: запись не добавлена*" (красным цветом) в зависимости от успешности выполнения операции. Форма задается в модуле "*add.php*" в статическом виде. *PHP*-код для добавления записи также располагается в этом модуле.

При переходе на пункт меню "*Редактирование записи*" загружается страница с формой, аналогичной форме для добавления записи. Перед формой выводятся ссылки с текстом, соответствующим именам и фамилиям из базы данных (список сортируется по фамилии, затем по имени). При переходе по ссылке страница перезагружается и в полях формы отображаются соответствующие выбранной записи значения. Текущая запись в списке выделяется цветом или рамкой; если запись не была выбрана пользователем – то текущей считается первая запись по порядку. Весь *HTML*- и *PHP*-код полностью находится в модуле "*edit.php*".

Доступная при переходе по ссылке "*Удалить запись*" страница содержит список ссылок, текст которых соответствует фамилии и инициалам из базы данных. При переходе по ссылке страница перезагружается, соответствующая запись удаляется из базы данных, выводится надпись: "*Запись с фамилией Иванов удалена*" (вместо "*Иванов*" указывается фамилия из удаляемой записи). Весь *HTML*- и *PHP*-код полностью находится в модуле "*delete.php*".

ПРИМЕНЕНИЕ ПОЛУЧАЕМЫХ В РАБОТЕ ЗНАНИЙ И НАВЫКОВ

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Использование модульного подхода позволяет значительно упростить программирование, реализуя каждый из них по очереди. Всего по заданию лабораторной работы необходимо реализовать один главный файл и пять программных модулей. Начнем с модуля *menu.php*, реализующего основное меню сайта.

Листинг В-1. 1

```
<div id="menu">
<?php
      // если нет параметра меню – добавляем его
      if( !is_array($_GET['p']) ) $_GET['p']='view';

      echo '<a href="/?p=viewer"'; // первый пункт меню
```

```
        if( $_GET['p'] == 'viewer' ) // если он выбран
              echo ' class="selected"'; // выделяем его
        echo '>Просмотр</a>';
        echo '<a href="/?p=add"';   // второй пункт меню
        if( $_GET['p'] == 'add' ) echo ' class="selected"';
        echo '>Добавление записи</a>';

        if( $_GET['p'] == 'viewer' )     //если был выбран первый пунт меню
        {
            echo '<div id="submenu">';   // выводим подменю

              echo '<a href="/?p=viewer&sort= byid "';// первый пункт подменю
              if( !isset($_GET['sort']) || $_GET['sort'] == 'byid' )
                    echo ' class="selected"';
              echo '>По-умолчанию</a>';

              echo '<a href="/?p=viewer&sort=fam"';    // второй пункт подменю
              if( isset($_GET['sort']) && $_GET['sort'] == 'fam' )
              echo ' class="selected"';
              echo '>По фамилии</a>';

            echo '</div>';          // конец подменю
        }
?>
</div>
```

В модуле могут одновременно располагаться как статический *HTML*-код, так и выполняемый *PHP*-код. Все статические фрагменты модуля будут перенесены в результат без изменений: это дает возможность выбора и добавления в документ в виде его фрагментов полностью статических файлов. Фактически это позволяет при программировании не думать о том, как будет использоваться получаемый файл. В данном случае представим себе, что он просто будет напрямую открываться в браузере: тогда неизменную часть *HTML*-кода (тег `<div>`) логично сделать в статическом виде, а динамически формируемую часть – в виде *PHP*-программы.

В самом ее начале осуществляется проверка наличия передаваемого при переходе по пунктам меню параметра "`p`". Если он отсутствует, т.е. если это первая загрузка страницы и необходимо по-умолчанию выделить пункт меню "*Просмотр*", мы принудительно добавляем в массив `$_GET` элемент с ключом "`p`", тем самым имитируя переход по пункту меню "*Просмотр*". Для повышения безопасности выполнения программы здесь также следует проверить допустимость значения параметра: если оно недопустимо (например, `p=dhgjfkehgb`) – то программу стоит принудительно остановить. Сделайте это самостоятельно.

Формирование пунктов меню стандартно и уже рассматривалась в предыдущих лабораторных работах. В листинге В-1. 1 приведены лишь два пункта, еще два (Редактирование и Удаление записи) необходимо сделать самостоятельно. Для определенности договоримся, что значение передаваемого параметра для них будет равно "`p=edit`" и "`p=delete`" соответственно.

Подменю реализуется аналогичным образом: если выбран пункт меню "*Просмотр*", т.е. значение элемента массива `$_GET['p']` равно "`view`", выполняется формирующий подменю *PHP*-код. Обратите внимание: для управления подменю используется другой параметр "`sort`", а параметр "`p`" передается в него без изменений. Кроме того, в данном примере существование параметра `sort` явно проверяется перед его сравнением с возможным значением: первый пункт меню становится активным, если соответствующего параметру элемента массива не существует или его значение равно `byid`. Остальные – если он существует и его значение равно передаваемому в них параметру. Доработайте приведенный пример в соответствии с заданием: добавьте отсутствующие пункты меню и подменю; создайте в *css*-файле необходимые стили.

Теперь, когда у нас есть модуль, полностью формирующий работоспособное меню, можно сформировать и код главного файла сайта (*index.php*). В нем будет осуществлен вызов описанного

и других, еще не сформированных модулей. Опустим всю статическую часть этого файла (реализуйте ее самостоятельно) и приведем в листинге только его PHP-код.

Листинг В-1. 2

```
require 'menu.php';  // главное меню
// модули с контентом страницы
if( $_GET['p'] == 'viewer' ) { include 'viewer.php'; } else
if( $_GET['p'] == 'add' ) { include 'add.php'; } else
if( $_GET['p'] == 'edit' ) { include 'edit.php'; } else
if( $_GET['p'] == 'delete' ) { include 'delete.php'; }
```

Модуль *menu.php* с главным меню подключается всегда, поэтому стоит использовать конструкцию `require`: при ее наличии в *PHP*-программу перед ее выполнением будет вставлен код из указанного в ней модуля. Далее, в зависимости от значения переданного параметра, подключается тот или иной модуль с контентом. Причем в этом случае используется конструкция `include`, которая добавляет код не перед выполнением программы, а во время выполнения. Таким образом, размер программы не увеличится за счет добавления в него кода сразу из всех теоретически используемых модулей (*viewer.php*, *add.php*, *edit.php*, *delete.php*) – будет выбран и добавлен код только из того модуля, который нам необходим (обратной стороной такой гибкости является уменьшение скорости работы).

Обратите внимание: нет необходимости проверять существование элемента массива `$_GET['p']` – это уже было сделано в модуле *menu.php*. Кроме того, можно несколько сократить код за счет того, что значение переменной и имя файла с модулем совпадает.

Листинг В-1. 3

```
require 'menu.php';  // главное меню
// модули с контентом страницы
if( file_exists($_GET['p'].'.php') ) { include $_GET['p'].'.php'; }
```

Действительно, имя подключаемого модуля совпадает со значением параметра "`p`" плюс соответствующее разрешение. Если бы использовалась конструкция `require` – динамически сформировать имя подключаемого файла было бы невозможно, т.к. оно должно быть известно до начала выполнения *PHP*-программы. Но т.к. `include` добавляет код во время выполнения, имя модуля может быть сформировано динамически. Теперь последовательно реализуем программы оставшихся трех модулей.

В модуле "*viewer.php*", согласно условиям лабораторной работы, должна размещаться пользовательская функция, которая по типу сортировки и диапазону выводимых записей формирует содержимое записной книжки. Фактически в этом примере демонстрируется возможность создания библиотеки функций.

Листинг В-1. 4

```
function getFriendsList($type, $page)
{
      // осуществляем подключение к базе данных
      $mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

       if( mysqli_connect_errno() ) // проверяем корректность подключения
             return 'Ошибка подключения к БД: '.mysqli_connect_error();

      // формируем и выполняем SQL-запрос для определения числа записей
      $sql_res=mysqli_query($mysqli, 'SELECT COUNT(*) FROM friends');

      // проверяем корректность выполнения запроса и определяем его результат
      if( mysqli_errno($mysqli) && $row=mysqli_fetch_rows($sql_res) )
       {
           if( !$TOTAL=$row[0] )      // если в таблице нет записей
                 return 'В таблице нет данных'; // возвращаем сообщение
```

```
            $PAGES = ceil($TOTAL/10); // вычисляем общее количество страниц

            if( $page>=$TOTAL )  // если указана страница больше максимальной
                     $page=$TOTAL-1;  // будем выводить последнюю страницу
            // формируем и выполняем SQL-запрос для выборки записей из БД
            $sql='SELECT * FROM friends LIMITS '.$page.', 10';
            $sql_res=mysqli_query($mysqli, $sql);

            $ret='<table>';       // строка с будущим контентом страницы
            while( $row=mysqli_fetch_assoc($sql_res) ) // пока есть записи
            {
                  // выводим каждую запись как строку таблицы
                  $ret.='<tr><td>'.$row['name'].'</td>
                                 <td>'.$row['mail'].'</td>
                                 <td>'.$row['telephone'].'</td></tr>';
               }
               $ret.='</table>';  // заканчиваем формирование таблицы с контентом

            if( $PAGES>1 )  // если страниц больше одной - добавляем пагинацию
            {
                  $ret.='<div id="pages">';       // блок пагинации
                  for($i=0; $i<$TOTAL; $i++)      // цикл для всех страниц пагинации
                        if( $i != $page )    // если не текущая страница
                              $ret.='<a href="?p=viewer&pg='.$i.'">'.($i+1).'</a>';
                        else                 // если текущая страница
                              $ret.='<span>'.($i+1).'</span>';
                  $ret.='</div>';
            }
            return $ret;          // возвращаем сформированный контент
      }
                                  // если запрос выполнен некорректно
      return 'Неизвестная ошибка';  // возвращаем сообщение
}
```

Входными параметрами функции служат две переменные: `$type` – для определения типа сортировки и `$page` – для указания страницы пагинации записи которой добавляются в контент (напомним, что одна страница согласно заданию, это 10 записей). В теле функции в первую очередь происходит подключение к серверу базы данных. В качестве адреса сервера используется "*localhost*" (локальный компьютер), в качестве имени пользователя и его пароля "*user*" и "*password*" соответственно (в зависимости от технического оснащения и настроек указанные реквизиты доступа могут меняться – уточняйте их перед выполнением работы). На сервере будет выбрана для работы база данных "*friends*", поэтому ее и содержащиеся в ней таблицы необходимо предварительно создать – сделайте это самостоятельно.

Если вы это не сделали или доступ к серверу невозможен – в следующих строках PHP-программы будет осуществлена проверка успешности подключения. Если произошла ошибка, то функция `mysqli_connect_errno()` вернет ее код (код "*0*" – успешное подключение). Поэтому, согласно правилам преобразования типов, любой отличный от нуля результат функции будет воспринят условным оператором `if` как `TRUE`, и, следовательно, он будет выполнен. В данном случае будет выведено сообщение об ошибке и его текст, возвращаемый функцией `mysqli_connect_error()`.

После успешного подключения можно непосредственно выполнять *SQL*-запросы, т.е. запрашивать необходимые для отображения данные. Это можно делать сразу, но в требованиях указана необходимость пагинации. Для ее формирования необходимо знать число записей на одной странице (в данном случае это известно: 10) и количество таких страниц. Очевидно, что количество страниц пагинации равно общему количеству выводимых записей деленное на размер одной страницы с округлением в большую сторону. Поэтому, необходимо как-то вычислить количество выводимых записей. Это можно сделать двумя путями: во-первых, выполнить *SQL*-запрос и посчитать количество записей в результирующей таблице. Во-вторых, выполнить *SQL*-запрос, который бы средствами *SQL*-сервера подсчитал количество записей.

Плюсом первого варианта является отсутствие необходимости выполнять какие-либо предварительные действия: выполняется сразу окончательный *SQL*-запрос, возвращающий

нужные нам результаты. Но, при большом объеме данных и результат будет очень большим (именно для уменьшения передаваемой и отображаемой информации и применяется пагинация) – даже для отображения небольшой страницы из 10 записей сервер будет формировать полный результат выборки, возможно содержащий тысячи записей. Такой минус на порядок перевешивает все плюсы, поэтому необходимо применять второй вариант: предварительный запрос с функцией `COUNT()`.

В следующем условном операторе проверяется корректность выполнения *SQL*-запроса с помощью функции `mysqli_errno($mysqli)` и одновременно формируется и проверяется наличии первой (и для данного запроса единственной) записи результирующей таблицы. В итоге оператор срабатывает при возможности корректной работы с таблицей `friends` (она существует и доступна пользователю), если оператор не выполняется – в конце функции возвращается сообщение об неизвестной ошибке базы данных.

Для удобства работы и понимания текста программы результат из массива `$row[0]` переносится в переменную `$TOTAL` и, если она равна нулю, функция возвращает сообщение об отсутствии в таблице данных. Если же записи есть, то вычисляется общее количество страниц в пагинации и заносится в переменную `$PAGE`. Не лишним будет проверить корректность переданной в функцию в качестве параметра номера выводимой страницы пагинации, т.е. переменной `$page`. Если она больше максимально возможной – она будет уменьшена: теперь мы можем в дальнейшем безбоязненно использовать эту переменную без проверок.

После всех предварительных вычислений можно выполнять и окончательный вариант *SQL*-запроса. В нем используется конструкция `LIMIT`, указывающая диапазон передаваемых записей результирующей таблицы. При этом диапазон вычисляется так, чтобы он полностью совпал с диапазоном соответствующей страницы пагинации, т.е. все полученные в результирующей таблице записи можно смело отображать. Проверять корректность работы запроса в данном случае не обязательно и даже нежелательно, т.к. успешное выполнение предыдущего запроса гарантирует успешное выполнение и окончательного (он не использует никакие другие таблицы, условия, поля и т.д.).

Поэтому, введя переменную `$ret` (в ней будем накапливать результат работы функции, т.е. контент страницы), в цикле переберем все записи результирующей таблицы с помощью функции `mysqli_fetch_assoc()`. Она возвращает ассоциативный массив с ключами, соответствующими именам полей результирующей таблицы. Каждый последующий вызов функции меняет значения элементов возвращаемого ей массива на значения следующей записи: таким образом она позволяет в цикле с предусловием `while` осуществить последовательный перебор всех записей результирующей таблицы (функция вернет `FALSE` при достижении конца таблицы). На каждой итерации в переменную `$ret` добавляется строка *HTML*-таблицы из значений соответствующей записи. После цикла тег `<table>` закрывается.

На этом вывод непосредственно данных из БД закончен, но, если количество страниц больше одной – необходимо вывести ссылки на них. Это условие проверяется и, при его выполнении, в переменную `$ret` добавляются тег `<div>` для пагинации, содержащий блок с надписью: "*Страницы:*" и ссылки на них. Ссылки формируются и выводятся в цикле: обратите внимание что в тексте ссылки страницы пагинации нумеруются, начиная с единицы, а в адресе – начиная с нуля. Действительно, для корректности вычисления диапазона выводимых записей первая страница должна быть на самом деле нулевой, что не совсем удобно для восприятия человеком.

Ясно, что не стоит выводить на HTML-странице ссылку на саму себя (это бессмысленно), кроме того следует выделить текущую страницу пагинации в списке. Для этого в цикле осуществляется проверка: если выводится текущая страница из нее формируется не ссылка, а тег `<span>`, что позволит средствами *CSS* как-то выделить страницу.

Добавление блока пагинации заканчивает формирование контента, и функция возвращает содержимое переменной `$ret` как результат своей работы. Для выполнения условий лабораторной работы необходимо самостоятельно добавить в формируемую HTML-таблицу оставшиеся поля (*фамилия*, *пол*, *дата рождения*, *адрес*, *комментарий*); добавить подписи полей

вверху таблицы; доработать *CSS*-файлы так, чтобы таблица и блок пагинации был представлен в требуемом виде. Кроме того, необходимо добавить в функцию обработку параметра `$type`: как в *SQL*-запросах, так и для сохранения типа сортировки при переходе на другую страницу пагинации.

Листинг В-1. 5

```
require 'menu.php';  // главное меню
if( $_GET['p'] == 'viewer' ) // если выбран пункт меню "Просмотр"
{
      include 'viewer.php'; // подключаем модуль с библиотекой функций

      // если в параметрах не указана текущая страница - выводим самую первую
      if( !isset($_GET['pg']) || $_GET['pg']<0 ) $_GET['pg']=0;

      // если в параметрах не указан тип сортировки или он недопустим
      if(!isset($_GET['sort']) || ($_GET['sort']!='byid' && $_GET['sort']!='fam' &&
         $_GET['sort']!='birth'))
            $_GET['sort']='byid'; // устанавливаем сортировку по умолчанию

      // формируем контент страницы с помощью функции и выводим его
      echo getFriendsList($_GET['sort'], $_GET['pg']);
}
else // подключаем другие модули с контентом страницы
if( file_exists($_GET['p'].'.php') ) { include $_GET['p'].'.php'; }
```

Теперь, когда был реализован модуль "*viewer.php*", необходимо вновь вернуться к файлу "*index.php*": ведь доступность функции не означает ее выполнение. Поэтому немного модифицируем код *PHP*-программы главной страницы, отделив отображение содержимого базы данных от подключения всех остальных модулей.

Итак, если был выбран пункт меню "*Просмотр*" или загружена первая страница сайта, то подключается модуль "*viewer.php*" – в нем мы только что реализовали функцию `getFriendsList()`, поэтому она становится доступна и в основной *PHP*-программе (просто мысленно замените строку `include 'viewer.php';` на содержимое этого файла). В качестве параметров ей передается значения элементов массива `$_GET`, т.е. *GET*-параметры страницы. Т.к. эти параметры могут быть какими угодно (они формируются через *URL*, который можно легко изменить вручную), то предварительно следует проверить их наличие и корректность. Если номер страницы не существует или он меньше нуля – то принудительно этот параметр устанавливается равным нулю (номер страницы не может быть меньше минимального значение, т.е. нуля; превышение максимального значения проверяется внутри функции `getFriendsList()`).

Для типа сортировки проверяется его существование и равенство его значения одному из трех возможных вариантов. Если это условие не выполняется – тип сортировки устанавливается по порядку добавления записей в базу данных. После всех проверок функция вызывается и в *HTML*-код страницы добавляется результат ее работы.

Теперь перейдем к добавлению записей в таблицу, т.е. к модулю "*add.php*". В отличии от библиотеки функций, этот модуль самостоятельно формирует *HTML*-код, поэтому редактировать *PHP*-код "*index.php*" необходимости нет.

Листинг В-1. 6

```
<form name="form_add" method="post" action="/?p=add">
       <input type="text" name="name" id="name" placeholder="Имя">
       <textarea name="comment" placeholder="Краткий комментарий"></textarea>
       <input type="submit" name="button" value="Добавить запись">
</form><?php
      // если были переданы данные для добавления в БД
      if( isset($_POST['button']) && $_POST['button']== 'Добавить запись')
      {
            $mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

              if( mysqli_connect_errno() ) // проверяем корректность подключения
                    echo 'Ошибка подключения к БД: '.mysqli_connect_error();
```

```
            // формируем и выполняем SQL-запрос для добавления записи
            $sql_res=mysqli_query($mysqli, 'INSERT INTO friends VALUES ("'.
                                    htmlspecialchars($_POST['name']).'", "'.
                                    htmlspecialchars($_POST['comment']).'")');
            // если при выполнении запроса произошла ошибка - выводим сообщение
            if( mysqli_errno($mysqli) )
                  echo '<div class="error">Запись не добавлена</div>';
            else // если все прошло нормально - выводим сообщение
                  echo '<div class="ok">Запись добавлена</div>';
      }
?>
```

По условию лабораторной работы форма формируется в виде статического *HTML*-кода. При его подключении в главной программе статический код будет без изменений передан в *HTML*-код страницы так, как будто бы эта была строка, выводимая оператором `echo`.

PHP-код модуля в данном случае должен определить: есть ли необходимость добавления новой записи в таблицу. Признаком этого является переданные в программу *POST*-параметры из формы: если форма была заполнена и отправлена, то эти параметры есть, т.е. в массиве `$_POST` присутствуют элементы. Как уже говорилось в предыдущих лабораторных работах, для проверки необходимости обработки формы не следует использовать параметры, имя и значение которых может быть использовано в других формах. Действительно, если признаком добавления новой записи будет проверка наличия параметра `name`, то он может использоваться и на других формах (например, для редактирования существующей записи) – тогда не будет уверенности какое действие необходимо произвести программе. Конечно, в данном случае такая ситуация невозможна т.к. эта страница обрабатывает только одну форму, но в общем случае лучше проверять наличие уникальных параметров или их значения: например, имя кнопки или специально введенного скрытого поля.

Если признак добавления новой записи был подтвержден, т.е. в массиве `$_POST` присутствует элемент с ключом "`button`" и значением "*Добавить запись*", то осуществляется подключение к серверу баз данных, проверяется его корректность, формируется и выполняется соответствующий *SQL*-запрос (см. описание предыдущего модуля). Если запрос выполнен с ошибкой – выводится соответствующее сообщение, если без ошибок – также выводится сообщение.

Обратите внимание, что при формировании *SQL*-запроса используемые в нем параметры обрабатываются функцией `htmlspecialchars()`: она заменяет специальные символы на *HTML*-сущности. Например, символ двойной кавычки будет преобразован в "`&quot;`". Это делается для возможности использования в вводимых пользователем элементах формы кавычек, слешей и других символов, которые могут повлиять на корректность *SQL*-запроса. Действительно, если в поле комментариев пользователь ввел часть строки в кавычках (например, название чего-либо), то при формировании запроса эти кавычки будут интерпретированы сервером БД как окончание строки, и, следовательно, сам *SQL*-запрос станет некорректным: `INSERT INTO friends VALUES ("Алексей", "Участник конкурса "WorldSkills"")` – окончание строки (а именно, `WorldSkills""`) будет воспринято *SQL*-сервером как ошибочное добавление после закрытия кавычки. Поэтому преобразование таких символов – один из способов избежать такой ситуации (другой способ – использование псевдопеременных, см. ниже).

Самостоятельно доработайте *CSS*-файл таким образом, чтобы элементы формы были одинакового размера, располагались друг под другом, надписи имели требуемый в условиях лабораторной работе цвет. Добавьте на форму остальные элементы для формирования записи таблицы согласно заданию лабораторной работы. Добавьте в таблицу базы данных числовое поле id и сделайте его первичным ключом. Доработайте *SQL*-запрос так, чтобы он корректно добавлял такую запись в базу данных (например, используйте автоинкремент).

Очень часто в информацию из базы данных необходимо вносить изменения. Кончено, это можно сделать, удаляя старую запись и добавляя новую, уже исправленную. Но такой подход в корне неверен: любой сбой, любая ошибка оператора может привести к потере данных. Поэтому на сайте должны быть функции изменения содержимого таблиц: обычно они совмещаются на одной

HTML-странице с функциями добавлением записей, но для обучения и упрощения программы в условиях лабораторной работы функционал по манипуляции с данными разнесен на разные страницы сайта.

Листинг В-1. 7

```
$mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

if( mysqli_connect_errno() ) // если при подключении к серверу произошла ошибка
{
       // выводим сообщение и принудительно останавливаем PHP-программу
       echo 'Ошибка подключения к БД: '.mysqli_connect_error();
       exit();
}
// если были переданы данные для изменения записи в таблице
if( isset($_POST['button']) && $_POST['button']== 'Изменить запись')
{
       // формируем и выполняем SQL-запрос на изменение записи с указанным id
       $sql_res=mysqli_query($mysqli, 'UPDATE friends SET name="'.
                                       htmlspecialchars($_POST['name']).'"
                                       WHERE id='.$_POST['id']);
      echo 'Данные изменены';    // и выводим сообщение об изменении данных
      $_GET['id']=$_POST['id']; // эмулируем переход по ссылке на изменяемую запись
}

// формируем и выполняем запрос для получения требуемых полей всех записей таблицы
$sql_res=mysqli_query($mysqli, 'SELECT * FROM friends');

if( !mysqli_errno($mysqli) )    // если запрос успешно выполнен
{
      $currentROW=array(); // создаем массив для хранения текущей записи

      echo '<div id="edit_links">';
      while( $row=mysqli_fetch_assoc($sql_res) ) // перебираем все записи выборки
      {
            // если текущая запись пока не найдена и ее id не передан
               // или передан и совпадает с проверяемой записью
            if( !$currentROW &&
                  (!isset($_GET['id']) || $_GET['id']==$row['id']) )
               {
                     // значит в цикле сейчас текущая запись
                     $currentROW=$row;// сохраняем информацию о ней в массиве
                     echo '<div>'.$row['name'].'</div>';    // и выводим ее в списке
               }
               else    // если проверяемая в цикле запись не текущая
                  // формируем ссылку на нее
                  echo '<a href="?p=edit&id='.$row['id'].'">'.$row['name'].'</a>';
      }
      echo '</div>';

      // формируем HTML-код формы
       echo '<form name="form_edit" method="post" action="/?p=edit">
             <input type="text" name="name" id="name"';
       // если текущая запись определена - устанавливаем значение полей формы
       if( $currentROW ) echo ' value="'.$currentROW['name'].'"';
       echo '><input type="submit" name="button" value="Изменить запись">
             <input type="hidden" name="id" value="';
       if( $currentROW )             // если текущая запись определена
             echo $currentROW['id'];      // передаем ее id как POST параметр формы
       echo '"></form>';
}
else                              // если запрос не может быть выполнен
      echo 'Ошибка базы данных';      // выводим сообщение об ошибке
```

Как и в других модулях в начале модуля "*edit.php*" производится подключение к серверу баз данных. Однако, в отличии от предыдущих рассмотренных вариантов проверки корректности подключения, здесь в случае неудачи предусмотрено принудительное завершение программы с

помощью функции `exit()`. Это упрощает программу, уменьшает количество фигурных скобок, позволяет реализовывать ошибку *404*.

Далее следует проверка необходимости изменить запись таблицы: обратите внимание, все манипуляции с данными необходимо делать до их запроса из базы данных для вывода на экран. Если сделать по-другому, то при изменении, например, фамилии в списке ссылок будет выводится ее старый вариант, что существенно запутает пользователя сайта. Итак, если запись необходимо изменить, признаком чего является наличие переданного методом *POST*-параметра "`button`" и его равенство надписи на кнопке отправки формы, формируется *SQL*-запрос с оператором `UPDATE`, который и изменяет необходимую запись. При этом `id` изменяемой записи также должно передаваться через *POST*-параметр.

