**Предисловие**

Для создания современных Web-приложений разработчику требуются знания технологий не только клиентского (Javascript), но и серверного программирования (PHP), а также умение их использования. Кроме того, немаловажным является освоение языка XML, который в настоящее время используется для хранения данных в Web-приложениях и их передачи.

Практикум является руководством для проведения лабора-торных работ по курсу «Программные средства создания Интер-нет-приложений» и предназначен для учащихся специальности 2-40 01 01 «Программное обеспечение информационных техно-логий».

Практикум представлен в двух частях.

В первой части практикума содержались задания по разде­лам «Web-сервер», «Язык гипертекстовой разметки HTML» и «Язык Javascript».

Вторая часть практикума содержит описания лабораторных работ по разделам «Основы технологии XML» и «Программиро-вание на стороне сервера». Последний раздел предполагает изучение широко использующегося в настоящее время языка программирования PHP.

Описание каждой лабораторной работы содержит теорети-ческие сведения, необходимые для ее выполнения, задание и контрольные вопросы. Некоторые практические работы предполагают выполнение заданий по вариантам,а также заданий, индивидуально выдаваемых преподавателем.

Приведенные примеры наглядно отображают структуру XML-документов, сопутствующих им файлов, а также особенности встраивания PHP-кода.

При защите лабораторной работы необходимо продемонст­рировать результат выполнения работы, представить отчет и ответить на контрольные вопросы.

К выполнению следующей работы учащиеся допускаются после сдачи отчета по предыдущей работе.

ЛАБОРАТОРНАЯ РАБОТА 1

**Создание XML-документа**

*Цель работы:* формирование практических умений создания XML-документов.

**Теоретические сведения**

Расширяемый язык разметки XML (eXtensible Markup Lan-guage) предназначен для описания данных с помощью разметки, синтаксис которой определен соответствующей спецификацией.

Синтаксис разметки XML-документа, представляя собой со­вокупность тегов и их атрибутов, внешне напоминает HTML. Однако есть существенное отличие: XML является *языком ме­таразметки*, что означает отсутствие фиксированного набора тегов и атрибутов для описания данных, и позволяет создателям документов определять названия тегов и атрибутов в соответст­вии с предметной областью и содержанием хранимых данных.

Разметка в XML-документе описывает структуру содержа­щихся в нем данных, не указывая при этом, как они должны отображаться. XML-документ может создаваться и редактиро­ваться с помощью любого текстового редактора (например, Блокнота) или специализированного XML-редактора (например, XML Notepad).

Рассмотрим структуру XML-документа, содержащего инфор­мацию об имеющихся в наличии книгах и статьях (пример 1.1).

***Пример 1.1***

*<?xml version="1.0" encoding="Windows-1251"?>*

*<!DOCTYPE documents SYSTEM "documents.dtd">*

*<documents>*

*<books>*

*<book id="Book1">*

*<author>Эрих Мария Ремарк</author>*

*<title>Три товарища</title>*

*<translate> Иванова И.В. </translate>*

*<year\_publication>2000</year\_publication>*

*</book>*

*<book id="Book2">*

*<author>РэйБрэдбери</author>*

*<title>Вино из одуванчиков</title>*

*<translate>Петрова А.К. </translate>*

*<year\_publication>2009</year\_publication>*

*</book>*

*</books>*

*<articles>*

*<article id="Article1">*

*<author>Виктор Кожевников</author>*

*<title>Классификация электронных средств обучения</title>*

*<year\_publication>1998</year\_publication >*

*</article>*

*</articles>*

*</documents>*

Первые две строчки называются прологом XML-документа. Непосредственно данные с разметкой являются телом докумен-та, которое должно быть заключено в так называемый корневой элемент (в примере *<documents>*).

В *объявлении типа документа* (строка *<?xml version="1.0" encoding="Windows-1251"?>*) определяются *версия* XML и *кодировка*, используемая в документе.