После вывода сообщения об успешном изменении записи происходит имитация перехода по ссылке из списка на изменяемую запись. Учитывая, что признаком перехода по ссылке является передача в *PHP*-программу *GET*-параметра `id`, в котором хранится *id* выбранной записи, то для этого достаточно присвоить элементу массива `$_GET['id']` *id* измененной записи, т.е. `$_POST['id']`. Таким образом, дальнейшее формирование *HTML*-кода будет таким же, как если бы мы перешли по ссылке на эту запись, т.е. она будет выбрана, а элементы формы будут заполнены значениями соответствующих полей. Поэтому в дальнейшем нет необходимости проверять: была ли изменена запись и ее необходимо выделить, также, как если бы она была выбрана в списке соответствующих ссылок.

Далее необходимо вывести все хранящиеся в базе данных записи в виде списка ссылок, при переходе по каждой из которых соответствующая ей запись становится текущей. Т.е. необходимый элемент списка ссылок выделяется, элементы формы заполняются значениями полей этой записи, при изменении значений элементов форм и ее отправки изменяются и соответствующие значения полей записи в базе данных.

Для этого уже известными методами формируем *SQL*-запрос, и если он успешно выполнен – выводим список ссылок, если нет – выводим сообщение об ошибке. Перед выводом списка создается пустой массив, в котором будет хранится информация о текущей записи. Т.к. при выводе все равно перебираются в цикле все записи из базы данных, то нет необходимости делать отдельный запрос для определения текущей: если *id* выводимой записи совпадает с *id* текущей, значит выводимая на данной итерации цикла запись и есть текущая и информацию о ней можно сохранить для дальнейшего использования в массиве `$currentROW`.

Именно это и происходит в цикле: если текущая запись еще не найдена и информация о ней не передана в программу (первая загрузка страницы, переход по ссылкам еще не осуществлялся), то первая же выводимая запись станет текущей, ее поля будут сохранены в массиве, а сама она будет выведена не в виде ссылки, а в виде блока `<div>`, т.е. будет выделена в списке. Это же будет сделано если на данной итерации цикла получена запись с переданным в качестве текущего *id*.

Если же текущая запись определена, и она не совпадает с рассматриваемой на данной итерации цикла записью, то будет выведена ссылка с *GET*-параметром `id` равным ее *id*. Как видно из предыдущего описания, при переходе по этой ссылке именно эта запись станет текущей, что и требовалось сделать.

Последнее задания в данном модуле – вывод формы. Она практически полностью совпадает с формой из модуля "*add.php*", но формирует значения полей в соответствии с текущей записью. Т.е. если текущая запись определена, то в *HTML*-код элементов форм добавляется свойства `value` в котором выводится соответствующее элементу значение поля записи (для некоторых элементов формы используется не свойство value, а другие способы определения его значения). Кроме того, в программу должно быть передано в качестве *POST*-параметра значение текущей записи, иначе не будет понятно какую именно запись необходимо изменить. Для этого используется скрытое текстовое поле со свойство `value` равным *id* текущей записи. В принципе, возможна передача этого параметра через свойство `action` формы путем добавления параметра *id* в *URL* страницы-обработчика, но метод со скрытым полем более корректный и безопасный.

Имеет смысл еще раз остановиться на получении информации о текущей записи из базы данных. В приведенном листинге В-1. 7 она копируется в переборе всех записей при их выводе из базы данных. Но, если необходимо выводить не все записи, а только часть из них (например, при реализации постраничной пагинации) то есть вероятность что текущая запись не попадет в перебираемые – в этом случае информация о ней не будет получена.

Кроме того, если комментарий достаточно объемен или просто в результирующей таблице много полей – т.е. если размер в памяти для каждой записи таблицы достаточно большой (а это особенно актуально если в базе данных хранятся, например, фотографии или другие цифровые данные), то такой подход будет очень затратным для ресурсов сервера. Поэтому, в таких случаях необходимо выполнять два разных *SQL*-запроса: для определения полной информации о текущей записи и для получения краткой информации о всех выводимых данных. Скорректируем программу в соответствии с приведенными аргументами, кроме того для примера будем передавать id параметр редактируемой записи через *GET*-параметр.

Листинг В-1. 8

```
$mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

if( mysqli_connect_errno() ) // если при подключении к серверу произошла ошибка
{
      // выводим сообщение и принудительно останавливаем PHP-программу
      echo 'Ошибка подключения к БД: '.mysqli_connect_error();
      exit();
}

// если были переданы данные для изменения записи в таблице
if( isset($_POST['button']) && $_POST['button']== 'Изменить запись')
{
      // формируем и выполняем SQL-запрос на изменение записи с указанным id
      $sql_res=mysqli_query($mysqli, 'UPDATE friends SET name="'.
                                      htmlspecialchars($_POST['name']).'"
                                      WHERE id='.$_GET['id']);
     echo 'Данные изменены';    // и выводим сообщение об изменении данных
}

$currentROW=array();            // информации о текущей записи пока нет
                                 // если id текущей записи передано –
if( isset($_GET['id']) )        // (переход по ссылке или отправка формы)
{
      // выполняем поиск записи по ее id
      $sql_res=mysqli_query($mysqli,
                  'SELECT * FROM friends WHERE id='.$_GET['id'].' LIMIT 0, 1');
     $currentROW=mysqli_fetch_assoc($sql_res); // информация сохраняется
}
if( !$currentROW )   // если информации о текущей записи нет или она некорректна
{
      // берем первую запись из таблицы и делаем ее текущей
      $sql_res=mysqli_query($mysqli, 'SELECT * FROM friends LIMIT 0, 1');
      $currentROW=mysqli_fetch_assoc($sql_res);
}

// формируем и выполняем запрос для получения требуемых полей всех записей таблицы
$sql_res=mysqli_query($mysqli, 'SELECT id, name FROM friends');

if( !mysqli_errno($mysqli) )    // если запрос успешно выполнен
{
     echo '<div id="edit_links">';
     while( $row=mysqli_fetch_assoc($sql_res) ) // перебираем все записи выборки
     {
          // если текущая запись пока не найдена и ее id не передан
             // или передан и совпадает с проверяемой записью
          if( $currentROW['id']==$row['id']) )
                  // значит в цикле сейчас текущая запись
                  echo '<div>'.$row['name'].'</div>';    // и выводим ее в списке
            else    // если проверяемая в цикле запись не текущая
               // формируем ссылку на нее
```

```
                echo '<a href="?p=edit&id='.$row['id'].'">'.$row['name'].'</a>';
        }
        echo '</div>';

        if( $currentROW ) // если есть текущая запись, т.е. если в таблице есть записи
        {
              // формируем HTML-код формы
                 echo '<form name="form_edit" method="post"
                       action="/?p=edit&id='.$currentROW['id'].'">
                       <input type="text" name="name" id="name" value="'.
                       $currentROW['name'].'"><input type="submit" name="button"
                       value="Изменить запись"></form>';
        }
        else echo 'Записей пока нет';
}
else                                // если запрос не может быть выполнен
        echo 'Ошибка базы данных';      // выводим сообщение об ошибке
```

При выполнении оператора `UPDATE` поиск изменяемой записи теперь осуществляется по полученному *GET*-параметру `id`. По этой причине нет необходимости имитировал переход по ссылке: этот параметр уже будет в массиве `$_GET`. Далее, как и в прошлом варианте, создаем пустой массив, в который попытаемся записать информацию о текущей записи.

Для этого проверим наличие в массиве элемента `$_GET['id']`, т.е. был ли передан в программу *id* текущей записи. Если он был передан, то попытаемся найти эту запись и получить информацию о ней. Для этого формируем *SQL*-запрос с условием выборки равенства поля *id* переданному параметру. Причем, т.к. нам точно известно, что такая запись всего одна, можно оптимизировать работу сервера явно указав ему это с помощью конструкции "`LIMIT 0, 1`": в этом случае в результирующей таблице будет только одна запись из всей выборки (после ее получения *SQL*-сервер не будет искать другие, а сразу вернет результат – это экономит ресурсы). Полученная с помощью запроса запись и является текущей (сохраняем ее в массив `$currentROW`).

Интересно, что если параметр не был передан или же в нем был передан неправильный *id*, то массив `$currentROW` останется пустым. Поэтому если это произошло, то необходимо повторно сформировать и выполнить *SQL*-запрос для получения первой записи из таблицы. Ее и будем считать текущей согласно заданию лабораторной работы. Если же и после этого массив пуст – это значит, что в таблице нет записей.

Далее происходит выполнение *SQL*-запроса для вывода списка записей: причем в отличии от предыдущего примера отбираются не все поля, а только необходимые для формирования ссылок. В цикле также убраны все операторы для сохранения информации о текущей записи: она уже получена. Достаточно только сравнения *id* выводимой записи и текущей: если они совпали, то вывод осуществляется не в виде ссылки, а в виде блока `<div>`. Обратите внимание: в условии нет необходимости проверять существование элемента массива `$currentROW['id']`. Если этот цикл выполняется, значит в таблице есть записи; если в таблице есть записи – значит текущая запись была гарантирована выбрана.

Для завершения коррекции кода остается только поменять вывод формы. Имеет смысл перед этим проводить проверку наличия текущей записи: если текущей записи нет, то изменять нам нечего и форму выводить не надо – вместо нее выводится предупреждение "*Записей пока нет*". Тогда код формы можно сформировать одним оператором `echo`. Из него было убрано скрытое поле для передачи *id* текущей записи в виде *POST*-параметра, но добавлен в *URL* обработчика формы *GET*-параметр *id* с этим же значением: т.е. передаваемые параметры приведены в соответствие с *PHP*-кодом обработки формы.

Реализуйте оба предложенные варианта модуля, убедитесь в идентичности их работы для пользователя. Кроме того, для приведения функционала модуля "*edit.php*" в соответствие с требованиями лабораторной работы самостоятельно доработайте *CSS*-файл, добавьте в приведенный код вывод текста ссылок в виде фамилии и имени (при необходимости модифицируйте *SQL*-запросы), добавьте сортировку в *SQL*-запросы.

Для удаления записи на основе разобранного материала самостоятельно разработайте и реализуйте модуль "*delete.php*", функционирующий в соответствии с заданием лабораторной работы. Обратите внимание – в списке записей необходимо выводить фамилию и инициалы, а не ФИО целиком.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Исключительные ситуации

Как правило в объектно-ориентированном подходе к программированию, который будет рассмотрен в последующих заданиях, и довольно часто в процедурном программировании вместо многочисленных проверок данных на корректность и хитроумных сообщений об ошибках, которые передаются из функции в функцию, удобно использовать так называемые исключительные ситуации (исключения).

Исключение – это отправляемое интерпретатором *PHP* сообщение о возникновении нештатной или исключительной ситуации во время выполнения кода программы, которая, как правило, является сигналом о наличии ошибки. К сожалению, не все ошибки в *PHP* приводят к исключительным ситуациям: деление на ноль, ошибки при использовании стандартных функций и т.д. необходимо обрабатывать традиционными способами, интерпретируя коды ошибок и проверяя валидность входных данных.

Но объектно-ориентированные расширения *PHP*, (например, *PDO*), использующие объекты, поддерживают исключения. Для их обработки используются три блока: `try`, `catch` и `finally` – последний необязателен. Интерпретируются они очень просто: если в любой строке блока `try` произошла исключительная ситуация, то дальнейший код блока не выполняется, а начинает выполняться код блока `catch` (исключительная ситуация поймана). Если присутствует блок `finally` – его код выполнится после завершения выполнения кода `try` или `catch`, было исключение поймано или нет.

Листинг В-1. 9

```
function someDBoperation()
{
       try                          // начало блока
       {
/* 01 */   $DBH= new PDO('mysql:host=localhost;dbname='.$dbname, $user, $pass);
/* 02 */   $DBH->exec('DELETE FROM table');     // что-то делаем с БД
       }
       catch(Exception $e)            // начало обработчика исключений
       {
            // вывод сообщения об исключительной ситуации
/* 03 */   echo 'Ошибка БД: '.$e->getMessage();
/* 04 */   return;    // возврат из функции
       }
       finally
       {
            // код, который выполнится даже если случилось исключение
/* 05 */   $DBH = null;
       }
}
```

В приведенном выше примере в блоке `try` осуществляется подключение к базе данных и выполнение *SQL*-запроса к ней. Если все происходит штатно – выполняется блок `finally` в котором освобождается память. Т.е. в этом случае будут выполнены операторы в строках 01, 02 и 05.

Если же в блоке `try` произошло исключение, например при подключении к БД в строке 01, – выполняется блок `catch` в котором выводится сообщение и функция `someDBoperation()` прекращает свою работу: программа выполнит операторы в строках 01, 03, 04 и 05. При этом код блока `finally` все равно гарантировано сработает и память будет освобождена: если код блока

try начал выполняться, то вне зависимости от наличия исключительных ситуаций и любых инструкций (кроме принудительной остановки программы), блок finally будет гарантировано выполнен. Если же очистку памяти вынести за пределы блока, то в случае исключения этот код не выполнится.

Исключительные ситуации – мощный инструмент организации программной архитектуры. Их использование упрощает код, делает его более наглядным и облегчает его понимание другими разработчиками. Для этого в *PHP* реализована возможность самостоятельной генерации исключительных ситуаций и их обработки.

Листинг В-1. 10

```
function div($a, $b)       // пользовательская функция
{
     if( !$b )             // если делитель равен нулю
          throw new Exception('Деление на ноль! '); // выбрасываем исключение
     return $a/$b;              // возвращаем результат деления
}

try
{
     $r = div($a, $b);               // вызываем функцию с попыткой деления на ноль
     echo 'Результат: '.$r.'. ';          // выводим результат
     if( $r > 255 )             // если результат больше 255
          throw new Exception('Выход за диапазон. ');     // выбрасываем исключение
     echo 'Calculation ok! ';        // сообщение об успехе
}
catch(Exception $e)
{
     echo $e->getMessage();     // сообщение об ошибке
}
echo 'Done!';
```

Программа начинает свое выполнение с захода в блок try и вызова из него пользовательской функции div(). Если второй аргумент $b не равен нулю, то она вернет результат деления $a на $b и выведет его в *html*-код. Если результат меньше или равен 255 – то следующим действием будет вывод слов Calculation ok! и Done! Если же он выйдет за пределы 255 – то в блоке try с помощью команды throw выбрасывается исключение, которое отлавливается в блоке catch, где произойдет вывод сообщения об ошибке. Даже если генерация исключительной ситуации происходит в функции, вызываемой в блоке try – она все равно будет обработана и отловлена в блоке catch: в данном примере таким образом проверяется деление на ноль. Таким образом, в зависимости от значений переменных $a и $b результат работы функции можно записать в следующей таблице.

Значения аргументов	$a=10; $b=10;	$a=10; $b=0;	$a=1000; $b=1;
Исключительные ситуации	Нет, нормальное выполнение.	Исключение в функции div()	Исключение непосредственно в блоке try
Выводимый результат	Результат: 1. Calculation ok! Done!	Деление на ноль! Done!	Результат: 1000. Выход за диапазон. Done!

Помимо текстового сообщения о произошедшей исключительной ситуации, *PHP* предоставляет возможность и более детального анализа исключения. Для этого у события Exception есть соответствующие методы, представленные в следующей таблице.

$e->getMessage();	Строка с сообщеним о исключительной ситуации.
$e->getCode();	Строка с кодом исключительной ситуации.
$e->getFile();	Строка с именем *PHP*-файла в котором произошло исключение.
$e->getLine();	Номер строки в PHP-коде в которой произошло исключение.
$e->getTrace();	Массив с содержанием стека вызванных функций.

`$e->getTraceAsString();`	Стек вызванных функций в виде строки.

Внутри блока `try` или `catch` могут размещаться вложенные обработчики исключений. В этом случае исключительная ситуация будет обработана ближайшем по вложенности блоком `catch`.

Листинг В-1. 11

```
try                                    // 01
{
      echo 1;                          // 02
      try                              // 03
      {
            echo 2;                    // 04
            throw new Exception('A');        // 05
            echo 3;                    // 06
      }
      catch(Exception $e)                    // 07
      {
            echo $e->getMessage();           // 08
            echo 4;                    // 09
            throw new Exception('B');        // 10
            echo 5;                    // 11
      }
}
catch (Exception $e)                   // 12
{
      echo $e->getMessage();                 // 13
      echo 6;                          // 14
}
```

Данный код начинается с `try` блока в котором в строке 02 выводится 1. Далее идет вложенный `try` блок, в котором после вывода символа «`2`» выбрасывается исключение «`A`». исключение обрабатывается в блоке `catch`, в котором выполняются строки 08, 09 и 10 – в последней выбрасывается исключение «`B`», которое отлавливается в блоке `catch`, выполняющего строки 13 и 14. Таким образом, приведенный пример сформирует *html*-код, содержащий строку «`12A4B6`».

Использование программных модулей

Добавление кода из другого файла во время выполнения программы	`include 'filename';` `// всегда`
	`include_once 'filename';` `// только один раз`
	При добавлении в программу конструкции `include` *PHP* приостанавливает выполнение основного файла и начинает выполнение программы из файла, указанного в конструкции. После завершения программы из модуля продолжается выполнение основного файла. Подключаемые модули также могут подключать модули, являясь для них основным файлом. Для предотвращения ситуаций подключения одного и того же модуля несколько раз используется конструкция `include_once` – в этом случае *PHP* сам будет проверять использовался ли модуль ранее. И если он уже был подключен – то его код выполняться не будет.

Добавление кода из другого файла перед выполнением программы	`require 'filename';` `// всегда`
	`require_once 'filename';` `// только один раз`
	Данная конструкция, в отличии от `include`, не приостанавливает выполнение программы, а заменяет себя на содержимое указанного в ней файла непосредственно перед началом ее выполнения. Это ускоряет процесс подключения, но также и увеличивает размер программы. Конструкция require_once предварительно проверяет был ли модуль подключен ранее. Если был – конструкция игнорируется.

Язык структурированных запросов SQL

Добавление записей в таблицу	`INSERT INTO table VALUES (val_1, … )`
	`INSERT INTO table (col_name_1, …) VALUES (val_1, … )`
	`INSERT INTO table SELECT val_1, … FROM another_table`
	Для добавления записи в таблицу `table` используется оператор `INSERT INTO`. В

	качестве добавляемых данных в скобках перечисляются все значения столбцов по порядку (если имена столбцов не указаны явно) или в том порядке, который указан перед словом `VALUES`. Во втором случае возможен пропуск каких-либо столбцов таблицы, в качестве их значений будут подставлены значения "по умолчанию". Для добавления в таблицу сразу нескольких записей, которые являются результатом выполнения *SQL* запроса, существует оператор `INSERT … SELECT`. Результатом работы оператора `SELECT` всегда является таблица с записями, которые и добавляются в таблицу `table`. Важно чтобы типы и количество столбцов возвращаемой `SELECT` таблицы совпадали с формируемой таблицей.

Примеры использования оператора INSERT	`INSERT INTO users VALUES (1, "Иванов", "qwerty")`
	Добавляем в таблицу `users` одну запись. Порядок следования значений полей строго соответствует порядку следования полей в таблице.
	`INSERT INTO users (id, name, password) VALUES (1, "Иванов", "qwerty")`
	Добавляем в таблицу `users` одну запись. Явно указываем имена полей таблицы `user`. Порядок следования значений соответствует порядку, указанному в операторе.
	`INSERT INTO users VALUES (1, "Иванов", "qwerty"), (2, "Петров", "1234"), (1, "Сидоров", "")`
	Добавляем в таблицу `users` три записи. Порядок следования значений полей строго соответствует порядку следования полей в таблице. Таким способом возможно добавление любого количества записей в таблицу.
	`INSERT INTO users (id, name, password) VALUES (1, "Иванов", "qwerty"), (2, "Петров", "1234"), (1, "Сидоров", "")`
	Добавляем в таблицу `users` три записи. Явно указываем имена полей таблицы `user`. Порядок следования значений соответствует порядку, указанному в операторе. Таким способом возможно добавление любого количества записей в таблицу.
	`INSERT INTO users SELECT id, fio, code FROM admins WHERE age>21`
	Получаем из таблицы admins все записи для которых столбец age имеет значение более `21`. Из этих записей отбираем столбцы `id`, `fio` и `code` значения которых вставляем в таблицу `users`. Имена столбцов в операторе `SELECT` могут не совпадать с необходимыми – главное соблюсти соответствие их типов. Таким образом будет добавлено столько строк, сколько удовлетворяющих условию записей нашлось в таблице `admins`. Оператор `SELECT` может использоваться в любой своей форме (см. ниже).
	`INSERT INTO users (name, password) VALUES ("Иванов", "qwerty")`
	В таблицу users добавляется одна запись. Если поле `id` было объявлено как ключевое с автоинкрементом, оно будет автоматически увеличено на `1` от прошлого значения. Если в дальнейшем необходимо знать вычисленное значение поля, например для использования в другой таблице, можно воспользоваться *SQL*-функцией `LAST_INSERT_ID()`.
	`INSERT INTO news (id, user_id, article) VALUES (1, LAST_INSERT_ID(), "Текст новости …")`
	Добавляет в таблицу `news` одну запись со значением поля `user_id` равному последнему значению автоинкремента. Если до этого выполнялся предыдущий пример оператора, то этим значением будет `id` добавленной записи таблицы `users`.

Удаление записей из таблицы	`DELETE FROM table_name`
	Удаляет все записи из таблицы `table_name`.
	`DELETE FROM table_name WHERE …`
	Удаляет из таблицы `table_name` только те записи, которые удовлетворяют указанному после `WHERE` условию. Например, `DELETE FROM users WHERE id=2` удалит из таблицы `users` записи у которых поле `id` равно `2`. Если `id` – ключевое поле, это означает удаление данной конкретной записи.
	`TRUNCATE TABLE table_name`
	Очищает таблицу удаляя все ее записи.

Изменение записей в	`UPDATE table_name SET field_name_1 = value_1, … WHERE …` Изменяет значение полей таблицы `table_name`. В операторе явно указывается каким

таблице	полям какое значение присваивается. Если поля не указаны, то они не будут изменены. Условие отбора записей в которых будут изменяться поля указывается после `WHERE`. Рассмотрим несколько примеров использования оператора.
	`UPDATE users SET name="Петров", age=19 WHERE id=1`
	В таблице `users` у записи с `id` равным `1` устанавливается значение поля `name` равным строке `"Петров"`, значение поля `age` равным `19`.
	`UPDATE users SET age=22 WHERE id>5`
	В таблице `users` для записей с `id` большим `5` устанавливается значение поля `age` равным `22`. Другие поля не изменяются.
	`UPDATE users SET age=age+1`
	В таблице `users` для всех записей значение поля `age` увеличивается на `1`.

Выборка записей в таблицах	`SELECT field_names FROM table_names WHERE conditions GROUP BY group_fields ORDER BY order_fields LIMIT rows, offset` Очень важно понимать что результатом выполнения оператора `SELECT` всегда является таблица. Наиболее часто встречающаяся форма оператора представлена выше. После слова `SELECT` указываются поля, которые должны входить в результирующую таблицу. После слова `FROM` указываются участвующие в формировании результата таблицы (одна или несколько). Условие отбора записей из исходных таблиц указывается после слова `WHERE`. Поля, по которым будет сортироваться результат и порядок сортировки (`ASC` – по возрастанию, `DESC` – по убыванию) указывается после слова `ORDER BY`. Если порядок сортировки не указан – производится сортировка по возрастанию значения. Поля для группировки записей указываются после `GROUP BY` (см. ниже разделе "Функции *SQL*") – не путайте его с полем для сортировки записей! После слова `LIMITS` указывается максимальное количество записей в результирующей таблице и отступ первой из них от начала результирующей таблицы (см. примеры). Все части оператора кроме перечня полей результирующей таблицы и исходных таблиц являются необязательными. Подробнее разбор оператора `SELECT` производится ниже с помощью примеров.

Примеры использования оператора SELECT	`SELECT * FROM users`
	Возвращает все поля и все записи таблицы `users`.
	`SELECT id, name FROM users`
	Возвращает таблицу из всех записей таблицы `users`, но только с полями `id` и `name`.
	`SELECT id, name AS fio FROM users`
	Отбирает в таблице `users` значение полей `id` и name для всех ее записей. При этом в результирующей таблице поле `name` будет переименовано в `fio`.
	`SELECT name FROM users`
	Возвращает таблицу с единственным полем `name`. Количество записей соответствует количеству записей в таблице `users`. Если в ней есть записи с одинаковым полем `name` – в результате будут несколько повторяющихся записей.
	`SELECT DISTINCT name FROM users`
	В отличие от предыдущего варианта, все повторения в результирующей таблице будут убраны.
	`SELECT id FROM users WHERE age>21 AND name<>"Иванов"`
	Формирует таблицу из одного столбца `id` со значениями, взятыми из таблицы `users` при условии, что у записи значение поля `age` больше `21` и значение поля `name` не равна строке `"Иванов"`.
	`SELECT * FROM users WHERE age<22 OR age>25 ORDER BY age`
	В таблице `users` отбираются записи у которых значение поля `age` меньше `22` или больше `25` и из них формируется результирующая таблица. Причем результат будет упорядочен по значениям поля `age` по возрастанию.
	`SELECT * FROM users WHERE id<>2 ORDER BY age, name DESC, id ASC`
	Из таблицы `users` отбираются все записи, у которых поле `id` не равно `2`. В результирующей таблице записи сначала сортируются по полю `age` по возрастанию значений, если значения поля `age` нескольких записей совпадают, то они будут отсортированы по полю `name` по убыванию, если и тут есть совпадение – по

	возрастанию `id`. Ключевое слово `ASC` может не использоваться, т.к. тип сортировки по возрастанию используется по умолчанию.
	`SELECT id FROM users ORDER BY age, name DESC`
	Результат – таблица с одним столбцом `id` со значениями из записей таблицы `users` отсортированных сначала по возрастанию поля `age`, а затем по убыванию значения поля `name`.
	`SELECT name AS fio, age+1 AS age_1 FROM users WHERE age/2<20 ORDER BY age_1 DESC`
	В результирующей таблице два поля: `fio` и `age_1`. В первое попадают значения поля `name` из таблицы `users`, во второе – значения `age` увеличенные на `1`. Для заполнения полей отбираются только те записи, у которых половина значения поля `age` меньше `20`.
	`SELECT name FROM users LIMIT 10, 20`
	Результирующая таблица формируется следующим образом: из таблицы `users` отбираются все значения поля `name`. При этом первые `20` записей пропускаются, а следующие за ними `10` (если они есть) формируют результирующую таблицу.
	`SELECT * FROM users WHERE age<23 ORDER BY age LIMIT 0, 15`
	Из таблицы `users` отбираются записи у которых значение поля `age` меньше `23`. Записи сортируются по возрастанию поля `age`. Первые `15` записей из получившегося списка попадают в результирующую таблицу.
	`SELECT users.name AS fio, news.article AS text FROM users, news WHERE user_id=users.id AND age<20 ORDER BY users.id, news.id`
	В результирующей таблице два поля: `fio` и `text`. Первое поле заполняется значениями из поля `name` таблицы `users`, второе – значениями из поля `article` таблицы `news`. Причем из таблицы `users` отбираются только те записи, у которых значение поля `age` меньше `20`. Значения из двух полей разных таблиц комбинируются следующим образом: берется запись таблицы `users`, если в таблице `news` есть записи у которых значение поля `user_id` равно значению `id` этой записи – все эти комбинации добавляются в результат.