В разделе *DOCTYPE* указывается имя корневого элемента и DTD-тип документа, описанный в отдельном файле (как в при­мере) или непосредственно в XML-документе в квадратных скобках [ ].

Содержимое XML-документа представляет собой набор эле-ментов, секций CDATA, директив анализатора, комментариев, специальных символов, текстовых данных.

Единица данных вместе с относящейся к ней разметкой на­зывается *элементом*. Любой непустой элемент должен состоять из начального, конечного тэгов и данных, заключенных между ними. Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные, и по­этому учитываются все символы форматирования (пробелы, пе­реводы строк, табуляции не игнорируются, как в HTML). Если элемент не имеет содержимого, то есть данных, которые он должен определять, он называется пустым. Для определения пустого элемента используется один тег с косой чертой перед закрывающей угловой скобкой (например, **<***empty***/>**).

Если при определении элементов необходимо задать какие-ли-бо параметры, уточняющие его характеристики, то можно ис­пользовать *атрибуты* элемента. В примере 1.1 для указания идентификатора книги или статьи используется атрибут *id.*

Комментарии обозначаются последовательностью символов *<!--* и *-->*.

Чтобы задать область документа, которую при разбореXML-анализатор будет рассматривать как простой текст, игно­рируя любые инструкции и специальные символы, но, в отличие от комментариев, иметь возможность использовать их в прило­жении, необходимо использовать тэги *<![CDATA[* и *]]>*. Внутри этого блока можно помещать, например, инструкции JavaScript.

Для того чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, сим-вол угловой скобки) и не вызвать при этом ошибок в про­цессе разбора, нужно использовать его *специальный символьный либо числовой идентификатор*. Например, *&amp; &lt; &gt*; *&quot*; или*&#6*0;(десятичная форма записи символа<), &#x3c (шестнадцатеричная форма записи) и т. д.

*Инструкции, предназначенные для анализаторов языка*, опи-сываются при помощи тэгов *<?*и *?>*.

В общем случае XML-документы должны удовлетворять сле-дующим требованиям:

- для непустых элементов обязателен закрывающий тег;

- нужно учитывать регистр символов;

- все значения атрибутов должны быть заключены в ка­вычки;

- элементы не должны перекрываться;

- должен быть только один корневой элемент;

- элемент не может иметь несколько атрибутов с одним именем.

**Задание**

СоздайтеXML-документ согласно варианту. Документ дол-жен содержать не менее трех записей.

***Вариант 1***

Информация о книгах в библиотеке:

- идентификационный номер;

- стеллаж:

✓ номер стеллажа;

✓ жанр;

✓ тематика;

- автор или авторы (*наличие необязательно*);

- название;

- под редакцией (*наличие необязательно*);

- общее количество экземпляров;

- количество выданных экземпляров.

***Вариант 2***

Информация о сотрудниках фирмы:

- фамилия;

- имя;

- отчество;

- адрес:

✓ индекс;

✓ тип населенного пункта (город, поселок, деревня);

✓ область;

✓ район;

✓ название населенного пункта;

✓ улица;

✓ дом;

✓ корпус (*наличие необязательно*);

✓ квартира;

- телефон:

✓ домашний;

✓ мобильный (*может быть несколько*);

- должность;

- подразделение.

***Вариант 3***

«Кулинарная книга»:

- тип блюда;

- название блюда;

- мера весов;

- ингредиент 1;

- количество ингредиента 1;

…

- рецепт;

- количество калорий.

***Вариант 4***

Информация об услугах фитнес-центра:

- разновидность услуги;

- стоимость каждой услуги;

- инструктор:

✓ фамилия,

✓ имя,

✓ отчество,

✓ образование (уровень квалификации);

- дни недели и время занятий.

***Вариант 5***

Информация о подписке на периодические издания:

- тип издания (газета или журнал);

- тематика;

- название издания;

- периодичность выхода издания;

- цена;

- подписной индекс.