Условия отбора в SQL	С помощью конструкций `WHERE` возможен отбор записей из таблицы с помощью указанного за ней условия. В условии могут фигурировать имена полей, вместо которых при проверке будут подставляться значения записей; скобки; математические операторы "`+`", "`-`", "`/`" и "`*`"; логические операторы "`=`", "`AND`", "`OR`" и "`NOT`"; функции. Приведем пример. `… WHERE age>22 AND (name="Иван" OR name="Петр")` Условие соответствует записям, у которых значение поля `age` больше `22`. Кроме того, значение поля `name` этой записи должно быть равно строке `"Иван"` или `"Петр"`. Другими словами, такое условие позволит отобрать из таблицы все людей по имени Иван или Петр которые старше `22` лет.

Функции в SQL	Для обработки значений в *SQL* существуют специальные арифметические, строковые и другие функции. Например, функция `CONCAT()` применяется для соединения нескольких строк.
	`SELECT CONCAT(name, "ович") FROM users`
	Возвращает таблицу со значениями, полученными как конкатенация значений поля `name` таблицы `users` и строки "`ович`".
	`SELECT * FROM users WHERE CONCAT(name, "ович")=secname`
	Возвращаются все записи таблицы, в которых значение поля `name` в соединении со строкой "`ович`" совпадает со значением поля `secname`.
	Функции `ROUND(X)`, `FLOOR(X)` и `CEILING(X)` – используются для округления аргумента. Причем первая использует правила математики, вторая округляет в меньшую сторону, вторая – в большую.
	Существует масса других функций, подробное описание которых будет размещено в блоке "Базы данных".

Группировка в операторе SELECT	При использовании слова `GROUP BY` в операторе `SELECT` результирующие записи группируются по указанным после слова полям. Это означает что если в таблице есть несколько записей с совпадающими значениями такого поля, то в результат будет

	добавлена только одна запись (первая). Этот метод используется вместе со специальными функциями, которые позволяют обработать значения разных записей для одного поля. Наиболее часто используются следующие функции.
	• `MAX()` – максимальное значение поля;
	• `MIN()` – минимальное значение поля;
	• `COUNT()` – общее количество различных значений в поле;
	• `SUM()` – сумма всех значений поля таблицы;
	• `AVG()` – среднее значение поля таблицы.
	`SELECT MIN(age), MAX(age), AVG(age) FROM users`
	Запрос возвращает таблицу с одной записью, включающей столбцы с минимальным, максимальным и средним возрастом людей информация о которых хранится в таблице.
	`SELECT COUNT(*) FROM users WHERE age>20`
	Возвращает таблицу с одной записью и одним полем в котором указывается число записей таблицы `users` у которых значение поля `age` больше `20`.
	`SELECT name, MAX(age), MIN(age), AVG(age) FROM users GROUP BY name`
	Для каждого имени определяется максимальный, минимальный и средний возраст для всех людей информация о которых хранится в таблице `users`. Запрос возвращает таблицу со столькими записями, сколько уникальных имен содержится в поле `name`.
	Если в операторе `SELECT` не указаны столбцы, по которым осуществляется группировка – то функции действуют на все значения поля.

Язык *SQL* также используется и для других манипуляций с базой данных: создание и удаление таблиц, создание ключей, самих баз данных, полная очистка таблиц и т.д. Выше приведена лишь небольшая часть основных сведений и вариантов использования языка *SQL* предназначенная для получения общего представления о его возможностях. Полностью язык *SQL*, включая подзапросы, будет рассматриваться в блоке "Базы данных". В качестве финального примера, для усвоения понимания, возьмем таблицы *book* и *box* со следующим содержимым.

book

id	Author	Name	Year	b_id
1	Пушкин	Сборник	2016	1
2	Толстой	Петр I	2005	1
3	Пушкин	Метель	1952	2

box

id	Name	Placement	Size
1	Шкаф №1	Комната №1	40
2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

Рассмотрим некоторые SQL-запросы и результат их выполнения.

`SELECT * FROM box`
Запрос всех записей таблицы box.

1	Шкаф №1	Комната №1	40
2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

`SELECT Name, Placement FROM box`
Запрос полей `Name` и `Placement` у всех записей таблицы `box`.

1	Шкаф №1	Комната №1	40
2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

`SELECT id, Name FROM book WHERE Year>1980`
Запрос всех значений полей `id` и `name` у записей таблицы `book`, у которых значение поля `Year` больше `1980`.

1	Пушкин
2	Толстой

`SELECT * FROM book WHERE Author="Пушкин"`
Запрос всех записей таблицы `book` у которых значение поля `Author` равно строке `"Пушкин"`.

1	Пушкин	Сборник	2016	1
2	Толстой	Петр I	2005	1
3	Пушкин	Метель	1952	2

```
SELECT * FROM box LIMIT 1, 2
```
Запрос двух записей из таблицы `box` с отступом в одну запись от первой.

2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

```
SELECT * FROM book WHERE box=1 LIMIT 1, 10
```
Запрос записей таблицы `book` у которых значение поля `box` равно `1`. В результат отбираются десять записей с отступом в одну запись от первой.

2	Толстой	Петр I	2005	1

```
SELECT book.id, book.Name, box.Name,
Placement FROM book, box WHERE b_id=box.id
```
Запрос записей из двух таблиц. В результирующей таблице поля `id` и `Name` из таблицы `book`, а также поля `Name` и `Placement` из таблицы `box`. При этом к записи таблицы `book` присоединяются соответствующие поля таблицы `box` при условии, что значение поля `b_id` равно `id` таблицы `box`.

1	Сборник	Шкаф №1	Комната №1
2	Петр I	Шкаф №1	Комната №1
3	Метель	Полка №1	Аудитория №2

```
SELECT MIN(Year) FROM book
```
Запрос минимального значения поля `Year`.

1952

```
SELECT MAX(Size) FROM box WHERE
Placement="Комната №1"
```
Запрос максимального значения поля `Size` среди всех записей таблицы `box`, у которых поле `Placement` равно строке `="Комната №1"`.

40

```
SELECT Author, MIN(Year) FROM book GOUP BY
Author
```
Запрос минимального значения поля `Year` для каждого автора отдельно.

Толстой	2005
Пушкин	1952

```
SELECT COUNT(*) FROM box WHERE Size>20
```
Запрос количества записей в таблице `box`, у которых значение поля `Size` больше `20`.

2

Доступ к базам данных MySQL с помощью расширения MySQLi

MySQLi – это расширение языка *PHP* для доступа к серверу баз данных *MySQL*. Поддерживает как процедурный стиль программирования, так и объектно-ориентированный. Может не работать на старых версиях *PHP* – в этом случае следует использовать функции предыдущего расширения *PHP* для *MySql*.

Процедурный стиль	
Подключение к серверу баз данных	`$mysqli = mysqli_connect($host, $user, $password, $database);` Функция осуществляет подключение пользователя `$user` с паролем `$password` к базе данных `$database` на сервере `$host`. Возвращает идентификатор подключения.
Получение кода ошибки подключения к серверу	`$errno = mysqli_connect_errno($mysqli);` Возвращает код ошибки при подключении к северу; `NULL`, если соединение успешно установлено.
Получение текста ошибки подключения к серверу	`$error = mysqli_connect_error();` Возвращает текстовое описание ошибки при подключении к северу.
Выполнение SQL-запроса	`$res = mysqli_query($mysqli, $query, $resultmode);` Для успешного подключения с идентификатором `$mysqli` выполняется *SQL*-запрос `$query`. Функция возвращает `FALSE` во всех случаях ошибочного выполнения. Для предполагающих возврат данных запросов будет возвращен идентификатор (объект) с данными, в остальных – `TRUE`. Параметр `$resultmode` определяет тип результата запроса: буферизованный (`MYSQLI_STORE_RESULT`, значение по умолчанию) или не буферизованный (`MYSQLI_USE_RESULT`). Первый тип занимает место в ОП и не выдает данные, пока запрос не будет выполнен до конца.

	Не буферизованный тип не занимает место в ОП, выдает данные сразу после начала работы, но количество возвращаемых им записей нельзя определить функциями *MySQLi*. Также нельзя выполнять другой запрос, пока текущий не закрыт.
Код ошибки выполнения SQL-запроса	`mysqli_errno($mysqli);` Возвращает код ошибки при последнем выполнении *SQL*-запроса для данного соединения.
Текст ошибки выполнения SQL-запроса	`mysqli_error($mysqli);` Возвращает описание ошибки при последнем выполнении SQL-запроса для данного соединения.
Определение количества записей в результате SQL-запроса	`mysqli_num_rows($res);` Работает только для буферизированных запросов. В случае не буферизированного запроса будет возвращать некорректное значение.
Получение текущей записи результата SQL-запроса в виде списка	`$row=mysqli_fetch_row($res);` Возвращает запись из результата выполнения запроса `$res`. Каждый следующий вызов функции возвращает следующую запись результата, если следующей записи нет – будет возвращен `NULL`.
Получение текущей записи результата SQL-запроса в виде ассоциативного массива	`$row=mysqli_fetch_assoc($res);` Работает аналогично предыдущей функции, но в качестве ключей массива используется имена полей возвращаемой таблицы.
Очистка результата запроса	`mysqli_free_result($res);` Для не буферизированных запросов невозможно выполнение следующего запроса без вызова этой функции. Для буферизированного запроса очищается оперативная память с результатами его выполнения.
Разрыв подключения к серверу баз данных	`mysqli_close($link);` Соединение с базой данных принудительно разрывается. Возвращает `TRUE` в случае успеха, `FASLE` – в случае ошибки.
Объектно-ориентированный стиль	
Подключение к серверу базы данных	`$mysqli = new mysqli($host, $user, $password, $database);` Создается объект для подключения к базе данных `$database` на сервере `$host` для пользователя `$user` с паролем `$password`.
Получение кода ошибки подключения к серверу	`$mysqli->connect_errno;` Возвращает код ошибки при подключении к серверу. При отсутствии ошибки – `0`.
Получение текста ошибки подключения к серверу	`$mysqli->connect_error;` Возвращает текст с описание ошибки при подключении к серверу. Объект `$mysqli` обязательно должен быть предварительно создан.
Выполнение SQL-запроса	`$res = $mysqli->query($mysqli, $query, $resultmode);` Параметры и возвращаемый результат абсолютно аналогичны соответствующей функции процедурного стиля.
Код ошибки выполнения SQL-запроса	`$mysqli->errno;` Аналогично процедурному стилю функции.
Текст ошибки выполнения SQL-запроса	`$mysqli->error;` Аналогично процедурному стилю функции.
Определение количества записей в результате SQL-запроса	`$res->num_rows;` Аналогично процедурному стилю функции.
Получение текущей записи результата SQL-запроса в виде списка	`$row=$res->fetch_row();` Аналогично процедурному стилю функции.

Получение текущей записи результата SQL-запроса в виде ассоциативного массива	`$row=$res->fetch_assoc();` Аналогично процедурному стилю функции.
Очистка результата запроса	`$res->close();` Аналогично процедурному стилю функции `mysqli_free_result()`.
Разрыв подключения к серверу баз данных	`$mysqli->close();` Аналогично процедурному стилю функции `mysqli_close()`.

Смешивание процедурного и объектно-ориентированного стилей в рамках одного проекта затрудняет работу с программой и не рекомендуется. Выше рассмотрены лишь основные функции расширения, более полный их список приведен в блоке "Базы данных".

Доступ к базам данных MySQL с помощью расширения PDO

PDO – универсальное расширение PHP для работы с любым сервером баз данных.

Подключение к серверу базы данных MySQL	`$DBH= new PDO('mysql:host=server;dbname=name', $user, $pass);` Осуществляется подключение к серверу *MySQL* по адресу `server`. Используется база данных `name`, пользователь `user` с паролем `pass`. При завершении работы скрипта подключение будет автоматически разорвано.

Обработка ошибок при использовании PDO	`try { … }`
	`catch(PDOException $e) { echo $e->getMessage(); }`
	Для обработки ошибок используются генерируемые *PDO* исключительные ситуации, которые в свою очередь обрабатываются блоком `try/catch` *PHP*. Т.е. если в коде блока `try{}` произошла ошибка (исключительная ситуация) – выполнение программы не будет остановлено, а будет передано в блок `catch{}`. Если ошибок не было – код блока `catch{}` не будет выполнен. Информация об исключительной ситуации передается в объект `$e`. Вывод описания ошибки возможно методом `getMessage()`.

Выполнение простых SQL-запросов	`$STH = $DBH->prepare( $query ); // выполнение запроса`
	`$STH->execute(); // без псевдопеременных`
	Метод `prepare()` подготавливает запрос `$query` к выполнению, если это невозможно – возвращает `FALSE`. Метод `execute()` выполняет запрос.
	`$STH = $DBH->query( $query ); // выполнение простого запроса`
	Сразу выполняет запрос `$query` и возвращает результат.

Выполнение SQL-запросов с безымянными псевдопеременными	`// выполнение запроса с безымянными псевдопеременными`
	`$STH = $DBH->prepare("INSERT INTO person values (?, ?)");`
	`$STH->bindParam(1, $name);`
	`$STH->bindParam(2, $id);`
	`$name = "Николай Петров";`
	`$id = "8У-15А-247";`
	`// INSERT INTO person values ("Николай Петров", "8У-15А-247")`
	`$STH->execute();`
	`$name = "Ксения Иванова";`
	`$id = "9У-17А-132";`
	`// INSERT INTO person values ("Ксения Иванова", "9У-17А-132")`
	`$STH->execute();`
	В подготавливаемом запросе вместо данных ставятся символы "`?`", которые затем с помощью метода `bindParam()` привязываются к *PHP*-переменным. Тогда при выполнении запроса методом `execute()` вместо псевдопеременных в нем будут подставлены текущие значения привязанных переменных. При изменении значений переменных можно всего лишь повторно выполнить метод `execute()` – в подготовленный шаблон будут подставлены новые значения. Такой подход имеет смысл при

	многократном выполнении однотипных запросов: уменьшает нагрузку на сервер БД за счет использования им кэширования результатов; защищает от *SQL*-инъекций; позволяет автоматически экранировать символы.
	`// псевдопеременные размещаются в массиве`
	`$pholders=array("Николай Петров", "8У-15А-247");`
	`$ST->execute($pholders);`
	Для удобства псевдопеременные можно не привязывать методом `bindParam()`, а сразу передать их значения для выполнения запроса с помощью массива. Порядок элементов массива соответствует порядку следования псевдопеременных в шаблоне запроса.

Выполнение SQL-запросов с именованными псевдопеременными	`// с именными псевдопеременными`
	`$STH=$DBH->prepare("INSERT INTO person values (:name, :id)");`
	`$STH->bindParam(':id', $id);`
	`$STH->bindParam(':name', $name);`
	`$STH->execute();`
	Принцип работы абсолютно аналогичен предыдущему случаю, но в шаблоне запроса псевдопеременные указываются не символами "?", а с помощью их имен.
	`// псевдопеременные размещаются в массиве`
	`$pholders = array('name'=>'Катя', 'id'=>'98У-15А-247');`
	`$STH->execute($pholders);`
	Также как и для безымянных псевдопеременных, для именованных возможна передача их значений в запрос с помощью массива, но в этом случае не списка, а ассоциативного массива.

Определение количества затронутых в результате выполнения SQL-запроса строк	`$STH->rowCount();` Для операторов `INSRT`, `UPDATE` и `DELETE` возвращает количество затронутых при их выполнении строк. Для оператора `SELECT` возвращение точного количество строк в результирующей таблице не гарантируется.

Получение результата SQL-запроса (перебор записей)	`while( $row = $stmt->fetch() )`
	`{ … }`
	Для возвращающего результат запроса определяет текущую запись из результирующей таблицы. При первом вызове метода будет возвращена первая запись, при каждом последующем вызове – следующая запись таблицы. Если следующей записи нет – метод вернет `FALSE`.

Очистка результата запроса	`$stmt->closeCursor();` Рекомендуется вызывать метод перед выполнением нового *SQL*-запроса.
ID последней вставленной записи	`lastInsertId();` Возвращает значение индекса последней вставленной записи. Удобно использовать для индексов с автоматическим инкрементом.
Разрыв подключения к серверу баз данных	`$DBH = null;` Принудительно разрывает соединения с базой данных.

Расширение MySQLi также предоставляет функции и методы для предварительной подготовки запросов с использованием псевдопеременных, но их рассмотрение необходимо провести самостоятельно.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Что такое внешние модули? Зачем они нужны? Как они подключаются?
2. Что такое библиотека функций? Зачем они нужны? Как они подключаются?

3. В чем отличие конструкций include и require?
4. В чем отличие конструкций include и include_once?
5. В чем отличие конструкций require и require_once?
6. Что такое SQL?
7. Основные операторы языка SQL?
8. Оператор INSERT. Для чего он предназначен, какие формы может иметь?
9. Источники данных для оператора INSETRT: как добавить записи в таблицу из другой таблицы?
10. Оператор UPDATE: назначение, синтаксис, форма записи?
11. Формы оператора SELECT?
12. Условие отбора в операторе SELECT?
13. Функции в операторе SELECT?
14. Сортировка записей и ее типы?
15. Группировка записей: назначение и отличие от сортировки?
16. РаботаSQL- функций с группировкой и без группировки?
17. Ограничение результата SQL-запроса: способы и параметры?
18. Псевдонимы полей в операторе SELECT?
19. Расширение MySQLi – назначение и способ применения?
20. Отличие процедурного и объектно-ориентированного стиля в MySQLi?
21. Буферизированные и не буферизированные запросы в MySQLi?
22. Функции и методы расширения MySQLi?
23. Что такое PDO?
24. Чем отличаются расширения PDO и MySQLi?
25. В чем отличие именованных и неименованных псевдопеременных В PDO?
26. Способы передачи значений в псевдопеременные запроса?
27. Какова цель и преимущества использования псевдопеременных в SQL?
28. Как изменится работа программы, если в листинге В-1. 1 в ссылках подменю не будет передаваться параметр "p"?
29. Почему в листинге В-1. 4 при выводе пагинации в тексте ссылок страницы нумеруются, начиная с единицы, а в адресе ссылок начиная с нуля?
30. В каких случаях имеет смысл делать отдельный SQL-запрос для определения текущей записи из списка выводимых записей таблицы базы данных?
31. Как изменится работа программы в листинге В-1.11, если внутренний блок try-catch вынести в отдельную функцию?
32. Как будет выглядеть сформированная программой на листинге В-1.11 строка, если во вложенный try-catch блок добавить finally, выводящий символ «Z»?

Преобразование типов. Сессии. Калькулятор.

Лабораторная работа № В-2.

ЦЕЛЬ РАБОТЫ

Изучение правил и особенностей *PHP* при преобразовании типов данных в строчных и численных переменных. Реализация в *PHP* механизма сессий.

Механизм сессий – это очень удобный способ хранения промежуточных данных при работе *PHP*-программы. Действительно, очень часто возникают ситуации, когда результат выполнения программы необходимо использовать в следующий раз или даже на других страницах сайта. В предыдущих лабораторных работах мы уже научились делать это, передавая эти данные через *GET*-параметры. Но, во-первых, это нельзя сделать для большого объема данных, во-вторых, эти данные все время видны посетителю сайта: а если мы, например, создаем систему тестирования или другой схожий сервис, где эти данные должны быть скрыты, то такой способ абсолютно не допустим.

Конечно, вполне возможно использовать для хранения данных файловую систему. В локальных приложениях это волне допустимо: программа создает файл, в котором хранит промежуточные результаты и при необходимости читает и изменяет их. Но на веб-сервере одна и та же страница может одновременно загружаться десятками, сотнями и даже тысячами пользователей – если все данные хранить в одном файле, то они неизбежно перепутаются. Использовать для каждого пользователя свой временный файл, определяя его по *ip*-адресу – уже лучше, но у нескольких пользователей вполне может быть один и тот же *ip*-адрес. Кроме того, необходим способ надежно определять, когда пользователь закончил работу с сайтом, и следующая загрузка страницы должна очистить данные и начать работу заново. А когда просто завершилась загрузка одной страницы: в *PHP* программа полностью заканчивает свою работу после загрузки страницы, на которой она размещена.

Все эти проблемы очень элегантно решает механизм сессий. Для программиста сессия предоставляет собой массив, в который он может записывать любые элементы и который доступен для всех PHP-программ на любой странице сайта до тех пор, пока посетитель не закрыл браузер. Это позволяет не только не заботится об идентификации пользователя и необходимости очистки данных, но и не думать об способах хранения данных в файле.

Для этого применяется следующий подход. При каждой загрузке страницы на сервер передается небольшой параметр, в котором хранится так называемый идентификатор сессии. Именно по нему *PHP* определяет из какого хранилища брать данные. Причем, если в браузере доступно использование *cookies* – то идентификатор хранится там и передается на сервер именно через них. Если же нет – то *PHP* автоматически и самостоятельно добавляет для каждой ссылки это идентификатор в качестве *GET*-параметра. Тогда, до тех пор, пока пользователь не закрыл браузер, его идентификатор сессии не изменится, а значит он все время будет работать с одним и тем же хранилищем данных. При этом идентификатор никак не связан с *ip*-адресом: он случайным образом генерируется сервером если механизм сессий был включен, но идентификатор передан не был.

С помощью сессий в *PHP*-программировании реализуется большое количество функциональных возможностей. Например, при необходимости сделать доступным часть страниц сайта только после аутентификации, информация об данных пользователя и о возможности просматривать им эти страницы удобно хранить именно в сессии.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа работы в аудитории, 4 академических часа – самостоятельно.

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу *HTTP* документ, представляющий из себя арифметический калькулятор для целых чисел и десятичных дробей.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Работа оформляется в виде одного *HTML*-документа с интегрированным *PHP*-кодом. При открытии страницы в браузере отображается *POST*-форма со следующими элементами:

- строковое поле для ввода вычисляемого выражения;
- кнопка «Вычислить».

Выражение может содержать целые числа, знаки арифметических операций (плюс "+", минус "-", умножить "*", разделить "/" или ":"), скобки "(" и ")". Посторонние символы, а также нарушение правил математики для арифметических выражений должны приводить к выводу сообщения об ошибке. При нажатии кнопки «*Вычислить*» на сформированной странице перед формой должен быть выведен результат вычисления выражения (либо сообщение об ошибке). Для вычислений запрещается использовать любые функции и методы *PHP* для разбора строк и преобразования (определения) типов данных, кроме приведенных в рекомендациях к структуре программы.

В подвале сайта должна построчно отображаться история вычислений: выражение и полученный результат (в том числе текст ошибки, если он произошла при анализе выражения). Текущий полученный результат вычислений не должен отображаться в истории, но должен попадать туда при следующем обновлении страницы.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Самостоятельно создайте требуемую форму для ввода выражения, пусть для определенности имя поля на ней будет "`val`". Теперь разделим задачу на два этапа:

- разработка функции `calculate()` для подсчета значения выражения без скобок;
- разработка функции `calculateSq()` для подсчета значения выражения со скобками на основе уже разработанной функции `calculate()`.

Обязательно доведите первый этап до конца прежде чем приступить ко второму: это значительно упростит понимание работы программы и облегчит отладку. Без этого велика вероятность запутаться во взаимных вызовах функций и не успеть выполнить работу в срок.

ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЯ БЕЗ СКОБОК

Тогда для вычисления выражения разработаем две пользовательские функции: `calculate()`, которая интерпретирует переданную в нее в качестве параметра строку в виде выражения и вычисляет его значение, и вспомогательную функцию `isnum()`, определяющую является ли переданный в нее строковый параметр на самом деле числом. Т.е. если в нее передана строка "`23450`" – то это число и функция должна вернуть `TRUE`; если же передана строка "`46в62`" или даже "`048`" – то это не число, и функция возвращает `FALSE`.

Листинг В-2. 1

```
if( isset($_POST['val'] )       // если передан POST-параметр val
{
     $res= calculate( $_POST['val'] ); // вычисляем результат выражения
     if( isnum($res) )    // если полученный результат является числом
          echo 'Значение выражения: '.$res;    // вывод значения
     else            // если результат не число - значит ошибка!
          echo 'Ошибка вычисления выражения: '.$res;      // вывод ошибки
}
```

Тогда, если две таких функции определены, то обработчик формы будет выглядеть довольно просто. В программе проверяется наличие переданного с помощью метода *POST* значения параметра "`val`". Если он передан (а все переданные этим методом параметры хранятся в

массиве $_POST), то необходимо вычислить его значение. Для этого используется вызов функции calculate() и сохранение результата ее работы в переменной $res.

Т.к. эту функцию реализуем тоже мы, то запланируем, что в случае ошибки в выражении она будет возвращать текстовое описание ошибки. Тогда для определения успешности вычисления необходимо всего лишь проверить тип возвращаемого значения: если это число – значит оно выводится как результат вычисления; если не число – как сообщение об ошибке. Теперь остается только реализовать эти две функции: рассмотрим сначала функцию isnum().

Листинг В-2. 2

```
function isnum( $x )
{
     // перебираем все символы строки в цикле
     for($i=0; $i<strlen($x); $i++)
          // если в проверяемой строке недопустимый в числе символ
          if( $x[$i]!='0' && $x[$i]!='1' && $x[$i]!='2' && $x[$i]!='3' &&
              $x[$i]!='4' && $x[$i]!='5' && $x[$i]!='6' && $x[$i]!='7' &&
                 $x[$i]!='8' && $x[$i]!='9' && $x[$i]!='.' )
                    return false;  // возвращаем FALSE
     return true; // если все символы строки допустимы - возвращаем TRUE
}
```

Самое очевидное – это перебрать все символы в строке и если хоть один из них окажется не цифрой, то и строка – не число. Поэтому в теле функции размещается цикл, в котором последовательно для каждого *i*-ого символа строки проверяется: если символ отличается от цифры и точки – то возвращается false. Если после проверки всех чисел функция продолжает работу – это значит, что все символы строки цифры или точки. К сожалению, такой проверки недостаточно: если в строке будет несколько точек или она будет начинаться с нескольких нулей, то функция все равно вернет true. Для и пустая строка тоже окажется числом. Модифицируем функцию: для этого дополнительно проверим, не пустая ли строка передана, посчитаем число точек в строке (их не может быть больше одной), проверим, не начинается ли строка с точки или нуля, не заканчивается ли точкой.

Листинг В-2. 3

```
function isnum( $x )
{
     if( !$x ) return false;              // если строка пустая - это НЕ число!
      if( $x[0]=='.' || $x[0] == '0' )    // число не может начинаться с точки или
      нуля
            return false;
      if( $x[ strlen($x)-1 ] == '.' )     // число не может заканчиваться на точку
            return false;
     // перебираем все символы строки в цикле
     for($i=0, $point_count=false; $i<strlen($x); $i++)
     {
          // если в проверяемой строке недопустимый в числе символ
          if( $x[$i]!='0' && $x[$i]!='1' && $x[$i]!='2' && $x[$i]!='3' &&
              $x[$i]!='4' && $x[$i]!='5' && $x[$i]!='6' && $x[$i]!='7' &&
                 $x[$i]!='8' && $x[$i]!='9' && $x[$i]!='.' )
                    return false;          // недопустимые символы в строке
          if($x[$i]=='.')       // если в строке встретилась точка
          {
               if( $point_count )         // если точка уже встречалась
                    return false;   // то это не число
               else                  // если это первая точка в строке
                    $point_count=true;         // запоминаем это
          }
     }
     return true; // все проверки пройдены - это число
}
```

В начале функции идет проверка корректности первого и последнего символа строки – если в первом ноль или точка, или же в последнем символе точка – то это не число. Далее организован

аналогичный предыдущей версии функции цикл. Однако в нем помимо сравнения каждого символа с цифрой или числом производится проверка количества точек в строке. Если их более двух – то это не число.

Для этого в начале цикла инициализируется переменная `$point_count`, ее значение "`true`" будет означать что в строке в процессе перебора символов уже встречалась точка. Именно это мы и делаем: в этом же цикле добавляется еще один условный оператор, в котором проверяется не точка ли текущий символ. Если да, то если значение переменной `$point_count` "`true`", то это означает что точка уже встречалась ранее, а значит строка – не число. Если же нет – то этой переменной присваивается значение "`true`": на следующих итерациях цикла мы будем знать, что символ уже встречался.

Теперь, завершив подготовку, необходимо реализовать функцию `calculate()`, которая и будет собственно вычислять значение выражения. Начнем ее разработку с одного математического действия: сложения.

Листинг В-2. 4

```
function calculate( $val )
{
      if( !$val ) return 'Выражение не задано!'; // если строка пуста – ошибка

        // разбиваем строку на аргументы и заносим их в массив
      $args = explode('+', $val);
        $sum=0;                          // начальное значение суммы аргументов
      for($i=0; $i<count( $args); $i++)    // перебираем все слагаемые
      {
            // если слагаемое не число – прекращаем работу и возвращаем ошибку
            if( !isnum($arg[$i]) )
                  return 'Неправильная форма числа!';
            $sum += $arg[$i];          // суммируем слагаемое с предыдущими
      }
        return $sum; // если все слагаемые числа – возвращаем сумму
}
```

Функция начинает работу с проверки переданного выражения – если это пустая строка, то и делать ничего не надо: сразу возвращаем ошибку. Если выражение передано, оно разбивается на подстроки (разделитель – символ «+»). Фактически такое разбиение формирует массив со слагаемыми: остается только суммировать их для получения требуемого результата. Это и происходит в цикле по всем элементам массива: для каждого слагаемого осуществляется проверка – число слагаемое, или нет. Если нет – работа функции прекращается с возвратом сообщения об ошибке. Если да – то слагаемое прибавляется к текущей сумме, которая и возвращается в виде результата при успешном сложении всех слагаемых. Обратите внимание: любой не являющийся числом аргумент приведет к возврату ошибки: даже строка "*4++4*" приведет к сообщению об ошибке, т.к. второй аргумент – пустая строка, а она не является числом.

Однако согласно заданию к лабораторной работе, программа должна выполнять все четыре математических действия. Поэтому н следующем шаге доработаем функцию таким образом, чтобы она выполняла и сложение, и умножение.

Листинг В-2. 5

```
function calculate( $val )
{
      if( !$val ) return 'Выражение не задано!'; // если строка пуста – ошибка

      if( isnum($val) ) return $val;  // если выражение число – возвращаем его

        // разбиваем строку на аргументы и заносим их в массив
      $args = explode('+', $val);

      if( count($args)>1 ) // если в выражении есть символы «+»
      {
            $sum=0;  // начальное значение суммы аргументов
```

```
            for($i=0; $i<count($args); $i++)     // перебираем все слагаемые
            {
                  $arg = calculate( $args[$i] );  // вычисляем значение слагаемого
                  if( !isnum($arg) )                    // если результат не число
                              return $arg;                    // возвращаем ошибку
                  $sum += $arg;        // суммируем слагаемое с предыдущими
            }
            return $sum;          // если все слагаемые числа - возвращаем сумму
      }

        // разбиваем строку на множители и заносим их в массив
      $args = explode('*', $val);
      if( count($args)>1 ) // если в выражении есть символы «*»
      {
            $sup=1;  // начальное значение произведения аргументов
            for($i=0; $i<count($args); $i++)     // перебираем все множители
            {
                  $arg = $args[$i];               // текущий множитель

                      // проверяем - если множитель не число -возвращаем ошибку
                  if( !isnum($arg) ) return 'Неправильная форма числа!';
                  $sup *= $arg;    // умножаем множитель с предыдущими
            }
            return $sup;    // если все множители числа - возвращаем произведение
      }
        // выражение - не число, но и символов «+» или в нем нет
      return 'Недопустимые символы в выражении';
}
```

Для понимания принципа работы этой функции необходимо вспомнить о рекурсии в программировании: это вызов в функции самой себя. Ясно, что если делать это просто так – то образуется бесконечный цикл вызовов. Поэтому крайне важно правильно определить базу рекурсии: условий при которых функция заканчивает свою работу без рекурсивного вызова самой себя. В нашем случае в базе к попытке вычисления пустой строки добавляется вычисление состоящего из одного числа выражения. Действительно, если выражение просто число – то дальнейшее вычисления не нужны: достаточно просто вернуть его как результат работы функции.

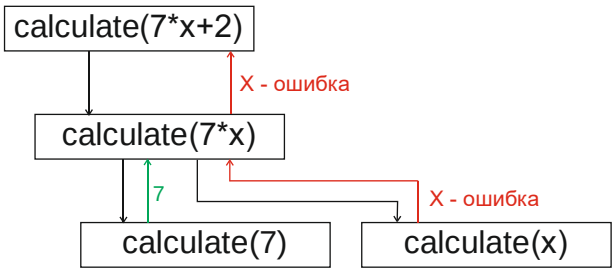
Далее идет разбивка выражение на слагаемые. Если их больше одного, то происходит их суммирование. Но, в отличии предыдущей функции, каждое слагаемое может содержать несколько множителей. Например, выражение "*2+3*4+7*9*" будет разбито на три слагаемых: "*2*", "*3*4*" и "*7*9*". Просто так сложить их мы не можем, поэтому необходимо вычислить каждое из них, что и делается рекурсивным вызовом функции: вычисляем выражения "*2*", "*3*4*" и "*7*9*". Если дальнейшая реализация функции правильная и в результате вычисления мы получим "*2*", "*12*" и "*63*", то окончательный результат вычисления будет равен их сумме. При этом если вычисление значения какого-либо из слагаемых не является числом, т.е. привело к ошибке, то все дальнейшие вычисления прекращаются, а полученная ошибка сразу возвращается. Если же слагаемое только одно – это значит, что возможно в функцию было передано произведение и его необходимо вычислить, поэтому их суммирование и перебор не нужны.

Обратите внимание, что если убрать проверку количества слагаемых, то функция будет работать бесконечно долго. Например, при попытке вычислить значение "*2*3*" мы получим одно слагаемое, которое равно собственно выражению "*2*3*". В цикле мы пытаемся вычислить его значение, рекурсивно вызывая функцию. Ясно, что вызов функции с этим же параметром приведет к такому же результату, т.е. опять-таки вызову функции с тем же самым параметром "*2*3*". И так до бесконечности. Если же перед перебором слагаемых проверять их количество (`if(count($args)>1 )…`), то если их окажется меньше двух бесконечный вызов функций не случится.

Итак, если выражение не просто число и не содержит знаков "*+*", то возможно это произведение. Тогда, по аналогии со слагаемыми, разбиваем выражение на множители. В принципе, обработка произведения аналогична суммированию – только лишь начальное произведение множителей

равно "1", а не "0". Кроме того, рекурсивный вызов из блока произведений не производится, т.к. это конечный результат вычислений.

Если же и множитель в выражении только один, то это означает что в нем нет знаков "*+*", нет знаков "***" и это не число – делаем вывод, что выражение ошибочно, о чем и сообщаем, возвращая соответствующее выражение. Вообще такая реализация задачи хороша тем, что возникшая на любом этапе вычислений ошибка сразу же останавливает дальнейшую работу и будет возвращена в качестве результата самого первого вызова функции.



Например, при вызове функции для выражения "*7*x+2*" из нее будет вызвана функция с аргументом "*7*x*" (попытка вычислить первое слагаемое). Из этой функции – сначала функция с аргументом "*7*" (попытка вычислить первый множитель, вернет 7), а затем – с аргументом "*x*" (второй множитель). Этот вызов вернет ошибку, которая тут же передастся на верхний уровень: функция не будет продолжать вычислять произведение, а тут же вернет текст ошибки. Первая вызванная функция так же не будет продолжать суммирование, а сразу вернет текст ошибки – таким образом всегда сообщение о первой же встреченной ошибке будет возвращено как результат работы функции.

Для завершения реализации функции `calculate()` необходимо самостоятельно добавить в нее поддержку вычитания и деления. Обратите внимание: блоки реализуются аналогичным образом, но крайне важен их правильный порядок: сложение, вычитание, умножение, деление. Кроме того, для вычитания и деления начальным накапливаем значением будет значения первого аргумента в цепочке, а не 0 или 1 для суммы и произведения соответственно. Тогда цикл перебора будет начинаться не с *0*-ого элемента (первого), а с *1*-ого (второго) – учтите это при программировании. Также самостоятельно реализуйте возможность использования в качестве символа деления и "*/*" и "*:*".

Для выполнения всех требований лабораторной работы также необходимо реализовать механизм сохранение истории вычислений, для чего удобно применить сессии.

Листинг В-2. 6

```
<?php
      session_start();    // подключаем механизм сессий
      if( !isset($_SESSION['history']) )  // если первая загрузка страницы
            $_SESSION['history']=array();      // создаем в сессии массив для истории
?>

<?php
          // код для вычисления выражения, статический HTML-код страницы
?>

<php
      // для всех элементов строк с историей вычислений
      for($i=0; $i<count($_SESSION['history']); $i++)
          echo $_SESSION['history'][$i].'<br>';      // выводим строку
      if($_POST['val'] )  // если было вычисление
          // сохраняем его в истории
          $_SESSION['history'][]=$_post['VAL'].' = '.$res;
?>
```

Для этого немного модифицируем код главной страницы: во-первых, в самом начале файл, до начала любого статического *HTML*-кода или до начала любой *PHP*-программы, добавляется

подключающий механизм сессий *PHP*-код. Это необходимо делать именно таким образом, т.к. для сессий используются так называемые заголовки файла, которые должны быть переданы до вывода его содержимого. Поэтому, если перед `session_start()` будет хотя бы один байт содержимого документа, сессии не будут подключены – вместо них появится сообщение об ошибке.

Сразу после подключения сессий, в программе проверяется: не первая ли это загрузка документа. Если первая, то в массиве `$_SESSION` с данными сессии ничего нет – в таком случае мы создаем там пустой массив для хранения истории: при следующей проверке история, пусть даже и в виде пустого массива, уже будет там, а значит проверка на первую загрузку пройдена не будет.

После этого можно добавлять статический *HTML*-код, вычислять результат выражения и т.д. В конце документа, в его подвале, размещается последний фрагмент *PHP*-программы. В первую очередь он выводит содержимое истории построчно, как и сказано в задании к лабораторной работе. Затем, если в *PHP*-программу было передано выражение для вычисления, он добавляет в историю собственно выражение и результат его вычисления (число или текст ошибки), который хранится в переменной `$res` (см. листинг В-2. 1). Таким образом, текущий результат вычисления не выводится в истории, но сохранится в ней для последующего вывода – что и требовалось по условиям лабораторной работы.

Обратите внимание, что если просто обновить страницу (нажать *F5*), то каждый раз в историю будет добавляться новая строка. Действительно, при обновлении страницы в нее поступают те же параметры, что и в прошлый раз, следовательно, *PHP*-программа сработает та же, т.е. добавит в массив с историей вычислений новую строку. С одной стороны, это логически правильно, но с пользовательской точки зрения – абсолютно неверно. Давайте модифицируем код так, чтобы избежать подобной ситуации.

Для этого можно использовать следующий подход. Введем в сессии некоторый элемент, в котором будет хранится номер загрузки документа. Т.е. при каждой перезагрузке документа в рамках одной сессии он будет увеличиваться на 1. Для этого в обработчик первой загрузки страницы (см. листинг В-2. 6) включим инициализацию этого элемента в массиве данных сессии: `$SESSION['iteration']=0`. Кроме этого, при каждой загрузке страницы это счетчик будет увеличиваться на 1. Тогда, при первой загрузке в этом элементе будет хранится 1, при второй – 2, и т.д.

Листинг В-2. 7

```
if( !isset($_SESSION['history']) )   // если первая загрузка страницы
{
      $_SESSION['history']=array(); // создаем в сессии массив для истории
      $SESSION['iteration']=0;      // первая загрузка документа
}
$SESSION['iteration']++;
```

Величину этого счетчика будем загружать в качестве значения для одноименного скрытого поля в форме. Тогда, при каждой отправке данных формы, в качестве параметра в *PHP*-программу будет передаваться значение счетчика на момент генерации *HTML*-кода формы.

Листинг В-2. 8

```
<input type="hidden" name="iteration" value="<?php echo $SESSION['iteration']; ?>">
```

Если теперь в качестве условия возможности обработки данных формы использовать не просто проверку существования переданных параметров, но и совпадение увеличенного на единицу переданного из формы значения счетчика с хранящимся в сессии, то повторная обработка при обновлении страницы происходить не будет.

Листинг В-2. 9

```
// если было вычисление и это не обновление страницы
if($_POST['val'] && $_POST['iteration']+1=$_SESSION['iteration'])
```

```
    // сохраняем его в истории
    $_SESSION['history'][]=$_post['VAL'].' = '.$res;
```

Действительно, если значение счетчика на момент формирования *HTML*-кода формы было 60, то и в обработчик поступит именно это значение. Но при обработке значение счетчика будет уже 61 (это следующая загрузка страницы), значит 60+1=61, т.е. необходимо обработать данные формы. Если же после этого страница была обновлена, то это уже 62 загрузка, но данные формы остались прежними: 60+1≠62, т.е. обработка не требуется. Реализуйте этот механизм для данной лабораторной работы.

Вычисление выражения со скобками

Итак, лабораторная работа полностью выполнена, но в выражении недопустимо использование скобок: доработаем PHP-программу для исправления такого несоответствия. Для этого, во-первых, разработаем новую вспомогательную функцию `SqValidator()`, которая бы проверяла корректность расстановки скобок в выражении: она должна соответствовать правилам математики. В основе работы будем использовать классический вариант алгоритма с подсчетом открывающихся и закрывающихся скобок.

Листинг В-2. 10

```
function SqValidator( $val )
{
    $open=0;                        // создаем счетчик открывающихся скобок
    for($i=0; $i<strlen($val); $i++)          // перебираем все символы строки
    {
        if( $val[$i]=='(' )  // если встретилась «(»
                $open++;          // увеличиваем счетчик
          else
        if( $val[$i]==')' )        // если встретилась «)»
        {
            $open--;        // уменьшаем счетчик
            if( $open<0 )         // если найдена «)» без соответствующей «(»
                return false;   // возвращаем ошибку
        }
    }
     // если количество открывающихся и закрывающихся скобок разное
     if( $open!==0 )
           return false;// возвращаем ошибку
    return true;          // количество скобок совпадает – все ОК
}
```

В начале работы инициализируем переменную в которой будем хранить количество встретившихся открывающихся скобок: естественно, что перед разбором строки она хранит ноль. Затем в цикле мы перебираем все символы переданной в функцию строки (проверяемого выражения). Если текущий символ является открывающейся скобкой "*(*", то счетчик `$open` увеличивается на единицу. Если закрывающейся скобкой – то счетчик уменьшается на 1. При этом если его значение стало меньше нуля – это значит, что количество встретившихся закрывающихся скобок больше количества открывающихся, т.е. анализируется выражение вида "*(2+3))-1*" – т.е. оно нарушает правила и поэтому функция возвращает ошибку.

Таким образом каждая встретившаяся открывающаяся скобка увеличивает счетчик, каждая закрывающаяся – уменьшает. Если в конце работы функции значение счетчика `$open` не равно нулю – значит количество скобок разное и функция опять-таки возвращает ошибку. Такой алгоритм хорош тем, то не только проверяет совпадение числа открывающихся и закрывающихся скобок, но и проверяет порядок их следования, т.е. чтобы каждому символу "*(*" соответствовал свой символ "*)*". Без этого выражение "*9+)3+4(*" было бы при проверке признано верным.

Во-вторых, вновь возвращаясь к собственно реализации вычислению значения выражения, необходимо разработать новую функцию `calculateSq()`, которая собственно и будет анализировать и вычислять значение выражения со скобками. Причем у нас уже имеется вычисляющая выражение без скобок функция `calculateSq()`, поэтому задача вновь

разрабатываемой функции на самом деле проще: выделить в выражении все что в скобках, вычислить эти части выражения, а затем вычислить и все выражение целиком.

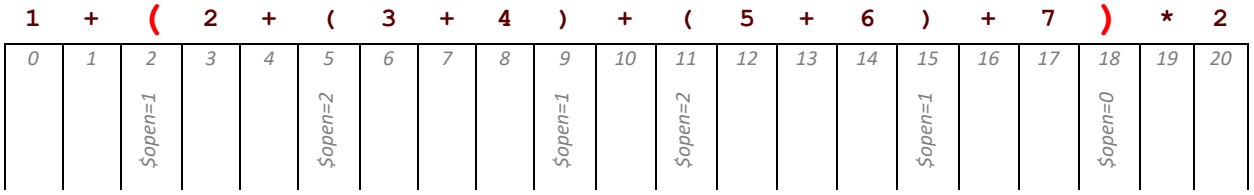
Листинг В-2. 11

```
function calculateSq( $val )
{
       // проверка на корректность использования скобок в выражении
      if( !SqValidator($val) ) return 'Неправильная расстановка скобок';

       $start= strops('(', $val);     // ищем первую открывающуюся скобку

      if( $start===false )            // если в выражении нет скобок
             return calculate($val);       // используем функцию calculate()

       ////////////////////////////////////////////////////////////////////
       // ищем соответствующую открывающейся закрывающуюся скобку
       ////////////////////////////////////////////////////////////////////
       $end=$start+1; // первое место поиска – следующий символ
      $open=1;        // количество найденных открывающихся скобок пока 1 шт.

      // цикл пока скобка не найдена или не дошли до конца строки
      // признаком найденной скобки является обнуление счетчика скобок
      while( $open && $end<strlen($val) )
      {
            if( $val[ $end ]=='(' )         // символ «(» увеличивает счетчик
                  $open++;
            if( $val[ $end ]==')' )         // символ «)» уменьшает счетчик
                  $open--;
               $end++;
       }

       ////////////////////////////////////////////////////////////////////////////
       // формируем новое выражение, путем замены содержимого скобок на вычисленное
       ////////////////////////////////////////////////////////////////////////////
       $new_val = substr($val, 0, $open); // часть исходного выражение левее скобок

       $new_val.=calculateSq( substr($val, $open+1, $end-$open-2) ); // часть в скобках

       $new_val .= substr($val, $end);     // часть исходного выражение правее скобок

      return calculateSq( $new_val ); // вычисляем новое выражение и возвращаем его
}
```

Для работы данная функция, как и `calculate()`, использует рекурсивный подход. В качестве базы рекурсии используется, во-первых, проверка корректности расстановки скобок с помощью разработанной ранее функции `SqValidator()`: если скобки расставлены неверно, то и дальнейшая работа бессмысленна – функция сразу возвращает сообщение об ошибке. Во-вторых, если калькулятор может считать выражение со скобками, это не значит, что он не должен считать выражения без них. Поэтому, и это основная часть базы, осуществляется поиск открывающейся скобки в выражении: если таких скобок нет, то это значит, что значение выражения можно посчитать уже имеющейся в распоряжении функцией `calculate()`. Причем в этом случае в выражении гарантировано отсутствуют и закрывающиеся скобки: иначе функция `SqValidator()` возвратила бы ошибку. Но даже если бы такой проверки не было бы, то передача в функцию `calculate()` выражения с закрывающейся скобкой (да и с любой другой) также привело бы к ошибке. Поэтому дальнейший код выполняется только для правильных выражений с имеющимися скобками – для остальных случаев мы уже умеем получать правильный результат.

Давайте представим, как мы сами вычисляем значение содержащего скобки выражение и постараемся запрограммировать эти же действия на *PHP*. Вкратце этот процесс можно описать следующим образом: если в выражении встретилась открывающаяся скобка, то человек ищет соответствующую ей закрывающуюся и вычисляет находящееся в них выражение, после чего заменяет все то что в скобках полученным числом.

Поэтому, следующая часть функции с помощью стандартной функции *PHP* `strops()` ищет в строке позицию первого вхождения символа "`(`" и сохраняет ее в переменной `$start`. Затем, начиная со следующего символа, мы в цикле перебираем все символы строки до тех пор, пока она не закончится или же не будет найдена соответствующая открывающейся скобке закрывающаяся. При этом используется такой же алгоритм с увеличивающимся и уменьшающимся счетчиком скобок, как и в функции `SqValidator()`: каждый встреченный символ «*(*» увеличивает счетчик скобок на единицу, «*)*» – уменьшает. Значение счетчика `$open` "`0`" является признаком нахождения искомой закрывающейся скобки и цикл прекращается. Например, разберем следующее выражение.



Первая открывающаяся скобка встречается в символе №2 – поэтому значение переменной `$start` равняется "`2`", а счетчика `$open` – "`1`". Далее, по мере работы цикла, при проверке символа №5 открывающаяся скобка встретится вновь, а значит счетчик будет увеличен на 1: `$open=2`. Символ №9 опять уменьшит ее на 1, 11-ый и 15-ый символы сделают тоже самое, пока символ № 18 не обнулит счетчик: это значит, что соответствующая закрывающаяся скобка располагается именно в 18-ом символе. Таким образом, даже если внутри скобок присутствуют другие пары скобок сколь угодно большой степени вложенности, алгоритм найдет именно соответствующую самой первой закрывающуюся скобку.

Итак, в данном примере, после завершения работы цикла, в переменных будут следующие значения: `$start=2`, `$end=19` (обратите внимание, переменная `$end` содержит номер следующего за закрывающейся скобкой символа). Теперь остается только сформировать нового выражение для вычисления состоящее из трех частей:

- то что левее открывающейся скобки;
- то что правее закрывающейся скобки;
- значения выражения в скобках.

Именно это мы и делаем в последней части кода функции: вычисляем значение нового выражение состоящего из "`1+`", "`27`" и "`*2`", т.е. "`1+27*2`" (27 – значение выражения внутри скобок). При этом дважды рекурсивно вызывается та же самая функция `calculateSq()` – для вычисления значения найденного выражения внутри скобок и для значения нового выражения без скобок. Таким образом, каждый вызов функции `calculateSq()` убирает из выражения одну пару скобок до тех пор, пока их совсем не останется и его вычисление будет возможно функций `calculate()`.

Самостоятельно модифицируйте код обработки данных формы так, чтобы обработчик корректно обрабатывал выражения со скобками, тем более что для этого необходимо всего лишь добавить в него два символа.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Подключение механизма сессий	`session_start();` Начинает (или продолжает) работу с сессией. Без вызова этой функции механизм работать не будет. Важно: функция должна быть вызвана до любого статического или динамического формирования *PHP*-кода. Если перед функцией будет передан хотя бы один байт данных, вызов приведет к ошибке.
Инициализация данных в сессии	`if(isset($_SESSION['key'])) $_SESSION['key']=$value;` Все данные сессии доступны через обращение к суперглобальному массиву: он доступен из любого места программы, в том числе из

	любой функции. Его можно использовать как любой другой массив: создавать в нем любое количество элементов, удалять их, изменять значение и т.д. – все эти данные будут доступны при обращении к массиву после перезагрузки документа или даже из другой страницы сайта. Поэтому, если возможна ситуация, когда в программе перед записью данных в элемент массива идет обращение к нему, его необходимо предварительно инициализировать.
Запись данных в сессию	`$_SESSION['key']=$value;` При перезагрузке страницы в элементе массива `$_SESSION['key']` сохранится значение переменной `$value`.
Чтение данных из сессии	`echo $_SESSION['key'];` Читаем и выводим сохраненное в массиве значение.

Вычисление выражения средствами PHP	`eval('$res='.$task.';');`
	`echo $res;`
	Функция интерпретирует строковый аргумент как PHP-код. В данном примере если в переменной $task хранится какое-либо вычисляемое выражение, например, "*2*3+5/2*", то будет выполнен PHP-код: "`$res=2*3+5/2;`", т.е. в переменной `$res` будет сохранено вычисленное значение выражения. Остается только вывести его в браузер.

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Как изменится работа функции из листинга В-2. 4, если из нее убрать проверку на пустую строку?
2. Что такое рекурсия в программировании? Приведите примеры рекурсии.
3. Что такое база рекурсии?
4. Почему математические действия в функции calculate() выполняются именно в такой последовательности?
5. Что такое сессия в PHP?
6. Почему нельзя начинать инициализировать данные сессии после любого вывода?
7. Можно ли инициализировать данные сессии внутри тега <body>, но при до вывода данных средствами PHP?
8. Как PHP разделяет пользователей для доступа к разным хранилищам сессий?
9. Как получить доступ к данным сессии из PHP?
10. Как долго можно обращаться к данным сессии?
11. Можно ли хранить в сессии массивы и строки больше 1024 символов?
12. Как записать данные в сессию?
13. Можно ли прочитать данные из сессии если они туда не записаны? Как быть том случае, если такая ситуация возможна?
14. Как быстро вычислить любое выражение средствами PHP?
15. Можно ли в PHP интерпретировать символ строки как число?