***Вариант 6***

Информация о расписании междугородних автобусов:

- направление;

- остановки;

- дата отправления;

- время отправления (*автобусы могут отправляться не­сколько раз в день*);

- стоимость билетов;

- количество свободных мест.

***Вариант 7***

Информация о передачах на ТВ:

- передача;

- день:

✓ число;

✓ месяц;

✓ год;

✓ день недели;

- канал;

- жанр;

- время начала;

- продолжительность.

***Вариант 8***

Информация о графике работы врачей медицинского центра:

- специализация врача;

- врач:

✓ фамилия,

✓ имя,

✓ отчество,

- дни приема;

- время приема по дням;

- номер кабинета.

***Вариант 9***

Информация о предложениях туристического агенства:

- страна;

- тип тура (экскурсия, индивидуальный тур, тур выходного дня);

- описание тура;

- горящий тур (*непарный элемент*);

- стоимость;

- количество.

**Контрольные вопросы**

1. Для чего используется язык XML?

2. Чем XML отличается от HTML?

3. Что такое элемент XML?

4. Для чего используется раздел *DOCTYPE*?

5. Как обозначаются комментарии?

6. Перечислите правила построения XML-элементов.

**Отчет по лабораторной работе**

1. Название и цель работы. Листинг программного кода.

2. Файлы выполненного задания на диске.

ЛАБОРАТОРНАЯ РАБОТА 2

**Описание структуры XML-документа**

**средствами DTD**

*Цель работы:* формирование практических умений описания структуры XML-документа средствами DTD.

**Теоретические сведения**

В XML-документах DTD определяет набор действительных элементов, идентифицирует элементы, которые могут находить-ся в других элементах, и определяет действительные атрибуты для каждого из них. В XML использовать DTD не обязательно – документы, созданные без этих правил, будут правильно обрабатываться программой-анализатором, если они удовлетворяют основным требованиям синтаксиса XML. Однако контроль за типами элементов и корректностью отношений между ними в этом случае будет полностью возлагаться на автора документа. До тех пор, пока грамматика нашего нового языка не описана, его сможем использовать только мы, и для этого мы будем вынуждены применять специально разработанное программное обеспечение, а не универсальные программы-анализаторы.

В DTD для XML используются следующие типы правил:

- правила для элементов и их атрибутов;

- описания категорий (макроопределений);

- описание форматов бинарных данных.

Все они описывают основные конструкции языка – элемен-ты, атрибуты, символьные константы, внешние файлы бинарных данных.

Для того чтобы использовать DTD в документе, можно или описать его во внешнем файле и при описании DTD просто ука­зать ссылку на этот файл, или же непосредственно внутри са­мого документа выделить область, в которой определить нуж­ные правила. В первом случае в документе указывается имя файла, содержащего DTD- описания:

*<?xml version=“1.0” standalone=“no” ?>*

*<! DOCTYPE documents SYSTEM «def.dtd»>*

…

Внутри документа DTD-декларации включаются следую­щим образом:

*…*

*<! DOCTYPE documents [*

*<!ELEMENT documents (books+, articles+)>*

*…*

]>

…

Значение *standalone* определяет, может ли этот документ быть обработан без чтения каких-либо других файлов. Напри­мер, если XML-документ не ссылается на другие файлы, вы должны указать *standalone=“yes”.* Если же XML-документ ссы­лается на другие файлы, которые описывают, что документ мо­жет содержать, нужно указать *standalone=“no”*.

Атрибут *SYSTEM* указывает на то, что *def.dtd* находится на локальном сервере.

В том случае, если используются одновременно внутренние и внешние описания, то программой-анализатором будут сна­чала рассматриваться внутренние, то есть их приоритет выше. При проверке документа XML-процессор в первую очередь ищет DTD внутри документа. Если правила внутри документа не определены и не задан атрибут *standalone =“yes”* , то программа загрузит указанный внешний файл и правила, находящиеся в нем, будут считаны оттуда. Если же атрибут *standalone* имеет значение *yes*, то использование внешних DTD-описаний будет запрещено.

Элемент в DTD определяется с помощью дескриптора *!ELEMENT*, в котором указывается название элемента и струк­тура его содержимого.

Например, для элемента *<author>* можно определить сле­дующее правило:

*<!ELEMENT author (#PCDATA)>*

В определении элемента нужно указать сначала название элемента(*author*), а затем модель его содержимого – другие эле­менты или типы данных, которые могут встречаться внутри него. В данном случае содержимое элемента *author* будет опре­деляться при помощи специального маркера #*PCDATA* (что оз­начает *parseable character data*– любая информация, с которой может работать программа-анализатор). Существуют еще две инструкции, определяющие тип содержимого: *EMPTY* и *ANY*. Первая указывает на то, что элемент должен быть пустым (на­пример, <*red*/>), вторая – на то, что содержимое элемента специ­ально не описывается.

Последовательность дочерних для текущего элемента объ­ектов задается в виде списка названий элементов, разделенных запятыми. При этом для того, чтобы указать количество повто­рений включений этих элементов, могут использоваться симво-лы «+», «?».

Например:

*<!ELEMENT book (author+, title, translate?, year\_publica-tion)>*

При этом определении элемент <*book*> должен включать элементы *author, title, translate и year\_publication*, причем эле-менты *title* и *year\_publication* являются обязательными и могут встречаться лишь однажды, элемент *author* может встречаться несколько раз, а элемент *translate* является опциональным, то есть может отсутствовать. В том случае, если существуют несколько возможных вариантов содержимого определяемого элемента, их следует разделять при помощи символа «|».

Например:

*<!ELEMENT flower (#PCDATA | title)\*>*

Символ «\*» указывает на то, что определяемая последова-тельность внутренних элементов может быть повторена несколь-ко раз или же совсем не использоваться.

Если в определении элемента указывается «смешанное» со­держимое(текстовые данные или набор элементов), то необ­ходимо сначала указать *PCDATA*, а затем разделенный символом «|» список элементов.

Списки атрибутов элемента определяются с помощью клю­чевого слова *!ATTLIST*. Внутри него задаются названия атрибу­тов, типы их значений и дополнительные параметры.

Например, для элемента <*name*>

*<name first= “Иван” second= “Петрович” surname= “Си­доров”>*

атрибуты могут быть определены следующим образом:

*<!ATTLIST name first CDATA #REQUIRED second CDATA #IMPLIED surname CDATA #REQUIRED>*

Всего существуют шесть возможных типов значений атри­бута:

*CDATA* – содержимым документа могут быть любые сим­вольные данные;

*NMTOKEN* (*NMTOKENS*) – содержимым элемента может быть только одно отдельное слово (этот параметр является ограни-ченным вариантом *CDATA*);

*ID* – определяет уникальный идентификатор элемента в документе;

*IDREF* (*IDREFS*) – указывает, что значением атрибута долж-но выступать название (или несколько таких названий, разде-ленных пробелами во втором случае) уникального идентифи­катора определенного в этом документе элемента;

*ENTITY* (*ENTITIES*) – значение атрибута должно быть на­званием (или списком названий, если используется *ENTITIES*) компонента (макроопределения), определенного в документе;

*Список допустимых значений* – определяется список зна­чений, которые может иметь данный атрибут (в скобках без ка­вычек).

В определении атрибута также можно использовать сле­дующие параметры:

*#REQUIRED*– определяет обязательный атрибут, который должен быть задан во всех элементах данного типа;

*#IMPLIED*– атрибут не является обязательным;

*#FIXED «значение»*– указывает, что атрибут должен иметь только указанное значение, однако само определение атрибута не является обязательным, но в процессе разбора его значение в любом случае будет передано программе-анализатору;

*Значение*– задает значение атрибута по умолчанию.