Работа с локальными файлами сервера. Облачное хранилище файлов.

Лабораторная работа № В-3.

ЦЕЛЬ РАБОТЫ

Ознакомление с основными принципами и функциями при работе с локальными файлами на Веб-сервере, включая организацию совместного доступа. Понимание возможных путей реализации механизма аутентификации программными методами *PHP*.

Безопасность и сохранность данных – одни из важнейших требований к современным информационным системам, в том числе и построенным в Интернете. Основным механизмом для ее обеспечения – это подтверждение личности пользователя с помощью пароля. При этом проводится комплекс из трех процедур, которые обычно неотделимы друг от друга – процесс доступа к информации или функционалу сопровождается тремя этапами: идентификация пользователя, его аутентификация и авторизация для проверки прав доступа к информации.

Идентификация

Определение объекта по его характерным свойствам. Например, человек по лицу идентифицирует своих знакомых; по номеру паспорта и ИНН бухгалтерия идентифицирует работника предприятия; инспектор ГИБДД по номеру автомобиля идентифицирует транспортное средство. В компьютерных системах чаще всего для идентификации используется имя пользователя (логин). Но наличие характерных особенностей не позволяет гарантировать подлинность объекта – вполне возможно, что под маской прячется злоумышленник.

Аутентификация

Подтверждение подлинности идентифицируемого объекта. Наиболее распространенным механизмом аутентификации в информационных системах является пароль. Знание секретного слова, которое должно быть известно только системе и пользователю подтверждает и удостоверяет личность последнего.

Авторизация

Проверка права определенного пользователя использовать тот или иной ресурс. Например, администратор сайта получает доступ к полной функциональности административной панели, редактор новостей – только к функциям работы с новостной лентой; системный администратор – доступ к базе данных и т.д. Таким образом, все части информационной системы, включая данные и ее функции, имеют свои права доступа для разных групп пользователей.

Локальные файлы

Сегодня использование локальных файлов на Веб-сервере применяется очень редко. Наиболее часто они применяются при загрузке файлов на сервер с локального компьютера и их последующей обработки: чтения, переноса информации в базу данных и т.д.

Тем не менее до сих пор возможны ситуации, когда хранение информации именно с помощью файловой системы оправдано. Например, для хранения информации о сессии *PHP* использует специальные файлы определенной структуры, при этом в *PHP* есть возможность переопределить функции их обработки, оптимизировав таким образом решение конкретных задач.

Также можно выделить реализацию создания независимых копий баз данных, результатов обработки и т.п., информацию в которых важно сохранить даже при сбое базы данных. Помимо этого аналогично файлам обрабатываются и потоки данных – очень важный объект PHP, речь о котором пойдет в последующих лабораторных работах.

ПРОДОЛЖИТЕЛЬНОСТЬ

8 академических часов работы в аудитории, 8 академических часов – самостоятельно.

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу *HTTP* документы:

- *index.php* – главная страница сайта, использующаяся для аутентификации пользователя;
- *tree.php* – модуль отображения доступных файлов;
- *viewer.php* – модуль отображения содержащейся в файле информации.

Веб-сервис в целом должен позволять загружать файлы на сервер и отображать их информацию в браузере. Необходимо предусмотреть механизм защиты файлов от несанкционированного доступа со стороны других пользователей сервера. Задание должно быть выполнено без использования баз данных.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Модуль *index.php* должен удовлетворять следующим требованиям.

1. При первом открытии страницы в браузере отображается форма для ввода логина и пароля.
2. При отправке формы производится проверка наличия соответствующих данных в специальном файле *users.csv* – если их нет (пользователь не аутентифицирован) – выводится соответствующая надпись и повторно отображается форма. Если данные файле присутствуют (аутентификация пройдена) – запускается второй модуль.
3. Обновление страницы (возврат назад средствами браузера) после успешной аутентификация не должно приводить к потере аутентификации или выводом браузером каких-либо предупреждений (например, о необходимости повторно отправить данные формы).
4. В случае успешной аутентификации, вне зависимости от отображаемой информации, в правом верхнем углу должна размещаться ссылка "Выход" при переходе по которой аутентификация снимается. В этом случае повторно загружается форма ввода логина и пароля.

Модуль *tree.php* должен удовлетворять следующим требованиям.

1. При попытке выполнения без аутентификации выводится сообщение о прекращении работы и дальнейшее выполнение программы приостанавливается.
2. При открытии страницы в браузере отображается содержание текущего каталога (дерево файлов).
3. Каждый элемент дерева выводится в отдельном блоке (тег `<div>`), которые за счет отступов слева формируют легко читаемую структуру дерева каталогов.
4. Элементы типа "каталог" и "файл" должны отличаться друг от друга внешне.
5. Любой элемент "каталог" должен содержать все содержащиеся в нем элементы.
6. Элементы "файл" должны представлять собой ссылки, передающие третьему документу в качестве *GET*-параметра его полное имя. Ссылка должна открываться в другом окне или вкладке браузера.
7. Внизу дерева выводится форма, позволяющая загружать файлы с локального компьютера на сайт. Каталог, в который будет загружен файл указывается пользователем в специальном поле. Если каталог не существует – он должен быть создан. Если указан существующий каталог, но не выбран файл для загрузки – каталог вместе со всеми файлами должен быть удален.
8. Попытки удаления системных файлов и каталогов должны блокироваться.
9. Информация о принадлежности успешно загруженных файлов сохраняется в специальный файл *users.csv*. Имя файла на сервере генерируются как последовательность чисел от 1 с шагом 1; расширение загруженного файла сохраняется. В каждом каталоге последовательность чисел начинается заново.

Модуль *viewer.php* при передаче ему в качестве параметра имени файла должен удовлетворять следующим требованиям.

1. если пользователь не аутентифицирован – выводится ссылка: "Необходима аутентификация!" ведущая на главную страницу сайта, дальнейшее выполнение программы прекращается;
2. если в специальном файле *users.csv* присутствует информация об принадлежности файла не аутентифицированному в настоящее время пользователю – выводится надпись: "Нет прав доступа!";
3. при попытке отображения данных из специального файла *users.csv* выводится надпись: "Секретная информация!";
4. если параметр не передан – выводится надпись: "Имя файла не указано!";
5. если параметр передан, но файл не найден – выводится надпись: "Файл не найден!";
6. если параметр передан и файл существует – выводится содержимое этого файла в браузере, включая *HTML*-теги (если файл содержит тег `<br>`, то в браузере должен быть выведен текст "<br>", а не осуществлен перевод строки);
7. работа с файлами не должна приводить к потенциальным конфликтам доступа между различными пользователями сайта.
8. Если информации о файле в *users.csv* нет – выводится сообщение об этом.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Аутентификация

Отобразить дерево файлов и вывести из них данные очень просто, немного сложнее построить механизм аутентификации. Поэтому начнем разработку программы именно с него, тем более что соответствующий модуль идет первым по порядку. Смысл механизма достаточно прост – необходимо ввести имя пользователя и пароль, и проверить его: существует такой пользователь с таким паролем или нет. При обновлении страницы сайта или загрузке других страниц необходимость повторно вводить имя и пароль отсутствует – это было бы весьма неудобно, поэтому состояние аутентификации должно сохраняться на все время работы с сайтом. Из предыдущих лабораторных работ известно, что сохранять такую информацию можно либо с помощью *cookie*, либо с помощью механизма сессий.

Cookie очень удобный механизм, но все данные хранятся в браузере, а значит доступны любому пользователю локального компьютера – это не безопасно. Поэтому для аутентификации в лабораторной работе будут использоваться сессии – в этом случае снимается еще один недостаток *cookie*: ограничение на размер хранимых данных. Выбрав механизм аутентификации, составим естественно-языковое описание алгоритма его реализации.

1. Если пользователь не аутентифицирован, но в параметрах переданы его имя и пароль – осуществляем проверку существования пользователя. Если она прошла – сохраняем информацию о пользователе в сессии.
2. Если в сессии отсутствует информация об аутентифицированном пользователе – выводим форму для ввода логина и пароля.
3. Если в сессии есть информация об аутентифицированном пользователе – обрабатываем соответствующие параметры, выводим информацию и т.д.

Последовательное выполнения пунктов алгоритма обеспечивает нормальную работу механизма аутентификации. Действительно, при первой загрузке страницы никакой информации о пользователе в сессии не хранится и никакие параметры не переданы. Поэтому первый и третий пункты алгоритма не выполняются – выводится форма для ввода логина и пароля. Далее мы вводим эти данные и опять выполняем алгоритм.

В этом случае в параметрах указаны имя и пароль, но аутентификации пока нет. Поэтому выполняется первый пункт, который может привести к двум вариантам событий: если проверка прошла успешно (пользователь аутентифицирован) – данные о нем сохраняются в сессии. В таком случае п.2 не выполняется, но выполняется п.3 – выводится доступная пользователю информация или же предлагаются соответствующие сервисы. Обратите внимание – при таком подходе

признаком аутентификации пользователя является наличие информации о нем в сессии! При провале проверки (пользователь с таким именем не существует или не верно задан его пароль) – информация в сессии не будет сохранена, поэтому далее будет выполнен только п.2 – форма будет выведена повторно. На языке *PHP* рассмотренный алгоритм можно записать следующим образом.

Листинг В-3. 1

```
<?php
       session_start();    // подключаем механизм сессий
       ////////////////////////////////////////////////////////////////////
       // если аутентификации нет, но переданы данные для ее проведения
       ////////////////////////////////////////////////////////////////////
       if( !isset($_SESSION['user']) && is_set($_POST['login']) )
      {
            ... // попытка аутентификации
            if( /* результат аутентификации успешен */ )
                 $_SESSION['user'] = $user; // сохраняем данные о пользователе
      }
      ////////////////////////////////////////////////////////////////////
      // если аутентификации все еще нет
       ////////////////////////////////////////////////////////////////////
       if( !isset($_SESSION['user']) )
       {
            // выводим форму для аутентификации
              echo '<form name="auth" method="post" action="">
                    <input type="text" name="login">
                    <input type="password" name="password">
                    <input type="submit" value="Войти">
                    </form>';
       }
       else
       ////////////////////////////////////////////////////////////////////
       // если аутентификация успешно произведена
       ////////////////////////////////////////////////////////////////////
       {
              // выводится информация для аутентифицированного пользователя
           echo '<p>Добро пожаловать, '.$_SESSION['user']['name'].'!</p>';
       }
?>
```

Действительно, первый модуль разделен на три условных блока. Если пользователь не аутентифицирован (отсутствует признак этого, а именно наличия в сессии информации о нем, которая в данной реализации хранится в элементе массива `$_SESSION` с ключом `'user'`), но переданы параметры для его аутентификации (п. 1 алгоритма) – производится соответствующая проверка данных и, если она успешна, сохранение полученных при ее выполнении данных о пользователе (например, его ФИО) в элементе массива.

Если пользователь не аутентифицирован и данные не были переданы, либо же аутентификация не удалась – выводится форма для ввода и передачи этих данных на обработку. Иначе (если аутентификация удалась или была проведена при предыдущих выполнениях программы) – выводится информация для пользователя: в данном примере это его имя.

Доработаем предложенный "скелет" программы до требуемой в лабораторной работе функциональности, а именно:

- в случае успешной аутентификации организуем вывод содержимого локального каталога;
- реализуем проверку данных пользователя для его аутентификации (с помощью файла users.csv);
- предотвратим вывод сообщений браузера о повторной загрузке данных.

Первый пункт на данном этапе не должен создать никаких затруднений: ведь исходя из задания, модуль *tree.php* должен делать именно это. Поэтому сразу после вывода приветствия просто подключим его конструкцией `include`, а программирование самой функции отложим на будущее, до момента реализации модуля.

Проверку пользователя и его пароля очень легко реализовать с помощью баз данных, но в задании необходимо использовать файл с именами и паролями. В принципе, это неправильно – но задание есть задание, его необходимо выполнять. Поэтому, с помощью *Excel* создадим специальный файл *users.csv* каждая строка которого будет содержать информацию об одном пользователе: логин, пароль (подробнее о файлах .csv – см. справочную информацию к данной работе) и т.д. Тогда открыв этот файл, считав из него данные и найдя среди строк полученные из формы логин и пароль, можно быть уверенным, что пользователь найден, т.е. аутентификация пройдена. Остается "расшифровать" эту строку и сохранить все данные о пользователе в сессии.

Листинг В-3. 2

```
// попытка аутентификации
$f=fopen('users.csv', 'rt');    // открываем текстовый файл для чтения
if( $f )                  // если файл успешно открыт
{
       while( !feof($f) )            // пока не найден конец файла
       {
           $str = fgets($f);         // читаем текущую строку
           $test_user = explode(';', $str);            // разбиваем строку в массив
           if(trim($test_user[0])==$_POST['login']) // если первый элемент совпадает
           {                                  // с переданным именем
                                              // если второй совпадает
                if(trim($test_user[1])==$_POST['password'])
                {                             // с переданным паролем
                     $user = $test_user;        // найден аутентифицированный
                }                             // пользователь
                break;                // заканчиваем цикл досрочно
           }
       }
       fclose($f);                    // закрываем файл
}
if( is_set($user) )             // результат аутентификации успешен
     $_SESSION['user'] = $user;      // сохраняем данные о пользователе
```

В *PHP*, как и в большинстве языков программирования, для чтения данных из файлов или записи в них, файлы необходимо предварительно "открыть". Именно это мы делаем в первой строке программы: указываем имя файла, с которым собираемся работать и тип операций с данными – в данном случае чтение (`r` – от слова *read*) текстовой информации (`t` – от *text*). Функция `fopen()` возвращает так называем дескриптор файла, т.е. его код по которому *PHP* понимает к какому именно файлу в дальнейшем будет происходить обращение. Если функция вернула `false` – значит файл не существует или же он не может быть открыт. Выполнение этого условия проверяется в следующей строке – если файла нет, то и аутентификация невозможна, т.к. нам неизвестно ни об одном пользователе.

Если же файл успешно открыт, то в цикле можно последовательно прочесть все его строки, это происходит следующим образом. В начале работы данные не считаны, а файловый указатель (номер считываемого при следующей операции байта) установлен на начало. Поэтому, если файл не пустой, функция `feof()` возвратит `false` – она делает это всегда, когда файловый указатель не достигнул конца файла. Т.к. в операторе цикла это условие стоит под операцией отрицания – цикл будет продолжать свои итерации до тех пор, пока не достигнут конец файла, т.е. все данные не прочитаны.

Для чтения данных в примере используется функция `fgets()`, которая читает данные от текущего положения файлового указателя и до конца строки. После этого файловый указатель смещается на следующую после окончания строки позицию. Таким образом в цикле читается из файла первая строка, а файловый указатель передвигается на первый байт после ее окончания. На следующей итерации все проверяется: если указатель достиг конца файла – цикла прекращается, если же нет – аналогично читается следующая строка.

Итак – организован цикл, последовательно обрабатывающий все строки файла *users.csv*. Т.к. каждой строке соответствует описание только одного пользователя, то все что нам остается сделать – это поверить совпадение переданного логина текущему пользователю. Обратите

внимание – т.к. в строки полученные из файла содержат в конце пробельные символы, то лучше использовать функцию `trim()` для их удаления – это помимо всего удалит и ошибочно введенные при заполнении файла пробелы в начале или конце строки. Если логин совпал – то кандидат на аутентификацию найден и за этим следует проверка пароля. Но для описанного процесса необходимо предварительно "дешифровать" строку файла *csv*, "вынув" из нее необходимые данные. Это можно делать с помощью уже известной функции `explode()` – первые два элемента полученного массива и являются логином и паролем текущего пользователя.

Теперь требуемая проверка легко производится условным оператором: сравнивается первый элемент массива с полученным из формы логином пользователя. Если они совпадают – дальнейшие итерации можно прекратить и перейти к сравнению второго элемента массива и переданного пароля. Если пароли совпали – аутентификация произошла, и информация о пользователе сохраняется в специальную переменную `$user`. Если же пароль передан не верно – то переменная так и не будет создана, а дальнейший перебор строк файла – лишняя трата времени.

После окончания цикла необходимо закрыть файл (он будет закрыт и автоматически после завершения программы, но такой подход более правильный). Затем организуется проверка существования переменной `$user`: если в результате выполнения данного кода она появилась – пользователь был аутентифицирован, и информация о нем сохраняется в сессии, если же ее нет – то информации о нем в сессии не появляется. Т.е. признак автооризации полностью реализован предложенным способом.

Разобранный пример абсолютно корректно будет работать, но код несколько "шероховат". Давайте отшлифуем его – ведь чем более красива ваша программа, тем проще с ней работать и вносить необходимые изменения. Заодно немного доработаем код, проверив возможные приводящие к потенциальным ошибкам выполнения программы ситуации.

Листинг В-3. 3

```
// попытка аутентификации
if( $f=fopen('users.csv', 'rt') )          // если файл успешно открыт
{
       while( !feof($f) )              // пока не найден конец файла
       {
           // разбиваем текущую строку файла в массив
           $test_user = explode(';', fgets($f) );
           if(trim($test_user[0])==$_POST['login'] ) // если найден логин
           {
                if( is_set($test_user[1]) &&    // если пароли совпали
                       trim($test_user[1])==$_POST['password'] )
                     $_SESSION['user'] = $test_user; // сохраняем в сессию
                break;     // проверка закончена - следующих проверять нет смысла
           }
       }
       fclose($f);                     // закрываем файл
}
```

В примере открытие файла встроено прямо в условный оператор для проверки его существования: если его не удется открыть, значит будем считать, что он не существует. Строку для передачи в `explode()` получаем непосредственно вызвав функцию `fgets()`, без сохранения результата ее работы в промежуточной переменной `$str`. Для более стабильной работы перед проверкой совпадения переданного пароля и текущего производится проверка существования в полученном массиве элемента с индексом "`1`" – он будет отсутствовать если в строке файла по каким-то причинам нет разделителей элементов. И наконец, нет необходимости вводить промежуточную переменную `$user` и сохранять в нее данные пользователя – можно сразу создать соответствующий элемент массива сессии.

Представим себе работу скрипта при удачной аутентификации: *PHP*-программа получила параметры, обработала их, сохранила данные о пользователе в сессию, вывела необходимую пользователю информацию. Все хорошо, но что произойдет при попытке браузера обновить

страницу (если нажать клавишу *F5*)? Браузер понимает, что обновляемая страница была получена как результат обработки каких-то данных и честно предупреждает об этом пользователя, предлагая утвердить повторную их отправку на сервер. Вполне логично для браузера, но очень неудобно для человека, особенно если учесть, что подобная же ситуация возникает и при переходе с помощью кнопок браузера "Назад" и "Вперед". Для того чтобы избежать этого удобно использовать редирект.

Листинг В-3. 4

```
<?php
       session_start();    // подключаем механизм сессий
       ////////////////////////////////////////////////////////////////////
       // если аутентификации нет, но переданы данные для ее проведения
       ////////////////////////////////////////////////////////////////////
       if( !isset($_SESSION['user']) &&
           is_set($_POST['login']) && is_set($_POST['password']) &&
           $f=fopen('users.csv', 'rt')) // попытка аутентификации
      {
              while( !feof($f) )      // пока не найден конец файла
              {
                 // разбиваем текущую строку файла в массив
                 $test_user = explode(';', fgets($f) );
                 if( trim($test_user[0])==$_POST['login'] )      // если найден логин
                 {
                       if( is_set($test_user[1]) && // если пароли совпали
                                trim($test_user[1])==$_POST['password'] ) // сохраняем
                            $_SESSION['user'] = $test_user;  // в сессию
                       header('Location: /');     // редирект на главную
                       exit();    // дальнейшая работа скрипта излишняя
                 }
              }
              fclose($f);  // закрываем файл
       }
      ////////////////////////////////////////////////////////////////////
      // если аутентификации все еще нет
       ////////////////////////////////////////////////////////////////////
       if( !isset($_SESSION['user']) )
       {
           // выводим форму для аутентификации
              echo '<form name="auth" method="post" action="">
                    <input type="text" name="login">
                    <input type="password" name="password">
                    <input type="submit" value="Войти">
                    </form>';
       }
       else
       ////////////////////////////////////////////////////////////////////
       // если аутентификация успешно произведена
       ////////////////////////////////////////////////////////////////////
       {
              // выводится информация для аутентифицированного пользователя
           echo '<p>Добро пожаловать, '.$_SESSION['user']['name'].'!</p>';
           include 'tree.php';        // выводим содержимое дерева файлов
       }
?>
```

При объедении примеров кода была добавлена проверка существования в передаваемых параметрах не только имени пользователя, но и пароля. В принципе, эта проверка излишне, т.к. ситуация передачи логина без пароля невозможна, но для "универсальности" кода ее можно производить. Принудительное завершение цикла заменено выводом заголовка для перенаправления (редиректа) на главную страницу сайта. Причем для уменьшения трафика и увеличения скорости работы дальнейшая работа *PHP* и вывод каких-либо данных в браузер прекращается, а страница очень быстро и незаметно для пользователя перезагружается. Кроме того, при успешной аутентификации осуществляется подключения модуля отображения дерева каталогов и вывода формы загрузки файла (*tree.php*).

Получившийся вариант кода уже вполне работоспособен, но все же обладает некоторыми недостатками. В первую очередь это невозможность зайти под другим логином: для этого необходима кнопка "Выход". Кроме того, при попытке неудачной аутентификации следует выводить соответствующую надпись: без этого у пользователя будут возникать обоснованные сомнения в работоспособности программы. Ну и, наконец, для удобства следует автоматически заполнять поле `login` переданным в программу значение – если пароль задан с опечаткой, то и вводить заново следует только его; ведь ввод логина в такой ситуации – дополнительное неудобство для пользователя.

Листинг В-3. 5

```
<?php
      session_start();    // подключаем механизм сессий
      ////////////////////////////////////////////////////////////////////
      // обрабатываем выход
      ////////////////////////////////////////////////////////////////////
      if( is_set($_GET['logout']) )       // если был переход по ссылке Выход
      {
          unset( $_SESSION['user'] );     // удаляем информацию о пользователе
            header('Location: /');        // переадресация на главную страницу
          exit();                    // дальнейшая работа скрипта излишняя
      }
      ////////////////////////////////////////////////////////////////////
      // если аутентификации нет, но переданы данные для ее проведения
      ////////////////////////////////////////////////////////////////////
      if( !isset($_SESSION['user']) && is_set($_POST['login']) &&
          is_set($_POST['password']) && $f=fopen('users.csv', 'rt'))
     {
            while( !feof($f) )      // пока не найден конец файла
            {
               // разбиваем текущую строку файла в массив
               $test_user = explode(';', fgets($f) );
               if( trim($test_user[0])==$_POST['login'] )      // если найден логин
               {
                     if( is_set($test_user[1]) && // если пароли совпали
                               trim($test_user[1])==$_POST['password'] ) // сохраняем
                          {
                          $_SESSION['user'] = $test_user;  // в сессию
                          header('Location: /');     // редирект на главную
                          exit();    // дальнейшая работа скрипта излишняя
                     }
                          else            // если пароль не совпал
                          break;          // прекращаем итерации
               }
            }
          echo '</div>Неверный логин или пароль!</div>';
            fclose($f);  // закрываем файл
      }
     ////////////////////////////////////////////////////////////////////
     // если аутентификации все еще нет
      ////////////////////////////////////////////////////////////////////
      if( !isset($_SESSION['user']) )
      {
          // выводим форму для аутентификации
            echo '<form name="auth" method="post" action="">
                  <input type="text" name="login"';
            //если логин уже вводился ранее и был передан в программу
            if( is_set($_POST['login']) )
               echo ' value="'.$_POST['login'].'"';      // заполняем значение поля
            echo '><input type="password" name="password">
                  <input type="submit" value="Войти">
                  </form>';
      }
      else
      ////////////////////////////////////////////////////////////////////
      // если аутентификация успешно произведена
      ////////////////////////////////////////////////////////////////////
      {
            echo '<a href="/?logout=">Выход</a>';   // выводится ссылка для выхода
```

```
            // выводится информация для аутентифицированного пользователя
            echo '<p>Добро пожаловать, '.$_SESSION['user']['name'].'!</p>';
            include 'tree.php';        // выводим содержимое дерева файлов
        }
?>
```

В первую очередь обратите внимание на обработку выхода из аутентификации. Для этого ссылка на главную страницу с *GET*-параметром `logout` выводится над персональной информацией пользователя при подтвержденной аутентификации. В принципе, можно выбрать любое имя параметра и передавать его как в *GET*- так и в *POST*-параметрах – необходимо только правильно написать обработчик. В данном примере код обработки выхода выполняется в самом начале программы: если в массиве `$_GET` найден элемент с ключом `'logout'` – значит был совершен переход по соответствующей ссылке: информация о пользователе из сессии удаляется, после чего происходит переадресация на главную страницу (по тем же соображениям, что и в случае успешной аутентификации).

Вывод сообщения об неудачной аутентификации осуществляется при завершении цикла без ее подтверждения. Обратите внимание: если пользователь найден в текущих, но его пароль указан неверно – переадресация на главную страницу не даст нам возможность вывести требуемое сообщение. Поэтому в примере переадресация осуществляется только в случае совпадения и логина, и пароля; если пароли разные – цикл просто принудительно прекращает свою работу. Соответственно, если в какой-либо итерации цикла аутентификация не произошла – это значит, что или пользователь не найден (ошибка логина), или его пароль указан неверно. В обоих случаях необходимо вывести предупреждающее сообщение, что и было реализовано.

Наконец, при формировании *HTML*-кода формы проверяется наличие в переданных параметрах логина. Если такой параметр был передан – то он передается в соответственном поле как его начальное значение.

Самостоятельно сформируйте *css*-файл для красивого и удобного отображения элементов модуля. Доработайте представленный код таким образом, чтобы при открытии сайта даже после перезагрузки компьютера поля формы `login` и `password` были заполнены последними введенными при отправке формы значениями (используйте *cookies*). Обратите внимание – элементы массива с данными о пользователя в некоторых случаях могут быть дополнительно заключены в кавычки – обработайте эту ситуацию.

Содержимое каталога и загрузка файлов

Следующий модуль используется для отображения существующих на сервере файлов и загрузки на него новых. Фактически это стандартный обработчик данных: он отображает какие-то данные и предоставляет функции для их изменения. Ясно, что обработка должна производится до отображения (если данные сначала отобразить, а уже потом изменить – то пользователь увидит старый вариант данных, а новая информация будет доступна ему только после перезагрузки страницы). Поэтому структура модуля будет иметь следующий вид.

Листинг В-3. 6

```
<?php
        if(  /* необходимо загрузить файл (изменить данные) */  )
        {
            ... // загружаем файл (изменяем данные)
        }

        ... // отображаем содержимое текущего каталога (отображаем данные)

        ... // выводим форму для загрузки файлов (инструменты обработки данных)
?>
```

Начнем описание работы модуля со второй его части – это удобно т.к. без отображения текущего состояния данных очень сложно проверить корректность их обработки. Для построения дерева

каталогов используются функции `opendir()`, `closedir()`, `readdir()` и некоторые другие функции манипулирования файлами. Т.к. необходимо отобразить все элементы каталога (и файлы, и подкаталоги, для которых так же должны отображаться все их подкаталоги), то воспользуемся рекурсивным методом. Для этого построим пользовательскую функцию `outdirInfo()`, выводящую информацию о содержимом указанного каталога.

Листинг В-3. 7

```
function outdirInfo( $name, $path )
{
       echo '<div>';             // начало блока с содержимым каталога
       echo 'Каталог '.$name.'<br>'; // выводим имя каталога

       $dir = opendir( $path );            // открываем каталог
       // перебираем элементы каталога пока они не закончатся
       while( ($file=readdir($dir) ) !== false )
       {
              if( is_dir($file) )                    // если элемент каталог
                     echo 'Подкаталог '.$file.'<br>'; // выводим его имя
              else
              if( is_file($file) )                   // если элемент файл
                     echo 'Файл '.$file.'<br>';       // выводим его имя
       }
       closedir($dir);               // закрываем каталог

     echo '</div>';             // конец блока с содержимым каталога
}
```

В качестве параметров в функцию передается имя каталога, содержимое которого мы хотим вывести, и отдельно полный путь к нему (включая и само имя, например: `outdirInfo( 'mysite', 'htdocs/www/mysite')`). В функции мы предварительно выводим тег `<div>` для оформления с помощью *CSS* соответствующих отступов и другого оформления содержимого каталога, затем выводим его имя (переданный в функцию первый параметр `$name`). Для получения содержимого каталога его необходимо "открыть" и последовательно прочитать все элементы – здесь прослеживается полная аналогия с открытием текстового файла и чтением его строк, только используются не функции `fopen()` и `fgets()`, а `opendir()` и `readdir()`. Перед выводом с помощью функций `is_dir()` и `is_file()` определяется тип элемента каталога (соответственно подкаталог это или файл) и выводится вместе с названием этого элемента. В конце функции каталог необходимо закрыть и вывести закрывающую часть тега `</div>`.

Как результат, функция в отдельном блоке выводит имя каталога и на отдельных строках все его элементы: каталоги и подкаталоги. Теперь, согласно требованию лабораторной работы, доработаем ее так, чтобы выводилась также и информация о подкаталогах.

Листинг В-3. 8

```
function outdirInfo( $name, $path )
{
       echo '<div>Каталог '.$name.'<br>';  // выводим имя каталога

       $dir = opendir( $path );            // открываем каталог
       // перебираем элементы каталога пока они не закончатся
       while( ($file=readdir($dir) ) !== false )
       {
              if( is_dir($file) )                    // если элемент каталог
                outdirInfo( $file, $path.'/'.$file );     // выводим его содержимое
              else
              if( is_file($file) )                   // если элемент файл
                     echo makeLink($file, $path);          // выводим его имя
       }
       closedir($dir);               // закрываем каталог

     echo '</div>';             // конец блока с содержимым каталога
}
```

Теперь функция не просто выводит имя подкаталога, а запускает саму себя для вывода содержимого. При этом достаточно легко скорректировать передаваемые в нее параметры: имя каталога – это имя найденного элемента, а полный путь к нему – предыдущий полный путь плюс имя. Таким образом внутри блока (тега `<div>`) выводятся имена файлов в качестве строк и другие блоки с содержимым подкаталогов, которые также содержат строки и блоки сколь угодно большой глубины вложенности. Самостоятельно доработайте функцию так, чтобы ограничить эту глубину каким-либо параметром – обычно это необходимо для избежания возможных бесконечных вызовов функцией самой себя.

Следует отметить, что полученная программа может приводить к бесконечному циклу или ошибке, т.к. элементами каталога могут являться как родительский каталог (".."), так и сам каталог (".."). Модифицировать функцию так, чтобы исключить эти случаи также необходимо самостоятельно.

Был изменен и способ вывода имени файла – для этого теперь используется специальная функция `makeLink()`, которая формирует ссылку на третий модуль, открывающий переданный ему файл для чтения его содержимого в браузере. В функцию передается два параметра: имя файла и полный путь к содержащему его каталогу. В принципе, возможно и объединение обеих параметров в один: например, в полный путь к файлу. Но тогда нам придется для вывода адреса и текста ссылки вновь "разъединять" их, что несколько портит архитектуру программы.

Листинг B-3. 9

```
function makeLink( $name, $path )
{
	// формируем адрес ссылки
	$link='viewer.php?filename='.$path.'/'.$name;
	// выводим ссылку в HTML-код страницы
	echo '<a href="'.$link.'">Файл '.$name.'</a>';
}
```

Функция формирует и выводит ссылку, ведущую к документу *viewer.php* и передающий в него имя открываемого файла. Чтобы дополнительно обезопасить себя от символов, которые в ссылке могут трактоваться как элементы *URL*, модифицируем функцию, выводя параметры в специальном *URL*-кодированном виде с помощью функции `UrlEncode()`.

Листинг B-3. 10

```
function makeLink( $name, $path )
{
	echo '<a href='.'viewer.php?filename='. UrlEncode($path).
		'/'.$name.'">Файл '.$name.'</a>'; // выводим ссылку в HTML-код страницы
}
```

Действительно, не всякую строку можно передать как *GET*-параметр. Т.е. если нам необходимо передать в *PHP*-программу в параметре *parameter* строку "*&f=#100*", то при формировании ссылки `href="?parameter=&f=#100"` в скрип будет передано два значения: пустая строка в параметре *parameter* и *100* в параметре *f*. Функция `UrlEncode()` позволяет избежать этого, заменяя "опасные" символы понятными браузеру их *URL*-кодами.

Самостоятельно доработайте функцию так, чтобы формируемая ссылка открывалась в новом окне (вкладке) браузера. Обратите внимание: *CSS* описание блока (`<div>`) с отступом слева автоматически придаст выводимым данным форму дерева каталогов – сделайте это, а также придайте в CSS различный внешний вид именам файлов и каталогов самостоятельно.

После подготовки соответствующих функций остается только вызвать их для формирования дерева текущего каталога. Для этого передадим в функцию `outdirInfo()` пустые строки, т.е. начнем вывод прямо от корневого каталога сайта – именно он скорее всего и будет текущим.

Листинг B-3. 11

```
echo '<div id="dir_tree">';	// выводит начало тега блока дерева каталогов
```

```
outdirInfo( '', '' );           // выводит дерево каталогов

echo '</div>';              // конец блока дерева каталогов
```

Для большей стабильности программы пустой относительный путь можно заменить абсолютным, получив имя текущего каталога с помощью функции `getcwd()`. Сделайте это самостоятельно и проанализируйте отличие работы обоих вариантов для вашей операционной системы.

Итак, после того как реализован функционал по отображению данных (вывода содержимого каталога) – следует разработать функции по их изменению, в данном случае для загрузки файлов. При этом необходим *HTML*-код, который бы позволил выбрать загружаемый на сервер файл, механизм его передачи и собственно *PHP*-код для обработки переданного файла. К счастью, *HTML* предоставляет в наше распоряжение удобный механизм загрузки файлов, а именно формы `multipart/form-data` и текстовые поля типа `file`.

Листинг В-3. 12

```
<form method="post" enctype="multipart/form-data" action="/tree.php">
       <label for="dir-name">Каталог на сервере</label>
       <input type="text" name="dir-name" id="dir-name">

       <label for="myfilename">Локальный файл</label>
       <input type="file" name="myfilename">

       <input type="submit" value="Отправить файл на сервер">
</form>
```

Отправка такой формы (обязательно методом *POST*) передаст в скрипт *tree.php* не только параметр `dir-name`, но и целиком выбранный в поле `myfilename` файл. Надо сказать, что возможность загрузки файла может быть запрещена в настройках сервера, причем не только запрещена, но и ограничена: кроме явно задаваемого максимального размера загружаемого файла, влияние оказывает и максимальное время выполнения *PHP*-скрипта, и максимальный размер *POST*-данных. Если для хранения файлов используется база данных – в ней также могут присутствовать настройки по ограничению объема данных в запросе. Самостоятельно изучите все указанные выше настройки сервера.

После удачной отправки такой формы выбранный файл будет загружен на сервер и данныее из него будут записаны во временном файле. Т.к. сразу после окончания работы скрипта последний будет автоматически удален, необходим обработчик загрузки, который бы прочитал данные из файла и сохранил их в нужный каталог под нужным именем (отправил в базу данных, вывел в браузер или обработал данные каким-либо иным образом).

Листинг В-3. 13

```
if( isset($_FILES['myfilename']) ) // были отправлены данные формы
{
     if( isset($_FILES['myfilename']['tmp_name']) ) // если файл загружен
       {
           move_uploaded_file(   $_FILES['myfilename']['tmp_name'],  // копируем
                 makeName(.$_FILES['myfilename']['name']));     // файл с новым именем
           echo 'Файл '.$_FILES['myfilename']['name'].' загружен на сервер';
       }
}
```

Вся информация о загруженном файле хранится в суперглобальном массиве `$_FILES`, откуда она и может быть извлечена методами аналогичными обработки массива `$_POST` или `$_GET`. Причем в качестве хранилища данных о файле используется элемент этого массива с ключевым значением, совпадающим с именем поля типа `file`: в данном примере это `myfilename`. Таким образом, если необходимо загрузить несколько файлов для разных целей (например, копию различных документов) – в форме можно разместить несколько полей, а в массиве `$_FILES` появится несколько элементов.

Признаком отправки формы и попытки загрузки на сервер можно считать наличие в массиве `$_FILES` соответствующего элемента: `$_FILES['myfilename']['tmp_name']`, содержащее имя временного файла. Этот файл можно открыть и прочитать из него данные, или же просто скопировать куда-либо целиком. При необходимости можно узнать о нем и другую полезную информацию, например, имя на локальном компьютере: `$_FILES['myfilename']['name']`.

Однако, пользователь может отправить форму, не выбрав файл – аналогично тому, как он может не заполнить поле. Тогда в массиве `$_FILES` все равно будет содержаться элементы, соответствующие полю типа `file`, но все они будут пусты, что дает возможность отследить такую ситуацию и правильно обработать ее. Например, это происходит при обновлении информации о человеке, содержащую как ФИО, так и его фотографию: если в форме были заполнены поля ФИО, но не указана фотография – сайт должен изменить только их, оставив старую фотографию. Если же фотография была загружена – она должна заменить предыдущее фото.

Согласно заданию, при отсутствии загружаемого файла должен быть удален указанный в поле `dir-name` каталог – модифицируем код для реализации этого требования.

Листинг В-3. 14

```
if( isset($_FILES['myfilename']) ) // были отправлены данные формы
{
     if( isset($_FILES['myfilename']['tmp_name']) ) // если файл загружен
      {
          if( $_FILES['myfilename']['tmp_name'] )    // если файл существует
          {
               // копируем его и выводим сообщение об успешной загрузке
               move_uploaded_file($_FILES['myfilename']['tmp_name'],
                         makeName(.$_FILES['myfilename']['name']) );
               echo 'Файл '.$_FILES['myfilename']['name'].' загружен на сервер';
          }
          else
               rmdir( $_POST['dir-name'] );    // удаляем каталог
      }
}
```

Дополнительный условный оператор выполнит копирование временного файла только если он существует, в противном случае – указанный каталог будет удален. Обратите внимание: функция `rmdir()` удаляет только пустой каталог: чтобы удалить каталог с файлами необходимо предварительно удалить всё его содержимое – файлы и подкаталоги. Таким образом для удаления каталога необходимо разработать рекурсивную функцию `deleteCatalog()`, которая бы при запуске определяла содержимое каталога, проверяла возможность удаления файлов в нем, и, емли это возможно, удаляла их, после чего запускала саму себя для удаления подкаталогов – сделайте это самостоятельно и замените на нее стандартную функцию `rmdir()`.

Для завершения работы над кодом также необходимо реализовать функцию `makeName()`, которая использовалась выше для формирования имени файла на сервере. Для этого необходимо учесть два требования лабораторной работы: полное имя должно содержать каталог, в котором будет находится файл; имя файла генерируется как число от 1 с шагом 1.

Листинг В-3. 15

```
function makeName($filename)
{
     if( !file_exists($_POST['dir-name']) )     // если каталога не существует
     {

          umask(0);                  // сбрасываем значение umask
          mkdir($_POST['dir-name'], 0777, true);          // создаем ее
     }
     $ext = end(explode('.', $filename));

     $n=1;           // начиная с 1 цикл пока существует файл
      while( file_exists($_POST['dir-name'].'/'.$n.'.'.$ext )  // с текущем номером
             $n++;                                    // - увеличиваем номер
```

```
      return ($_POST['dir-name'].'/'.$n.'.'.$ext;          // возвращаем свободное имя
}
```

Функция сначала проверяет существование каталога, в котором будет размещен файл – если его не существует, он создается с максимально возможными правами доступа (см. справочную информацию). Для получения расширения файла разобьем строку из его имени в массив, использовав в качетсве разделителя «;» и возьмем его последний элемент. Затем в цикле перебираются все числа от 1 и далее до тех пор, пока в каталоге существует файл из текущего числа и полученного расширения. Как только такого файла не будет – искомое имя будет найдено и оно возвратится как результат работы функции.

Теперь остается реализовать последний функционал модуля – сохранение информации о принадлежности файла, т.е. о том какой пользователь его загрузил. Для этого проще всего реализовать такой формат: в соответствующей строке файла *users.csv* после указания логина и пароля будет перечислены полные имена всех загруженных пользователем файлов.

Листинг В-3. 16

```
function updateFileList($filename)
{
      $info = file($filename);         // читаем все строки файла в массив
      $f=fopen($filename, 'wt');       // открываем файл для записи
       flock($f, LOCK_EX);       // блокируем файл исключительно

      foreach( $info as $k=>$user )    // для всех строк массива
      {
            $data = str_getcsv($user, ';');       // декодируем данные
            if( $data[0]== $_SESSION['user'][0] )      // если найден пользователь
                  $user.=';'.$filename;           // добавляем к его файлам новый
            fputs($f, $user);                     // сохраняем данные в файл
      }
       flock($f, LOCK_UN );           // разблокируем файл
      fclose($f);                     // закрываем файл
}
```

Реализуем функцию `updateFileList()`, которая читает файл *users.csv* и для каждой его строки проверяет: если она описывает текущего пользователя, то в ее конец после символа «;» дописывается имя загруженного файла. После этого строка (измененная или нет) вновь сохраняется в файле – таким образом реализуется алгоритм: считать все из файла, обработать данные, сохранить их в тот же файл.

Но вполне возможна ситуация, когда во время загрузки файла программа будет проверять возможность аутентификации другого пользователя. Нетрудно видеть, что функция стирает файл *users.csv* – поэтому другой процесс при его чтении вообще не найдет в нем данных, если первый не успел их туда записать. Чтобы этого не случилось реализуется механизм блокировки с помощью функции `flock()`: заблокированный таким образом файл будет недоступен для чтения другим процессом до тех пор, пока не будет разблокирован. Самостоятельно модифицируйте код первого модуля для безопасного чтения файла *users.csv* с помощью механизма блокировки – тогда в описанной ситуации второй процесс будет ждать окончания сохранения данных первым и коллизии не случится.

Обратите внимание: при удалении файлов информация о их принадлежности сохранится в файле users.csv, что может привести к коллизиям если другой пользователь загрузит их: в этом случае окажется что файл будет загружен одновременно двумя пользователями, что невозможно. Во избежание этого при удалении файлов необходимо удалять и информацию о них – сделайте это самостоятельно.

Следует также упомянуть о существовании простого способа загрузки сразу нескольких файлов. Для этого необходимо немного изменить *HTML*-код описания поля типа `file` следующим образом: `<input type="file" name="myfilename[]" multiple>`. Тогда после запуска скрипта в элементе массива `$_FILES['myfilename']['tmp_name']` и других будет не строка, а массив со

строками, что позволяет обработать загрузку любого количества файлов. Но применять такой подход разумно только если все они принадлежат одной категории: при обработке не будет возможности отличить фотографию от скана документа.

Листинг В-3. 17

```
if( isset($_FILES['myfilename']) ) // были отправлены данные формы
{
     foreach( $_FILES['myfilename']['tmp_name'] as $i=>$f )
     {
          echo 'Загрузка '.($i).' – отправлено во временный файл '.$f;
     }
}
```

Для окончания работы над модулем самостоятельно реализуйте указанный тип загрузки нескольких файлов и окончательно скомпонуйте *PHP*-скрипт согласно указанной в листинге В-3.6 структуре.

Вывод содержимого файла в браузере

Третий модуль самый простой из реализуемых в данной работе – он должен выводить содержимое файла в браузере. Для этого используются функции манипулирования данными файла.

Листинг В-3. 18

```
if( !isset($_SESSION['user']) ) { echo 'Необходима авторизация'; exit(); }
$f = fopen( $_GET['filename'], 'rt' );          // открываем файл в текстовом режиме
if( $f )                                  // если файл успешно открыт
{
     $content = '';                       // содержимое файла пока пусто
     while( !feof($f) )                        // цикл, пока не достигнут конец файла
          $content .= fgets( $f );             // читаем строку файла

     echo $content;                       // выводим содержимое файла
     fclose( $f );                             // закрываем файл
}
else
      echo 'Ошибка открытия файла '. $_GET['filename'];
```

Файл открывается для чтения и в случае успеха построчно копируется в переменную `$content`, которая и выводится в браузер. Самостоятельно доработайте код таким образом, чтобы:

- выводилось сообщение при отсутствии параметра filename среди переданных;
- выводилось сообщение при отсутствии на сервере запрашиваемого файла;
- выводить сообщение при обращении к файлу users.csv;
- выводилось сообщение если открываемый файл был загружен другим пользователем – в этом случае содержимое файла не должно отображаться;
- для html-файлов в браузере выводился их html-код, а не внешний вид страницы.

Справочная информация

Файл – это именованная область данных на носителе информации. Работа с файлами – важная часть любого языка программирования. Даже в такой области, как ВЕБ-разработка, файлы используются довольно часто: для загрузки большого объема информации в базу данных, ведения журналов, составление и передача различных отчетов и т.д.

Все функции для работы файлами можно разделить на две группы:

- манипулирование с данными в файлах;
- манипулирование собственно файлами.

Первая группа операций предназначена для работы с информацией в файлах (запись и чтение), тогда как вторая группа – для работы с файлами целиком (копирование, переименование, удаление файлов и т.д.). Рассмотрим каждую группу подробнее.

Манипулирование данными файлов

PHP предоставляет ряд функций для доступа к данным в файлах. Но прежде чем рассматривать сами функции, необходимо понять разницу между двумя режимами доступа: бинарным и текстовым.

Текстовый режим предполагает работу со строками, т.е. основными функциями являются чтение и запись строки. При этом в разных ОС разные признаки конца строки: в *Linux* это символ «\n», в *Windows* – два символа «\r\n». Интерпретатор *PHP* транслирует (преобразовывает) эти признаки для файлов, созданных в других ОС, в поддерживаемый текущей ОС формат. Т.е. при чтении файла в текстовом режиме в ОС *Linux* из него автоматически будут удалены все символы «\r».

Если же они важны, т.е. если файл на самом деле не текстовый (но в нем все равно есть байты, которые совпадают с кодом символа конца строки), то прочитанные данные будут критически отличаться от хранящихся в файле, что недопустимо для их дальнейшей обработки. Поэтому для цифровых файлов используется бинарный режим, предполагающий работу с информацией неопределенного типа. При использовании этого режима записанная и прочитанная информация измеряется байтами, чтение и запись происходит без изменения данных.

Открытие файла	`$f=fopen($filename, $mode);` Открывает файл `$filename` в режиме `$mode`: прежде чем прочесть или записать данные в файл его необходимо открыть – операционная система считает в память его заголовки, подготовится к работе с данными файла. Функция может открывать как локальные файлы, так и файлы на удаленных серверах (используя *URL*). Имя локального файла `$filename` может быть относительным (от текущего каталога, обычно совпадает с каталогом открывающего файл *PHP*-скрипта) или абсолютным. В относительном имени возможно указывать каталоги «.» (текущий каталог) и «..» (родительский каталог). Например, «*../files/file.txt*» – в каталоге, содержащим текущий каталог, необходимо открыть каталог «*files*» и уже в нем найти файл file.txt. Режим доступа к данным файлам (режим открытия файла) `$mode` определяется следующей строкой: • `r` – режим только для чтения, указатель помещается в начало файла. • `r+` – режим для чтения и записи, указатель помещается в начало файла. • `w` – режим только для записи, указатель помещается в начало файла. Если файл существует – все данные из него удаляются (длина файла обрезается до 0). Если файл не существует – создается новый файл. • `w+` – режим для чтения и записи, указатель помещается в начало файла. Если файл существует – все данные из него удаляются (длина файла обрезается до 0). Если файл не существует – создается новый файл. • `a` – режим только для записи, указатель помещается в конец файла. Если файл не существует – создается новый файл. • `a+` – режим только для чтения и записи, указатель помещается в конец файла. Если файл не существует – создается новый файл. Упоминающийся в описании режимов указатель (файловый указатель) – номер байта данных (смещение от начала файла) с которого будет производиться следующая операции чтения или записи в файл. По умолчанию все файлы открываются в бинарном режиме, для открытия в текстовом режиме к параметру необходимо добавить флаг `t` (например, режим доступа `rt` – только для чтения, текстовый режим). Следует также учитывать и саму возможность открытия файла в требуемом режиме: если открывается локальный файл, то это определяется правами доступа, как к самому файлу, так и к содержащей его папке. Если указывается *URL* – то допустимость режима определяется используемым протоколом: например, файл «*http://mysite.ru*» не может быть открыт для записи. Кроме того, директива (чаще всего

<table>
<tr><td></td><td>настройка в файле php.ini) allow_url_fopen может запрещать работу с объектами URL как с обычными файлами. В этом случае для работы с удаленными файлами следует использовать альтернативные способы.
Кроме указанных выше, в функции есть ряд дополнительных параметров, расширяющих ее возможности, но лежащих за пределами данного описания. Функция возвращает идентификатор ресурса (дескриптор файла, указатель файла) для дальнейшей работы других функций, или <code>false</code> в случае какой-либо ошибки.</td></tr>
<tr><td>Закрытие файла</td><td><code>fclose($f);</code>
Закрывает файл с дескриптором <code>$f</code>, соответственно до этого файл должен быть открыт функцией <code>fopen()</code>. Возвращает <code>true</code>, в случае успешного закрытия файла, <code>false</code> – в случае ошибки.</td></tr>
<tr><td>Определение размера файла</td><td><code>filesize($filename)</code>
Функция возвращает для файла с именем в параметре <code>$filename</code> его размер в байтах.</td></tr>
<tr><td>Чтение из файла данных</td><td><code>$str=fread($f, $length);</code>
Функция читает из открытого в поддерживающем чтение данных режиме файла с идентификатором <code>$f</code> блок данных размером <code>$length</code> байт (или менее, если при чтении данных достигнут конец файла). При успешном чтении <code>$length</code> байт файловый указатель смещается на это же количество байт ближе к концу файла, т.е. при следующем обращении к функции будет прочитан новый блок данных. Функция возвращает прочитанные данные в виде строки. Может использоваться совместно с <code>filesize()</code> для чтения всего файла разом.</td></tr>
<tr><td>Запись данных в файл</td><td><code>fwrite($f, $data);</code>
Функция записывает в открытый в предусматривающем запись режиме файл с идентификатором <code>$f</code> блок данных из строки <code>$data</code>. Строка используется как наиболее удобный тип данных в PHP и может содержать любые данные, в том числе и не текстовые. Если в функцию передан необязательный параметр <code>$length</code>, то он определит максимальное количество записанных в файл байт данных. Т.е. запись остановится после <code>$length</code> записанных байт или при записи всех данных <code>$data</code>, смотря, что про изойдет первым. После записи файловый указатель смещается ближе к концу файла на записанное количество байт. Возвращает количество записанных байт или <code>false</code> в случае ошибки.</td></tr>
<tr><td>Чтение строки из текстового файла</td><td><code>$str = fgets($f);</code>
Читает из открытого в текстовом режиме файла одну строку, включая символ окончания строки (если он есть), и возвращает ее как результат. Если указан необязательный параметр <code>$length</code>, то строка будет содержать не более <code>$length-1</code> символов. Сдвигает файловый указатель на прочитанное количество символов ближе к концу файла – обычно (если параметр <code>$length</code> не указан) это начало следующей строки. Для ОС Linux результат последовательного вызова нескольких функции fgets() можно представить следующим образом.
<table>
<tr><th>Содержимое файла ($f)</th><th>Вызов функции</th><th>Результат</th><th>Длина результата</th></tr>
<tr><td rowspan="5">1234\n
5678\n
90AB\n
CDEF</td><td>fgets($f)</td><td>1234\n</td><td>5</td></tr>
<tr><td>fgets($f, 4)</td><td>567</td><td>3</td></tr>
<tr><td>fgets($f, 4)</td><td>8</td><td>1</td></tr>
<tr><td>fgets($f, 100)</td><td>90AB\n</td><td>5</td></tr>
<tr><td>fgets($f, 100)</td><td>CDEF</td><td>4</td></tr>
</table>
Файл из четырех строк («1234», «5678», «90AB», «CDEF») считывается четырьмя вызовами функции <code>fgets()</code>. Первый вызов (без указания длины строки) возвращает первую строку вместе с символом «\n». Второй вызов начинает чтение со второй строки и возвращает 3 символа, т.к. в функцию передано ограничение длины читаемой строки. Третий вызов начинает чтение не с третьей строки, а с символа «8», т.к. именно на него был перемещен файловый указатель. Четвертый вызов, аналогично первому, возвращает строку из 5 символов. Пятый – только строку из 4-х символов, т.к. последняя строка в файле не заканчивается символом «\n».</td></tr>
<tr><td>Перемещение файлового</td><td><code>fseek($f, $offset);</code>
Смещает файловый указатель открытого файла на <code>$offset</code> байтов. Точка отсчета</td></tr>
</table>

указателя	определяется необязательным параметром `$whence`: • `SEEK_SET` – смещение указано от начала файла. • `SEEK_CUR` – смещение указано от текущего положения указателя. • `SEEK_END` – смещение указано от конца файла. По умолчанию (если параметр `$whence` не указан) смещение происходит от начала файла. Функция возвращает «0» в случае успешной смены позиции указателя, «-1» – в случае возникновения ошибки. Например, если открыть текстовый файл и перед чтением строки функцией `fgets()` вызвать данную функцию `fseek($f, 2)`, то строка будет прочитана не с начала, а с третьего символа (первый плюс два).
Определение конца файла	`feof($f);` Функция проверяет текущее положение файлового указателя. Возвращает `true` если достигнут конец файла и `false` в противном случае.
Получение массива строк текстового файла	`$array = file($filename);` Функция читает файл с указанным именем (предварительно открывать файл не надо, необходимо просто указать его имя) и разбивает его на массив строк. Строки включают в себя символ окончания строки («\n» или «\r\n»), причем трансляции символов не происходит – окончание строки будет отмечено именно так, как записано в файле при его создании.
Чтение всего файла сразу	`$str = file_get_contents($filename);` Читает указанный файл с именем `$filename` в одну строку. Предварительно открывать файл не надо – следует просто указать имя файла.