Рассмотрим XML-документ с DTD-определением (пример 2.1).

***Пример 2.1***(*books\_dtd.xml*)

*<?xml version=“1.0” encoding=“Windows-1251”?>*

*<!DOCTYPE documents SYSTEM [*

*<!ELEMENT documents (books+, articles+)>*

*<!ELEMENT books (book\*)>*

*<!ELEMENT articles (article\*)>*

*<!ELEMENT book (author+, title, translate?, year\_publi-cation)>*

*<!ELEMENT article (author+, title, year\_publication)>*

*<!ATTLIST book id ID #REQUIRED>*

*<!ATTLIST article id ID #REQUIRED>*

*<!ELEMENT author (#PCDATA)>*

*<!ELEMENT title (#PCDATA)>*

*<!ELEMENT translate (#PCDATA)>*

*<!ELEMENT year\_publication (#PCDATA)>*

*]>*

*<documents>*

*<books>*

*<book id=“Book1” >*

*<author>Эрих Мария Ремарк</author>*

*<title>Три товарища</title>*

*<translate>Иванова И.В. </translate>*

*<year\_publication>2000</year\_publication>*

*</book>*

*<book id=“Book2” >*

*<author>РэйБрэдбери</author>*

*<title>Вино из одуванчиков</title>*

*<translate>Петрова А.К. </translate>*

*<year\_publication>2009</ year\_publication>*

*</book>*

*</books>*

*<articles>*

*<article id=“Article1” >*

*<author>Виктор Кожевников</author>*

*<title>Классификация электронных средств обучения</title>*

*<year\_publication>1998</year\_publication>*

*</article>*

*</articles>*

*</documents>*

Довольно часто при создании XML-элемента разработчику требуется определить, данные какого типа могут использоваться в качестве его содержимого.

Например, некоторый элемент *<last-modified>10.10.98</last-modified>* должен содержать строку, представляющую собой дату, а не число или произвольную последовательность символов.

С помощью типизации данных можно создавать элементы, значения которых могут использоваться, например, в качестве параметров SQL-запросов. Программа-клиент в этом случае должна знать, к какому типу данных относится текущее значе­ние элемента, и в случае соответствия формирует SQL-запрос.

Если в качестве программы на стороне клиента использу­ется верифицирующий XML-процессор, то информацию о типе можно передавать при помощи специально созданного для этого атрибута элемента, имеющего соответствующее DTD-определе­ние. В процессе разбора программа-анализатор передаст значе­ние этого атрибута клиентскому приложению, которое сможет использовать эту информацию должным образом. Например, чтобы указать, что содержимое элемента должно быть длинным целым, можно использовать следующее DTD-определение:

*<!ELEMENT counter (#PCDATA)>*

*<!ATTLIST counter data\_long CDATA #FIXED «LONG»>*

Задав атрибуту значение по умолчанию *LONG* и определив его как *FIXED*, мы, тем самым, позволили программе-клиенту получить необходимую информацию о типе содержимого дан­ного элемента, и теперь она может самостоятельно определить соответствие типа этого содержимого указанному в DTD-опре­делении.

В XML-документе, приведенном в примере 2.2, определя­ются и используются несколько элементов с различными типами данных:

***Пример 2.2***(*def.xml*)

*<?xml version=“1.0” encoding=“Windows-1251”?>*

*<!DOCTYPE houses SYSTEM [*

*<!ELEMENT price (#PCDATA)>*

*<!ATTLIST price data\_currency CDATA #FIXED «CURRENCY»>*

*<!ELEMENT rooms\_num (#PCDATA)>*

*<!ATTLIST rooms\_num data\_byte CDATA #FIXED «BYTE»>*

*<!ELEMENT floor (#PCDATA)>*

*<!ATTLIST floor data\_byte CDATA #FIXED «INTEGER»>*

*<!ELEMENT living\_space (#PCDATA)>*

*<!ATTLIST living\_space data\_float CDATA #FIXED «FLOAT»>*

*<!ELEMENT counter (#PCDATA)>*

*<!ATTLIST counter data\_long CDATA #FIXED «LONG»>*

*<!ELEMENT is\_tel (#PCDATA)>*

*<!ATTLIST is\_tel data\_bool CDATA #FIXED «BOOL»>*

*<!ELEMENT houses (house\*)>*

*<!ELEMENT house (rooms\_num, floor,living\_space, is\_tel, counter, price)>*

*<!ATTLIST house id ID #REQUIRED>*

*]>*

*<houses>*

*…*

*<house id=«0»>*

*<rooms\_num>5</rooms\_num>*

*<floor>2</floor>*

*<living\_space>32.5</living\_space>*

*<is\_tel>true</is\_tel>*

*<counter>18346</counter>*

*<price>34 р. 28 к.</price>*

*</house>*

*…*

*</houses>*

Как видно из примера, механизм создания элементов доку­мента при этом нисколько не изменился. Вся необходимая для проверки типов данных информация заложена в определении элементов внутри блока DTD.

**Задание**

Выполните DTD-описание XML-документа, созданного в ла-бораторной работе 1, которое должно включать: описание эле-ментов; описание атрибутов элементов; определение типа эле­ментов.

**Контрольные вопросы**

1. Для чего используется DTD?

2. Как определить элемент в DTD?

3. Как определить атрибут в DTD?

4. Какие параметры могут использоваться в определении ат-рибута?

5. Как определить тип элемента?

6. Как связать XML-документ c DTD-определением?

**Отчет по лабораторной работе**

1. Название и цель работы. Листинг программного кода.

2. Файлы выполненного задания на диске.

ЛАБОРАТОРНАЯ РАБОТА 3

**Описание схемы XML-документа на языке XSD**

*Цель работы:* формирование практических умений описа­ния структуры XML-документа на языке XSD.

**Теоретические сведения**

Язык XML Schema предоставляет набор встроенных типов дан­ных, которые могут использоваться разработчиками для определения содержимого документа (например:*float*, *string, boolean* и т.д.). Описание возможных стандартных типов приведено в прил. 1.

Кроме предоставления встроенных типов, в XML Schema, как и во многих языках программирования, разработчикам раз­решено определять свои собственные типы, обычно называемые *определенными пользователем типами* (UDT). При описании UDT можно также *определить* для них и пространство имен, чтобы не путать их с другими UDT, случайно использующими такое же имя.

Имена, определенные в схеме, принадлежат так называе­мому целевому пространству имен. Само по себе пространство имен является фиксированным, произвольным именем, которое должно соответствовать синтаксису URL (например,

*http://example.org/publishing*). В то же время, объявление про­странства имен не ссылается на файл с описанием схемы, а лишь указывает его имя.

Определения и объявления в схеме могут ссылаться также на имена, которые принадлежат другим пространствам имен, на­зываемым *исходными*. В каждой схеме может быть определено одно целевое пространство имен и возможно использование множества исходных пространств имен.

Рассмотрим особенности объявления и использования пространств имен на основе упрощенного шаблона XML-схемы (пример 3.1).

***Пример 3.1.***

*<xsd:schema*

*targetNamespace="http://example.org/publishing"*

*xmlns:xsd="http://www.w3.org/2001/XMLSchema"*

*xmlns:tns="http://example.org/publishing">*

*<!--определениетипов-->*

*<xsd:simpleType name="AuthorType">*

*<!--определение области значений типа-->*

*...*

*</xsd:simpleType>*

*<!--глобальное объявление элементов и их атрибутов-->*

*<xsd:element name="author" type="tns:AuthorType"/>*

*<xsd:element name="year" type="xsd: gYear"/>*

*...*

<*/xsd:schema*>

Элемент ***xsd:schema*** определяет содержимое пространства имен, а атрибут ***targetNamespace*** – его имя. Таким образом, целе­вым пространством имен является *http://example.org/publishing*, оно содержит имена *AuthorType, year* и *author*. Имена *schema, element, simpleType* и *gYear* принадлежат стандартному исходно-му пространству имен

*http://www.w3.org/2001/XMLSchema*.