Блокировка и совместный доступ к файлам

Основной целью любой программы на *PHP* является подготовка *HTML*-кода страниц сайта, доступных множеству посетителей одновременно. Поэтому следует учитывать возможность ситуации, при которой к одному и тому же файлу будут одновременно обращаться два процесса. Если оба процесса читают содержимое файла, то ничего страшного. Но если один из них записывает, а другой читает, или же оба записывают – конфликты неизбежны.

Для их разрешения *PHP* предоставляет механизм блокировки файлов с помощью функции `flock($f, $operation)`. Для открытого файла с дескриптором `$f` устанавливается режим блокировки `$operation`, определяемый одним из следующих значений.

- `LOCK_SH` – разделяемая блокировка. Если процесс «A» блокирует файл с помощью такой блокировки, то другие процессы могут открывать его для чтения и читать данные. Однако попытка открыть файл для записи будет блокирована – процесс «B» перед началом записи данных будет ожидать снятия блокировки процессом «A».
- `LOCK_EX` – исключительная блокировка. Никакой другой процесс, кроме установившего блокировку, не может ни читать, ни записывать данные в файл до тех пор, пока блокировка не будет снята. Другие процессы при обращении к файлу приостановят свою работу до тех пор, пока файл не будет разблокирован. Такая блокировка устанавливается перед обновлением (записью) информации в файле.
- `LOCK_UN` – снятие блокировки с файла. Файл считается заблокированным до тех пор, пока блокировка не снята. Если Вы завершили работу с файлом (завершили запись данных) – немедленно снимайте блокировку, чтобы другие процессы могли продолжить свою работу.

Блокировка заставляет другие процессы ожидать его разблокирования. Иногда полезно не ждать этого, а продолжить работу: вывести сообщение об ошибке или обратиться к другому файлу. Для этого к значению параметра `$operation` следует прибавить константу `LOCK_NB` – она запрещает процессу ожидать разблокировки. В этом случае попытка обращения к заблокированному файлу приведет не к ожиданию, а к возврату ошибки. Необязательный третий параметр принимает значение `true` если файл недоступен для блокировки (заблокирован другим процессом). Фактически он дублирует возвращаемый функцией результат.

Листинг В-3. 19

```
$f = fopen('filename.txt', 'a+');     // открываем файл для записи
if( $f )                            // если файл успешно открыт
{
      flock( $f, LOCK_EX );           // блокируем файл исключительно
                                    // если файл используется другим сценарием –
                                          // ожидание разблокировки файла
       fwrite( $f, $data );           // запись в файл данных
       flock( $f, LOCK_UN );          // разблокировка файла
       fclose($f);                    // закрытие файла
}
```

В примере производится безопасная запись данных в файл. Перед началом записи осуществляется попытка исключительной блокировки файла. Если файл используется другими скриптами (или этим же скриптом, но в другом процессе) – интерпретатор ждет разблокировки файла. Запись производится только тогда, когда файл заблокирован текущим процессом.

Листинг В-3. 20

```
$f = fopen('filename.txt', 'r');           открываем файл для чтения
if( $f )                            // если файл успешно открыт
{
      flock( $f, LOCK_SH );           // блокируем файл разделяемо
       $data = fgets( $f );           // чтение из файла данных
       flock( $f, LOCK_UN );          // разблокировка файла
       fclose($f);                    // закрытие файла
}
```

Безопасное чтение из файла осуществляется аналогичным образом. При попытке разделяемой блокировки программа приостановит свое выполнение, пока существует процесс с исключительной блокировкой файла (пока в этот файл другой ппроцесс записывает какие-либо данные).

Листинг В-3. 21

```
function writeFileNow( $base, $data )
{
       $f = fopen( $base, 'w');                  // открываем файл для записи
       while( !flock($f, LOCK_EX+LOCK_NB) )          // цикл пока файл заблокирован
       {
           fclose( $f );              // закрываем файл
           $base .= 'A';              // переименовываем его
           $f = fopen($base, 'w');    // открываем заново
       }
       fwrite( $f, $data );           // запись в файл данных
       flock( $f, LOCK_UN );          // разблокировка файла
       fclose($f);                    // закрытие файла
       return $base;              // возвращаем имя файла
}
```

Рассмотрим пример блокировки файла без ожидания. Пусть функция `writeFileNow()` записывает данные в файл: в начале файл открывается, затем в цикле проверяется возможность блокировки файла. Если файл нельзя заблокировать – он закрывается, формируется новое имя файла, файл снова открывается. На следующей итерации попытка блокировки продолжается. В конце концов находится незаблокированный файл, в который и сохраняются данные, его имя возвращается как результат работы функции. Таким образом блокировка файлов эффективно решает задачу совместного их использования в многопользовательских системах.

Использование CSV-файлов

Формат хранения данных *CSV* (*Comma Separated Values* — значения, разделённые запятыми) – удобных способ передачи табличных данных между различными программами. Он представляет собой текстовый файл, данные которого организованны в таблицу следующим образом: строки

таблицы – это строки в тексте; столбцы (ячейки) таблицы – это данные в строке, разделенные символом «;» или «;». Формат очень удобен тем, что с ним корректно работает *Excel*: таблицу можно преобразовать в такой формат при сохранении. Стоит отметить, что иногда (особенно если в ячейке таблицы присутствует текст с кавычками) – *Excel* при конвертировании «обернет» содержимое ячейки в дополнительные кавычки.

Таблица с данными

Москва	128	Все "ОК"
Санкт-Петербург	2345	В июле
Казань	12742	

CSV-файл с данными

```
Москва;128;"Все "ОК""
Санкт-Петербург;2345;В июле
Казань;12742;
```

В *PHP CSV*-файлы можно обрабатывать как обычные текстовые файлы: открывать, читать и сохранять в них данные – главное придерживаться указанного формата и структуры. Однако есть и дополнительные функции, позволяющие эффективнее работать с такими файлами.

Чтение и разбор сроки из CSV-файла	`fgetcsv($f, $length, $delimiter);` Функция работает аналогично `fgets()`, но возвращает строку, не просто как она есть, а уже разобрав ее в массив, используя разделитель `$delimiter`. Второй параметр `$length` указывает максимальную длину считываемой и разбираемой строки. Второй и третий параметр не обязательны: по умолчанию равны `0` (строка без ограничений, но скорость работы несколько медленнее) и `","`.
Запись данных из массива в файл	`fputcsv($f, $array, $delimiter);` Для записи данных из массива `$array` в CSV-файл также можно использовать специальную функцию. В переменной `$delimiter` указывается разделитель ячеек таблицы. Функция имеет и другие необязательные параметры.
Разбор строки в массив	`str_getcsv($str, $delimiter);` Преобразует строку из разделенных символом `$delimiter` элементов в массив. Аналог функции `explode()`.

Манипулирование файлами и каталогами

Функции манипулирования работают собственно с файлами и каталогами. Они позволяют удалять, переименовывать, копировать файлы и т.д.

Перенос загруженного по *http* файла на сервер	`move_uploaded_file( $src, $dst )` Проверяет был ли файл `$src` загружен по протоколу *http* и, если да – переносит его с переименованием в `$dst` (полное имя файла). При попытке переноса системного или любого другого файла, кроме загруженных по *http*, функция работать не будет. Если файл с именем `$dst` уже существует – он будет заменен. Возвращает `true` при удачном завершении, `false` – в случае каких-либо ошибок.
Проверка был ли файл загружен на сервер по *http*	`is_uploaded_file($filename)` Проверяет был ли указанный файл `$filename` (имя файла уже на сервере) загружен по *http* – применяется для обеспечения безопасности и недопущении доступа злоумышленников к системным файлам.
Копирование файла	`copy($src, $dst)` Копирует файл с именем `$src` в файл с новым именем `$dst`. Если файл `$dst` уже существует – он будет перезаписан заново. В случае успешного копирования возвращает `true`, в случае ошибки – `false`.
Проверка существования файла	`file_exists($filename)` Проверяет существование файла или каталога с указанным именем `$filename`. Если файл или каталог существует – возвращает `true`, если нет – `false`.

Переименование файла	`rename($oldname, $newname)` Переименовывает файл с именем `$oldname` (должен существовать) в файл с именем `$newname` (не должен существовать). Возвращается `true`, если переименование удалось, `false` – если нет.
Удаление файла	`unlink($filename)` Удаляет файл с именем `$filename`. Если удалить файл оказалось невозможно – возвращает `false`.
Создание нового каталога	`mkdir($path, $perms)` Создает пустой каталог именем `$path` и указанными правами доступа `$perms` (см. следующий подраздел). При создании нового каталога, во-первых, следует обращать внимание на существование каталога с таким же именем, во-вторых, у процесса с *PHP*-скриптом должны быть права записи в тот каталог, где он создается. Третий, необязательный параметр определяет: создавать ли указанные в `$path` несуществующие подкаталоги (`true`), или же запрещать (`false` – по умолчанию). Если создать каталог невозможно – функция вернет `false`.
Удаление каталога	`rmdir($path)` Удаляет пустой каталог с именем `$path`. Если каталог содержит файлы или подкаталоги, либо же каталог не может быть удален по другой причине – функция вернет `false`.
Определение текущего каталога	`getcwd()` Возвращает абсолютное имя текущего каталога. Часто применяется для построения абсолютных имен к файлам при модульном программировании.
Изменение текущего каталога	`chdir($path)` Заменяет текущий каталог на указанный в `$path`. Если смена невозможна – возвращает `false`. Если смена произошла успешно – возвращает `true`.
Проверка файла	`is_file($filename)` Функция возвращает `true`, если указанный `$filename` является обычным файлом, `false` – в противном случае.
Проверка каталога	`is_dir($path)` Функция возвращает `true`, если указанный `$path` является обычным каталогом, `false` – в противном случае. Именем каталога может быть не только абсолютный путь, но и относительный, с использованием имен «.» (текущий каталог) и «..» (родительский каталог).
Доступ к элементам каталога (к файлам и подкаталогам)	`opendir($path)` Открывает каталог `$path` для чтения его структуры и возвращает его дескриптор. Если каталог не существует или не может быть открыт по каким-либо иным причинам – функция вернет `false`.
Окончание работы с каталогом	`closedir($handler)` Закрывает предварительно открытый функцией `opendir()` каталог с дескриптором `$handler`. Функция ничего не возвращает.
Чтение элемента каталога	`readdir($handler)` Функция возвращает следующий по порядку (зависит от ОС) элемент открытого функцией `opendir()` каталога с дескриптором `$handler`. Если следующего элемента нет – возвращает `false`.

ПРАВА ДОСТУПА К ФАЙЛАМ

Операционная система для каждого файла и каталога всегда определяет права доступа – т.е. тот набор операций, которые могут совершать с ним те или иные пользователи. Эти права часто не

дают манипулировать файлами или данными так, как это необходимо. Поэтому следует понимать не только как их прочитать и интерпретировать, но и как изменить.

0	7	5	4
Признак шестнадцатеричного числа.	**Права владельца файла**. 1+2+4=7, т.е. права на чтение, запись и исполнение.	**Права участников группы владельца сайта**. 5=4+1, т.е. права только на чтение и исполнение.	**Права обычных пользователей**. 4 – только на чтение файла.

Чаще всего права доступа определяются в виде восьмеричного числа (начинается на ноль) в котором поразрядно определяются права для владельца файла, для группы владельца файла и для всех остальных пользователей. Сумма чисел 4 (доступ на чтение), 2 (на запись) и 1 (на выполнение) определяют искомый доступ.

Кроме того, при работе с файлами на их права накладывается текущее значение маски *umask*: маска *umask* (определяется в настройках *Linux*) по умолчанию равна 0002 для пользователя и 0022 для пользователя *root*. Маска "отбирает" указываемые права поразрядно: 7 – все права, 2 – на запись, 0 – оставляет как есть. Поэтому, чтобы, например, создать каталог с правами 0777 необходимо предварительно сбросить маску 0002 в 0000, иначе созданный каталог будет иметь "меньше прав" – всего 0755. Для изменения прав доступа к файлу или каталогу применяются следующие функции (в зависимости от ОС результат работы функций может отличаться).

Изменение маски прав *umask*	`umask($mask)` Изменяет маску прав доступа umsk на новое значение `$mask` и возвращает предыдущее значение. Если параметр `$mask` не указан – просто возвращает значение маски.
Изменение прав доступа к файлу или каталогу	`chmod($filename, $perms)` Изменяет права доступа к файлу или каталогу `$filename` на `$perms`. Возвращает `true`, если права изменены, `false` – если нет.
Определение прав доступа к файлу	`fileperms ($filename)` Возвращает восьмеричное число – права доступа к файлу `$filename`.

Для улучшения внешнего вида и упрощения программы можно не определять и "декодировать" права доступа, а использовать следующие функции, автоматически учитывающие статус текущего процесса *PHP* для проверяемого файла.

Проверка возможности читать из файла	`is_readable($filename)` В соответствии с правами доступа для процесса с программой *PHP* определяет возможность открытия файла с именем `$filename` для чтения. Возвращает `true`, если файл может быть прочитан, `false` – если нет.
проверка возможности записи в файл	`is_writeable($filename)` В соответствии с правами доступа для процесса с программой *PHP* определяет возможность открытия файла с именем `$filename` для записи. Возвращает `true`, если файл может быть записан, `false` – если нет.
Проверка возможности запустить файл на выполнение	`is_executable($filename)` В соответствии с правами доступа для процесса с программой *PHP* определяет возможность выполнения файла с именем `$filename`. Возвращает `true`, если файл является исполняемой программой и имеет соответствующие права доступа, `false` – если нет.

Использование URL-кодирования

При использовании ссылок с *GET*-параметрами может возникнуть ситуация, когда какое-либо значение может быть интерпретировано как часть *URL*. Например, если параметр содержит строку

с символом «#», то все что будет размещено правее него будет не будет воспринято как его значение. Более того, все параметры правее символа будут интерпретированы как имя якоря, их значения не будут переданы для обработки. Похоже дело обстоит и с другими значимыми символами: «@», «&» и др.

Например, передача параметров *p1=”123#45”* и *p2=”x&y=z”* в адресе ссылки без специальных мер приведет к переходу по URL `?p1=123#45&p2=x&y=z`. Если бы мы не знали заранее о передаваемых параметрах, легко можно было бы предположить, что их три: `p1`, `p2` и `y`. Более того, символ # означает переход на соответствующий якорь, а значит реально в программу придет только значение только одного параметра, да и то не до конца: *p1=”123”*.

Для предотвращения этого разумно при формировании адреса ссылки кодировать значение параметров, заменяя все символы их значений на *URL*-коды. Тогда можно совершенно безопасно передать любую информацию в *GET*-параметре ссылки, будучи уверенным что информация не исказится.

Кодировать все символы строки в URL-коды	`$param = urlencode($str);` Возвращает закодированную для безопасной передачи в браузер строку.
Раскодировать строку из URL-последовательности	`$str = urlencode($param);` Раскодирует строку. При передаче закодированных значений параметров методом GET раскодировать их самостоятельно в PHP-программе нет необходимости – в массиве $_GET они окажутся уже декодированными!

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Назовите отличия идентификации, аутентификации и авторизации?
2. Назовите группы функций для работы с файлами и их отличие друг от друга.
3. Сформулируйте отличие бинарного и текстового режима работы с файлами.
4. Назовите режимы доступа к фалам в PHP.
5. Укажите отличие функций fread() и fgets().
6. Определите файловый указатель, расскажите о перемещающей его функции.
7. Расскажите известные Вам способы чтение файла в строку и массив.
8. Расскажите, когда и как следует проводить проверку существование файла.
9. Сформулируйте отличие при удалении файла и каталога.
10. Расскажите о создание нового каталога и правах доступа к нему.
11. Расскажите об использовании маски прав доступа umask.
12. Расскажите о способе получения элементов каталога. Назовите элементы каталога.
13. Сформулируйте режимы блокировки файла и способы их реализации.
14. Как изменится естественно-языковое описание алгоритма аутентификации после добавления в него возможности выхода (снятия состояния аутентификации пользователя)?
15. Почему в листинге В-3.2 проверка совпадения логина и пароля переданным параметрам разнесены в разные условные операторы?
16. Почему в листинге В-3.2 следует использовать прекращение цикла break? Что изменится в проверке аутентификации, если его убрать?
17. Возможна ли в листинге В-3.3 ситуация, когда при проверке совпадения имени пользователя произойдет ошибка из-за того, что массив $test_user не содержит элемента с индексом '0'?
18. Как изменится результат работы программы, если в листинге В-3.4 заменить конструкцию include на require, include_once, require_once?
19. Как для пользователя изменится работа сайта, если из обработчика выхода в листинге убрать переадресацию и принудительное завершение программы?
20. Как изменится работа программы, если в листинге В-3.6 перенести обработку загрузки файлов в конец программы?
21. В чем отличие кода листинга В-3.7 и В-3.8 кроме приведенных в описании лабораторной работы?

22. Где и каким образом определяются настройки сервера, влияющие на возможность загрузки файлов. Перечислите их.
23. Возможна ли загрузка файла через форму с методом GET?
24. Как организована структура CSV-файлов?
25. Какие специальные функции PHP используются для работы с CSV-файлами? Можно ли обойтись без них? Если да – то как?
26. Необходимо ли в листинге В-3.18 для добавления проверки принадлежности открываемого файла аутентифицированному пользователю обращаться к файлу users.csv? Если да – нужно ли при этом реализовывать механизм блокировки файла? Если использовать блокировку не нужно – то почему, если да – то какого типа блокировку использовать?
27. Какие изменения в какие листинги необходимо внести для добавления к информации о пользователе в файле user.csv его ФИО?

Доступ к удаленным сайтам и серверам. Анализатор страниц сайта.

Лабораторная работа № В-5.

ЦЕЛЬ РАБОТЫ

Ознакомление с возможностями работы *PHP* по работе с удаленными на другие сервера файлами и использованию с этой целью протокола *http*. Введение в принципы построения и понимание возможностей использования регулярных выражений в *PHP*.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа (2 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу *http* документ, формирующий для указанного *URL*:

1. URI страницы;
2. заголовок страницы (тег `<title>`);
3. описание и ключевые слова страницы (теги `<description>` и `<keywords>`);
4. содержание всех тегов `<h1>` и `<h2>`;
5. список всех ссылок на странице, разделенных на три группы:
    - локальные;
    - абсолютные, ведущие на страницы этого-же сайта;
    - абсолютные, ведущие на другие сайты.
6. Для всех внутренних страниц сайта, на которые есть ссылки с проверенной ранее страницы, необходимо повторить все пункты с 1 по 9 включительно.
7. Информация о каждой уникальной странице сайта должна выводится только один раз.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Работа оформляется в виде одного HTML-документа с интегрированным *PHP*-кодом. При первой загрузке страницы должна формироваться форма, содержащая поле для ввода строки и кнопку "Анализ". При нажатии кнопки документ перезагружается, выводя требуемую информацию для переданного *URL* страницы и кнопку "Обратно". *URL* может как содержать имя протокола (начинаться со строки "*http*" или "*https*"), так и нет – анализ должен быть произведен в любом случае. При нажатии кнопки "Обратно" документ перезагружается вновь, формируя *html*-код формы ввода *URL* и кнопки "Анализ".

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Для решения поставленных задач достаточно разработки нескольких пользовательских функций, из которых затем можно построить готовое программное решение:

- получение htm-кода страницы по ее URL;
- получение массива определенных тегов по html-коду;
- получение массива текстов и адресов всех ссылок в html-коде.

СОДЕРЖИМОЕ ФАЙЛОВ НА УДАЛЕННОМ СЕРВЕРЕ

С файлами на других серверах в *PHP* можно обращаться точно так же, как и с локальными – все отличие в том, что прав на манипулирование с ними самими и с их данными у вас гораздо меньше. Но, если это прямо не запрещено в настройках самого *PHP*, вы всегда сможете открыть файл в режиме для чтения и получить содержащуюся в нем информацию. Для этого необходимо

указать полный *URL* к желаемому файлу, включая протокол: "*http://mysite.ru/mydir/myfile.txt*" или "*http:// 77.108.75.40/myfile.txt*" или даже "*https://yandex.ru*" – с точки зрения веб-сервера *URL mysite.ru*, *77.108.75.40* и *yandex.ru* –указывают на определенный файл, содержимое которого и будет возвращено.

Листинг В-5. 1

```
function getHTMLcode( $url )
{
      $ret = '';              // начальное значение контента - пусто
      $f = fopen( $base, 'rt');      // открываем файл для записи
      while( !feof() )        // пока все данные из файла не прочитаны
          $ret .= fgets($f);  // читаем новую строку
      fclose($f);             // закрываем файл
      return $ret;            // врозвращаем данные как результат работы функциии
}
```

Тогда функция для получения содержимого *html*-страницы будет всего лишь открывать файл, читать всю его информацию в одну переменную и возвращать ее как результат своей работы. В принципе именно это и делает стандартная функция `file_get_contents()`, использовать которую и рекомендуется в данном случае: обе они эквивалентны, но стандартная, во-первых, оптимизирована, а во-вторых не требует разработки дополнительного *PHP*-кода.

Но не на всех платформах хостинга и веб-серверах имеется возможность использовать стандартные функции *PHP* для работы с файлами на других серверах: настройки могут запрещать это. В этом случае можно использовать библиотеку *libcurl*, содержащую функции для взаимодействия с серверами по различным протоколам, в том числе и по *http*.

Листинг В-5. 2

```
function getHTMLcode( $url )
{
      $ch = curl_init( $url );                  // инициируем новый сеанс
      curl_setopt($ch, CURLOPT_HEADER, 0);           // заголовки не передаем
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);  // возвратить содержимое файла
      curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);  // разрешить перенаправление
      curl_setopt($ch, CURLOPT_TIMEOUT, 10);        // задать тайм-аут 10 секунд

      $ret = curl_exec( $ch );                  // выполнение запроса
      curl_close($ch);                          // завершение сеанса
      return $ret;                              // возвращаем результат работы
}
```

Библиотека *libcurl* – мощное средство для обмена информацией между серверами. С ее помощью можно не только открывать файлы по *URL* с *GET*-параметрами, но и передавать на другой сервер информацию методом *POST*, в том числе и загружать на него целые файлы. Перед началом работы необходимо открыть сеанс, инициализировать начало работы библиотеки и получить дескриптор сеанса межсерверного обмена, что полностью аналогично получению дескриптора файла при его открытии функцией `fopen()`.

Далее необходимо задать список параметров сеанса – их довольно много, но для данной задачи необходимо обязательно указать один: `CURLOPT_RETURNTRANSFER`. Если он задан и не равен нулю (не равен `false`), то при выполнении запроса данные будут возвращены как результат работы, а не выведены непосредственно в *html*-код: по умолчанию (если параметр не определен) весь ответ сервера (в данном случае содержимое файла) будет передан в *html*-код страницы, как если бы была выполнена инструкция `echo file_get_contents()`.

Другие параметры уточняют способ выполнения сеанса – выводить ли ошибку в случае перенаправления на другую страницу, следовать ли этому перенаправлению, сколько максимально ожидать ответа сервера и т.д. Если необходимо передать в запросе *POST*-данные – это также можно указать в параметрах.

Результат сеанса возвращается функцией `curl_exec()` – если параметр `CURLOPT_RETURNTRANSFER` не указан или же принимает значение `false`, то он направляется непосредственно в *html*- код для вывода в браузере, аналогично инструкции `include` или `require`.

Функции *libcurl* также могут отключаться настройками *PHP*, поэтому имеет смысл объединить оба способа обращения к удаленному файлу в один: таким образом, код с гораздо большей вероятностью будет без ошибок выполняться на различных серверах.

Листинг В-5. 3

```
function getHTMLcode( $url )
{
       try
       {
              // пытаемся инициировать сеанс с помощью cURL
              if( !$ch = curl_init( $url ) )     // инициализируем сеанс, если
       невозможно
                 throw new Exception();     // то генерируем исключительную ситуацию

              curl_setopt($ch, CURLOPT_HEADER, 0);          // устанавливаем параметры
              curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
              curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
              curl_setopt($ch, CURLOPT_TIMEOUT, 10);
              $ret = curl_exec( $ch );                      // выполнениям запрос
              curl_close($ch);                          // завершаем сеанс
              return $ret;                              // возвращам результат
       }
       catch(Exception $e)       // если не удалось использовать cURL
       {
           return @file_get_contents( $url );   // используем стандартную
      }
}
```

Большинство стандартных функций *PHP*, в отличие от его стандартных классов, не поддерживает исключительные ситуации. Поэтому в случае если прочитать файл по `$url` с помощью функции `file_get_contents()` – она вернет false, что при выводе будет преобразовано в пустую строку. Однако, на некоторых платформах при таком вызове будет выведено также и предупреждение, что неудобно для посетителей сайта. Поэтому, для блокировки вывода сообщений об ошибках и предупреждений перед вызовом любой функции указывается символ "`@`" – это не является синтаксической ошибкой.

Поиск тегов и их атрибутов в HTML-коде

Поиск тегов в коде – не такая простая задача. Нюанс заключается в том, что набор атрибутов в тегах не фиксирован: `<p>`, `<p id="p12">`, `<p class="cap">`, `<p id="p12" class="cap">` и даже `<p id="p12" class="cap" title="Комментарий" onclick="alert('!!!!!');">` – вполне допустимые формы атрибута `<p>`. Поэтому просто искать в тексте подстроку `<p>` для определения начала тега абзаца недостаточно – любой атрибут приведет к тому, что тег не будет найден. Для решения задачи разобьем ее на два этапа:

- поиск начала тега и окончания размещенного в нем текста;
- поиск начала размещенного в теге текста.

Иначе говоря, для текста с тегом "`<p class="cap">Текст абзаца</p>`" сначала найдем строку, лежащую между "<p " и "</p>", т.е. "`class="cap">Текст абзаца`". На втором шаге найдем строку, лежащую за символом "`>`", т.е. за окончанием описания открывающейся части тега: "`Текст абзаца`". Таким образом, какие бы атрибуты не присутствовали в теге, мы гарантировано найдем его содержимое. Все что для этого необходимо – это его название, в данном случае "`p`". Учитывая это, разработаем функцию, которая для переданного ей в качестве параметров текста кода и названия тега возвращает массив с элементами из всех его вхождений в данный текст – воспользуемся для этого регулярными выражениями.

Листинг В-5. 4

```
function getALLtag( $text, $tag )
{
      // формируем шаблон для тега
      $pattern='#<'.$tag.'([\s]+[^>]*|)>(.*?)<\/'.$tag.'>#i';
      // получаем массив со строками соответствующими второму этапу задачи
       preg_match_all( $pattern, $text, $ret, PREG_SET_ORDER );
      foreach( $ret as $k=>$v ) $ret[$k] = $v[2];     // берем текст тегов
      return $ret;                    // возвращаем массив с текстами тегов
}
```

Регулярное выражение описывает состав строки, создавая для нее некую маску. С помощью заданного регулярного выражения всегда можно определить соответствует ли строка описанной структуре. Поэтому, в начале функции формируется маска, или шаблон, со структурой тега, например для " ", который будет передан в функцию в виде значения `$tag='p'`, выражение примет следующий вид: "`#<p([\s]+[^>]*|)>(.*?)<\/p>#i`". Чтобы разобраться в нем рассмотрм все части регулярного выражения отдельно – тогда его кажущаяся сложность исчезнет.

Для определения начала и окончания структуры сроки в выражении использован символ "`#`" – поэтому собственно шаблон описывается "`<p([\s]+[^>]*|)>(.*?)<\/p>`". Последний же символ "`i`", расположенный после маски, обозначает применение модификатора шаблона, который указывает на необходимость игнорировать регистр символов в анализируемой строке. Символ "`\`" в регулярных выражениях используется для экранирования и не несет другой нагрузки. Зная это, становится понятно, что шаблону соответствуют строки, начинающиеся на подстроку "`<p`" и заканчивающиеся "`</p>`", кроме того содержащаяся между этими подстроками информация должна соответствовать шаблону "`([\s]+[^>]*|)>(.*?)`", который и представляет наибольшую сложность в рассматриваемом выражении.

Тогда такому шаблону соответствуют строки, в которых:

- присутствует символ "`>`";
- подстрока перед этим символом соответствует шаблону "`([\s]+[^>]*|)`";
- подстрока после символа соответствует шаблону "`(.*?)`".

Символ точки в шаблоне соответствует любому символу, символ "`*`" – квантификатору, обозначающему ноль или более вхождений, символ "`+`" – одно или более вхождений, "`?`" – преобразование жадного квантификатора в ленивый, "`^`" – отрицание для последовательности символов, "`\s`" – пробелу.

Тогда шаблон "`(.*?)`" можно прочесть следующим образом: любой символ повторяющийся ноль или более раз с ленивым квантификатором. Первый шаблон немного сложнее – пробел, повторяющийся один или более раз, после которого идет любой символ, кроме "`>`", или же ничего. Таким образом все части шаблона описывают соответствующие ему части *html*-тега: наглядно это можно увидеть в следующей таблице.

Части регулярного выражения	`<p`	`([\s]+[^>]*|)`	`>`	`(.*?)`	`<\/p>`
Описание части регулярного выражения	Подстрока "<p"	Или строка из повторяющегося один или несколько раз пробела, после которого идет ноль или более любых символов, кроме символа ">", или же пустая строка.	Подстрока ">"	Любой символ, повторяющийся ноль или более раз. Квантификатор ленивый.	Подстрока " "
Соответствующие выражению теги с различными атрибутами	`<p`		`>`	Абзац	`</p>`
	`<p`	_ _ _	`>`	Абзац	`</p>`
	`<p`	_ class="red"	`>`	Абзац	`</p>`
	`<p`	_ _ title="Важно!"	`>`	Абзац	`</p>`
	`<p`	_ onclick="alert(1);"	`>`	Абзац	`</p>`

Обратите внимание – данное регулярное выражение не будет правильно обрабатывать строки, содержащие тег в теге: например, разработанный шаблон некорректно выделит содержимое абзаца в строке " 1 2 3 ". К счастью, в лабораторной работе требуется обнаружить теги <title>, <description>, <keywords>, <h1>, <h2> и <a>, которые не предполагают такого варианта использования.

Для выделения соответствующих регулярному выражению подстрок в тексте используется функция preg_match_all(), параметрами которой являются шаблон, текст и массив в которой будет заносится информация о найденных соответствиях шаблону – в данном случае информация о найденных тегах. Важным параметром является флаг PREG_SET_ORDER – он определяет структуру массива в которой каждый следующий элемент хранит полную информацию о найденном соответствии – такой массив более понятен и с ним легче работать.

Тогда, последовательно перебирая массив $ret в каждом его элементе можно увидеть другой массив, в котором:

- элемент с индексом [0] – содержит всю соответствующую шаблону строку;
- элемент с индексом [1] – содержимое первой подмаски (части маски, заключенной в скобки);
- элемент с индексом [2] – содержимое второй (если она есть);
- и т. д.

Т.к. в нашем выражении подмасок (выражений в круглых скобках) две – то размер массива будет равняться трем: тег целиком, атрибуты тега и текст тега. Следовательно, чтобы получить массив с текстом всех тегов необходимо просто собрать все элементы с индексом [2] в одномерный массив, что и реализовано в функции.

Листинг B-5. 5

```
$code = getHTMLcode( $_POST['url'] );      // определяем html-код страницы

$titles = getALLtag( $code, 'title' );               // получаем массивы
$descriptions = getALLtag( $code, description);// с соответствующими тегами

$keywords = getALLtag( $code, 'keywords' );

$h1 = getALLtag( $code, 'h1' );
$h2 = getALLtag( $code, 'h2' );

$a = getALLtag( $code, 'a' );
```

Таким образом, получив html-код страницы и последовательно запустив функцию поиска всех требуемых тегов можно получить массивы с их текстом. Для тегов <title>, <description>, <keywords>, <h1> и <h2> этого вполне достаточно, для тега <a> дополнительно необходимо получить список адресов. Это также достаточно просто сделать, т.к. адрес ссылки – это атрибут href тега <a>, а строка с описанием всех атрибутов для каждого тега располагается в элементе массива с индексом [1]. Модифицируем функцию getALLtag() так, чтобы для тега <a> возвращаемый массив содержал не только текст, но и адреса.

Листинг B-5. 6

```
function getALLtag( $text, $tag )
{
      // формируем шаблон для тега
      $pattern='#<'.$tag.'([\s]+[^>]*|)>(.*?)<\/'.$tag.'>#i';
      // получаем массив со строками соответствующими второму этапу задачи
       preg_match_all( $pattern, $text, $ret, PREG_SET_ORDER );
      foreach( $ret as $k=>$v )  // для всех найденых тегов
        {
              if( $tag == 'a' )       // если мы искали входдения тега <a>
              {
                 $href = '';     // определяем адрес ссылки
                     preg_match( '#(.*)href="(.*?)"#i', $v[1], $arr);
                 if( $arr )                  // если успешно
```

```
                    $href = $arr[2];      // сохраняем адрес в переменной

                // возвращаем адрес и текст ссылки
                $ret[$k] = array( 'href'=>$href, 'text'=>$v[2]);
            }
            else                        // иначе
                $ret[$k] = array( 'text' => $v[2] ); // возвращаем текст тега
        }
        return $ret;                    // возвращаем массив с текстами тегов
}
```

Теперь, получив необходимую информацию о странице с помощью построенных выше функций, достаточно просто вывести ее в приемлемом виде, согласно требованиям лабораторной работы – сделайте это самостоятельно. Для последнего шага, а именно вывода информации по всем внутренним страницам сайта, на которые ведут найденные ссылки, удобно использовать рекурсивный подход.

Листинг В-5. 7

```
function getINFO( $url, $deep )
{
       global $MAX_DEEP;         // читаем максимальную глубину из внешней переменной

       if( $deep>$MAX_DEEP ) return; // если превышен максимальный уровень вложности

       $code = getHTMLcode( $url );  // определяем html-код страницы

       $titles = getALLtag( $code, 'title' );              // получаем массивы
       $descriptions = getALLtag( $code, description);     // с информацией

       $keywords = getALLtag( $code, 'keywords' );

       $h1 = getALLtag( $code, 'h1' );
       $h2 = getALLtag( $code, 'h2' );

       $a = getALLtag( $code, 'a' );

       foreach($a as $link)                        // для всех страниц по ссылкам
           getINFO( $link['href'], $deep+1 );      // выводим информацию о них
}

$MAX_DEEP = 8;             // инициируем максимальную глубину обхода сайта
getINFO( $_POST['url'], 1 );    // запускаем обход сайта с исходного URL
```

В функцию `getINFO()` передается не только *URL* анализируемой страницы, но и текущее значение глубины просмотра. В начале своей работы функция проверяет: если это значение превысило максимальный порог – то дальнейший анализ не производится. Отметим, что максимальное значение хранится в переменной `$MAX_DEEP`, которая инициализируется вне функции – а значит, ее значение не будет видно внутри нее. Для того чтобы переменная стала глобальной, необходимо явно указать это с помощью инструкции `global`, либо, как вариант, взять ее из массива `$GLOBALS` (см. Справочную информацию к данной лабораторной работе).

После анализа кода и вывода информации функция вызывает для всех найденных ссылок саму себя, т.е. рекурсивно обходит дочерние страницы, увеличивая для них свою собственную глубину обхода на единицу. Однако просто открывать все ссылки нельзя: если ссылка внешняя и ведет на другой сайт, то анализировать такую страницу не имеет смысла – поэтому необходим инструмент классификации ссылок, к тому же такое требование предъявляет и задание лабораторной работы. Реализуем для этого специальную функцию, возвращающую "1" – для глобальных ссылок, "2" – для глобальных ссылок на этот же сайт и "3" – для локальных ссылок.

Листинг В-5. 8

```
function getLINKtype( $href, $url )
{
```

```
        // если в адресе ссылки нет протокола – ссылка локальная
        if( strops('://', $href)===false ) return 3;
        // выедляем в ссылке имя сервера
        $domen = parse_url($href, PHP_URL_HOST);
        // если  имя сервера в ссылке равно текущему имени сервера
        if($domen == parse_url($url, PHP_URL_HOST) )
             return 2;                  // глобальная ссылка на этот же сайт
        return 1;                       // иначе ссылку считаем глобальной
}
```

В качестве параметров в функцию передается как адрес проверяемой ссылки, так и переданный *URL* сайта для которого строится структура. Тогда проанализировав ссылку и определив совпадение или несовпадение имени домена в ней и в *URL* можно установить ее тип. Если в адресе ссылки нет признака используемого протокола (`http` или `https`) – ссылка является локальной. Если есть – проверяется совпадение имени сервера в ссылке, выделенное с помощью стандартной функции `parse_url()`, с именем сервера из переданного изначально *URL*. Если они совпадают – то это глобальная ссылка на тот же сайт. Если нет – это просто глобальная ссылка.

Самостоятельно выполните остальные требования лабораторной работы, указав при выводе для каждой ссылки ее тип. Обратите внимание, что при рекурсивном вызове функции `getINFO()` в нее следует передавать только локальные ссылки или глобальные, ведущие на тот же сайт. Причем в первом случае необходимо преобразовать ссылку в глобальную, иначе файл не откроется.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

РАБОТА С ФАЙЛАМИ НА ДРУГИХ СЕРВЕРАХ

Самый простой способ прочитать файл на удаленном сервере – это использовать директиву `include`. Если в качестве имени файла указать какой-либо *URL*, то в *html*-код будет выведено содержимое соответствующего ему файла, причем, полученное содержимое будет завесить от используемого протокола. Например, если в *URL* указан *http*/*https* протокол – то полученные данные, как и в других случаях его использования, будут содержать результат выполнения *PHP* и других серверных скриптов. Если же используется *ftp* – данные будут получены без изменений.

Инструкция	**Сформированный html-код**
`include 'http://www.site.ru/';`	Привет, мир!
`include 'http://site.ru/index.php';`	Привет, мир!
`include 'ftp://user:password@site.ru/index.php';`	<?php echo 'Привет, мир!'; ?>

Не всегда такой доступ к файлам разрешен на веб-сервере: если опция *allow_url_include* или *allow_url_fopen* в файле *php.ini* выключена, то файлы будут недоступны.

Второй способ – это открыть удаленный файл с помощью стандартных функций работы с файлами, что становится возможным при включенной опции *allow_url_fopen* в файле *php.ini*. Работа с таким файлом ничем не отличается от работы с локальными файлами, за исключением ограничений протокола: нельзя открыть файл на запись через *http* протокол, но можно используя протокол *ftp*.

Открытие удаленного файла для чтения	`$f=fopen('http://www.site.ru/index.php?id=2', 'rt');` Файл открывается для чтения в текстовом режиме. Для веб-сервера любой URL является документом (файлом), содержимое которого необходимо передать клиенту (обычно браузеру). Поэтому доменное имя сайта с указанным протоколом также является файлом, который можно открыть для чтения.
Открытие удаленного файла для записи	`$f=fopen('ftp://user:password@site.ru/index.php', 'w');` Для записи в файл на другом сервере необходимо использовать протокол ftp. Для этого, как правило, требуется логин и пароль для доступа к серверу.
Чтение всего файла	`$content = file_get_contents('http://www.site.ru/?id=2');` Для работы с удаленными файлами можно использовать все стандартные функции PHP, в том числе и `file_get_contents()`.

Преимуществом второго способа является возможность получения содержимого файла в качестве значения какой-либо переменной для последующего анализа, но к сожалению не всегда такой доступ разрешен настройками *PHP*. Кроме того, с его помощью на удаленный сервер можно передать *GET*-параметры, но затруднительно *POST*-данные: а без этого невозможно полноценно реализовать межсерверный обмен.

Для удобной и гибко настраиваемой работы с другим сервером, в том числе для получения сформированных по переданным *GET*- или *POST*-параметрам страниц, можно использовать библиотеку функций *cURL*. Принцип работы ее функций аналогичен работе с локальными файлами: сеанс связи с другим сервером необходимо инициализировать, настроить его параметры, выполнить и закрыть. Добавляемый пункт настройки параметров аналогичен определению типа доступа к локальному файлу через функцию `fopen()`, просто в случае *cURL* опций гораздо больше и их установка вынесена в отдельную функцию.

Инициализация сеанса cURL	`$d = curl_init();` Функция возвращает дескриптор сеанса межсерверного общения, т.е. идентификатор по которому однозначно будет определяться сеанс.

Установка параметров сеанса связи	`curl_setopt( $ch, OPTION, $value );`
	Для указанного сеанса устанавливается значение опции. Обычно для сеанса необходимо указать несколько опций – это можно сделать, последовательно вызвав функцию необходимое количество раз в произвольном порядке. Следующие опции используются наиболее часто.
	`// адрес страницы на сервере`
	`curl_setopt($ch, CURLOPT_URL, $url);`
	`// передача данных с помощью POST`
	`curl_setopt($ch, CURLOPT_POST, true);`
	`// передаваемые POST-параметры`
	`curl_setopt($ch, CURLOPT_POSTFIELDS,`
	`array('name'=>'Petr', 'age'=>'20') );`
	`curl_setopt($ch, CURLOPT_POSTFIELDS,`
	`'Some data in post area' );`
	`// результат выполнения вернуть как результат работы функции`
	`// curl_exec(), а не вывести в html-код (по умолчанию)`
	`curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);`
	С помощью опций можно настроить и другие параметры сеанса: указать загружаемый на сервер файл, частичный обмен (только заголовки) и т.д.

Выполнение сеанса связи	`$content = curl_exec($ch);` Выполняет сеанс связи. При включенной настройке `CURLOPT_RETURNTRANSFER` возвращает содержимое сформированной удаленным сервером страницы в качестве результата своей работы. По умолчанию или при явно выключенной опции сразу отправляет содержимое в html-код.
Закрытие сеанса связи	`curl_close($ch);` Закрывает сеанс межсерверного обмена и очищает память. После этого выполнение сеанса или установка опций для него становятся невозможными.

Область видимости переменных

Если говорить простыми словами, то область видимость переменной – это та часть программного кода, в которой она доступна. По умолчанию в *PHP* все переменные считаются локальными, т.е. если в функции определена переменная `$a`, то ее область видимости будет только эта функция. Переменные с таким же именем в другой функции или же в основном коде программы – это совсем другая переменная и их значения не имеют никакого отношения друг к другу.

Листинг В-5. 9

```
function f1()
{
	$a = 10; // локальная переменная внутри функции
	echo $a; // выводим значение локальной для функции f1() переменной $a
```

```
}
function f2()
{
       echo $a; // пытаемся вывести значение локальной для f2() переменной $a
}

$a=20;                // глобальная переменная – определена вне функции
f1();
echo $a;         // выводим значение глобальной переменной $a
f2();
```

В приведённой в качестве примера программе будет создана глобальная переменная `$a` инициированная значением `20`. Затем вызывается функция `f1()` в которой создается другая, локальная, переменная с таким же именем, инициированная значением `10`, которое и выводится перед выходом из функции. После этого в основной части программы выводится значение глобальной переменной `$a`, т.е. `20`. Обратите внимание, изменение значение переменной `$a` внутри функции не оказывает никакого влияния на переменную `$a` вне ее – это совершенно разные переменные! Программа заканчивается ошибкой выполнения: при вызове функции `f2()` идет обращение к локальной переменной этой функции `$a`, которая еще не была в ней определена. Таким образом, в данном примере присутствуют три разных переменных с одинаковым именем:

- глобальная, с областью видимости в основной части программы;
- локальная для функции `f1()`;
- локальная для функции `f2()`.

Также важно помнить, что переменные внутри отдельных подключенных с помощью инструкций `include` и `require` модулей, имеют ту же область видимости, что и место их подключения. Т.е. переменная `$a` внутри отдельного модуля – эта та же самая переменная `$a`, которая определена перед его вызовом.

Но не всегда ситуация, когда все переменные локальные, удобна для программиста. Если в функции необходимо определять или изменять значение глобальной переменной, т.е. определенной в основной части программы, то это может быть осуществлено двумя способами.

Листинг В-5. 10

```
function f()
{
       $a=40;          // инициализируем локальную переменную $a
       echo $a; // выводим значение локальной переменной $a = 40
       global $a;      // переменная $a – глобальная!
       $a = 10; // присваиваем значение глобальной переменной

      $b = 20;   // присваиваем значение локальной переменной $b
       echo $GLOBALS['b'];       // выводим значение глобальной переменной $b = 30
       echo $b; // выводим значение локальной переменной $b = 20
       $GLOBALS['b'] = 50;       // присваиваем значение глобальной переменной $b
}
$a=20;                // глобальная переменная – определена вне функции
$b=30;                // присваиваем значение глобальной переменной $b
f();             // вызов пользовательской функции
echo $a;         // выводим значение глобальной переменной $a = 10
echo $b;         // выводим значение глобальной переменной $b = 50
```

Первый – использовать в функции перед обращением к переменной инструкцию `global`. После ее использования переменная считается глобальной, т.е. при обращении к ней внутри функции по имени будет возвращаться (или присваиваться) значение не локальной, а одноименной глобальной переменной. Кроме того, значения всех глобальных переменных доступны, в том числе и для изменения, в ассоциативном массиве `$GLOBALS`, элементы которого в качестве ключей имеют имена инициированных в основной части программы переменных.

Еще одной возможностью для управления областью видимости переменной является инструкция `static`. Инициированные внутри функции с инструкцией `static` переменные остаются локальными, но не теряют своего значения после выхода из нее при повторном вызове. Такие статические переменные можно использовать, например, в рекурсивных функциях вместо дополнительных параметров для определения уровней вложенности их вызова.

Листинг В-5. 11

```
function f()
{
	static $a=0;		// инициализация статической переменной
	echo $a;	// вывод значения переменной
	$a++;			// увеличиваем значение переменной на 1
	if( $a<10 ) f();	// рекурсивный вызов
}
f();			// вызов рекурсивной функции
```

При первом вызове функции переменная `$a` инициализируется нулем, но при любом следующем значение переменной `$a` не инициализируется, а принимается равным предыдущему значению. Т.е. второй вызов функции не обнулит переменную, а выведет значение 1, третий – 2 и так далее.

Обработка URL и имен файлов

Разбор URL сайта на составные части	`$parts = parse_url( $url );` Функция возвращает ассоциативный массив с элементами URL. В качестве ключей массива используются: • scheme – используемый протокол; • host – доменное имя или IP-адрес сайта (имя хоста); • port – используемый порт; • user – имя пользователя для серверной авторизации; • pass – пароль для серверной авторизации; • path – полное указанное в URL имя документа на сервере; • query – заданные GET-параметры и их значения; • fragment – часть URL после символа # (якорь). Для URL равного `http://user:pass@site.ru:8080/script/?id=8#news` функция вернет следующие значения: "`http`", "`site.ru`", "`8080`", "`user`", "`pass`", "`/script/`", "`id=8`", "`news`". У функции есть необязательный параметр, задающий возвращаемый результат. Если он указан – результатом работы будет не массив, а строка с соответствующим значением.
Разбор полного имени локального файла	`$parts = pathinfo( $full_file_name );` Функция возвращает ассоциативный массив с элементами полного имени файла. В качестве ключей используются: • dirname – полное имя директории, в которой размещен файл; • basename – имя файла, включая его расширение; • extension – расширение файла; • filename – имя файла без расширения. Для полного имени файла 'htdocs/www/site.ru/index.php' результат работы функции будет следующий: "htdocs/www/site.ru", "index.php", "php", "index". У функции есть необязательный параметр, задающий возвращаемый результат. Если он указан – результатом работы будет не массив, а строка с соответствующим значением.
Получение канонического пути к файлу	`$path = realpath( $path );` Приводит путь (имя директории) в соответствие с требованиями и правилами ОС, в том числе корректно транслируя директории вида "." и "..".

Библиотека стандартных функций *PHP* содержит много других функций по обработке имен файлов и *URL* – подробнее о них можно узнать в документации по языку.

Использование регулярных выражений может значительно облегчить решение некоторых задач, но при этому необходимо проявить особую внимательность: очень часто небрежное составление шаблонов или непонимание принципов их работы приводит к новым, сложно определяемым ошибкам.

Проверить соответствие строки регулярному выражению	`$res  = preg_match( $pattern, $str);` Функция проверяет наличие в строке `$str` наличие соответствующей шаблону `$pattern` подстроки. Если произошла ошибка – возвращает `false`, если соответствий нет – возвращает `0`, если есть – `1`. В качестве дополнительных параметров можно указывать ссылку на массив для получения найденной подстроки – но в любом случае функция прекратит свою работу после первого успешного соответствия шаблону, поэтому в массив будет сохранена только одна подстрока. Не рекомендуется использовать функцию для проверки наличия подстроки в строке: стандартная функция `strops()` работает намного быстрее.
Найти все соответствующие шаблону подстроки	`$res = preg_match_all( $pattern, $str, $array );` Ищет все соответствующие шаблону `$pattern` фрагменты строки `$str` и сохраняет их в массив `$array`. Функция возвращает количество найденных соответствий (`0`, если не найдено) или `false` в случае ошибки. Массив `$array` изменяет свое содержимое после вызова функции, т.к. в нее передается ссылка на него, а не его значение. В его первом элементе (с индексом `0`) будет содержаться массив с соответствующими всему шаблону (маске) найденными подстроками; во втором (с индексом `1`) – массив с соответствующими первой подмаске строками; в третьем – второй подмаске и т.д. Формат массива может быть изменен путем передачи в функцию необязательного параметра.
Заменить все соответствующие шаблону подстроки	$res = preg_replace($pattern, $new, $str); Функция в строке $str ищет соответствующие шаблону $pattern подстроки и заменяет их на строку $new. Имеется возможность формировать новую строку из найденных подмасок соответствующей подстроки – см. документацию PHP. В качестве необязательного параметра передается максимальное количество замен.

Также в *PHP* имеются другие функции для работы с регулярными выражениями: `preg_split()` — разбивает строку по заданным регулярным выражением границам; `preg_replace()` – заменяет все найденные соответствующие шаблону фрагменты на строку; `preg_last_error()` – возвращает код ошибки выполнения последнего регулярного выражения.

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Расскажите о механизме исключений и ее реализации в PHP?
2. Как генерировать исключение в PHP?
3. Для чего используется библиотека cURL?
4. Для чего используется параметр CURLOPT_FOLLOWLOCATION?
5. Какие параметры передаются в функцию file_get_contents?
6. Можно ли в листинге В-5.3 заменить внутренний блок try на проверку существования функции file_get_contents() с помощью function_exists()?
7. Как изменится выполнение программы, если в листинге В-5.4 в шаблоне функции preg_split() изменить "*" на "+"?
8. Что означает символ "" в конце регулярного выражения в листинге В-5.4?
9. Что такое ленивый и жадный кватификатор?
10. Почему в листинге В-5.6 для определения адреса ссылки используется элемент массива $arr с индексом 2, а не 1 или 0?
11. Как изменится работа функции на листинге В-5.7, если убрать из кода конструкцию global?

12. Как изменится работа программы на листинге В-5.8, если в функции parse_url() убрать второй параметр?
13. Какая строка будет сформирована в html-коде после выполнения программы на листинге В-5.10?
14. Как изменится работа функции вы листинге В-5.11, если убрать из нее инструкцию static?
15. На какие часть разбивает URL функция parse_url()?
16. Что является результатом работы функции pathinfo()? Есть ли у нее параметры по умолчанию? И если да – то какие?
17. Что такое регулярное выражение?
18. Какие функции в PHP используют регулярные выражения?
19. Что такое область видимости переменной?
20. Какие типы переменных есть в PHP (с точки зрения видимости)?
21. Каким образом внутри функции изменить значение глобальной переменной?
22. Что такое статические переменные?