Имена пространства имен сокращаются при помощи конст­рукции с зарезервированным словом *xmlns*. Так, в примере 3.1 целевому пространству имен назначается псевдоним *tns*, а исходному – *xsd*. Затем в документе указание конкретного имени предваряется псевдонимом пространства имен, к которому оно принадлежит (например, *tns:AuthorType*).

*Определение элемента* заключается в определении его име-ни и модели (типа) контента.

Шаблон определения элемента выглядит следующим образом:

*<xsd:element name="…" type="…" [minOccurs="…"] [maxOccurs="…"] [default="…"| fixed="…"]>*

Обязательными являются только параметры *name* и *type*, определяющие соответственно имя и тип элемента. Параметры *minOccurs* и *maxOccurs* определяют минимальное или макси­мальное количество вхождений элемента. Например, если эле­мент является обязательным, значение *minOccurs* больше или равно 1. Параметр *maxOccurs* может кроме числового значения принимать также значение *unbounded*, что указывает на отсутст-вие ограничения максимального числа появлений. Значение по умолчанию для *minOccurs* и для *maxOccurs* равно 1.

*Атрибут элемента* определяется по следующему шаблону:

*<xsd:attribute name="…" type="…" [use="…"] [default="…"| fixed="…"]>*

Атрибуты, в отличие от элементов, могут появиться только однажды или ни разу. Поэтому и синтаксис для определения по­явления атрибутов отличается от синтаксиса для определения числа появлений элементов. В частности, атрибуты могут быть объявлены с параметром *use*. В зависимости от значения этого параметра атрибут обязателен (*use="required"*), необязателен (*use="optional"*) или запрещен (*use="prohibited"*).

Значения по умолчанию и атрибутов, и элементов могут быть объявлены с использованием параметра *default*, хотя этот параметр в том или ином случае работает по-разному. Атрибут со значением, определенным по умолчанию, может появляться или не появляться в документе. Если атрибут не появляется в документе, то обработчик схемы обеспечивает атрибут со значе­нием, равным значению *default*. Таким образом, значения по умолчанию для атрибутов имеют смысл, только если сами атри­буты являются необязательными, поэтому будет ошибкой опре­делить значение по умолчанию вместе с параметром *use,* отлич­ным от *use="optional"*.

Значение по умолчанию для элементов обрабатывается не­много по-другому. Если элемент появляется в документе, но не содержит какого-либо значения, то в качестве его значения под­ставляется значение по умолчанию. Однако, если элемент не по­является в документе, то обработчик схемы не обеспечивает его значения вообще. В общем, различия между значениями по умолчанию элемента и атрибута в следующем: заданное по умол-чанию значение атрибута применяется тогда, когда атрибут отсутствует, а заданное по умолчанию значение элемента при­меняются тогда, когда элемент присутствует в документе, но не имеет значения (пуст).

Атрибут *fixed* используется в объявлениях и атрибутов, и элементов. Он используется, чтобы указать, что атрибут или элемент принимают фиксированные значения (независимо от их наличия в документе). Обратите внимание, что понятия «фикси­рованное значение» и «значение по умолчанию» являются взаимоисключающими, поэтому объявление не может одновременно содержать атрибуты *fixed* и *default*.

Тип элемента может быть *простым* или *комплексным* (*сложным*). Элемент простого типа не может содержать другие элементы или атрибуты. Комплексный тип может использоваться для встраивания элементов в другие элементы или ассоциирования атрибутов с элементом.

Предопределенный простой тип ограничивает значения по их базовому типу и создается с помощью элемента *simpleType*. Для ограничения диапазона значений используется вложенный элемент *restriction*. Он позволяет определить, как именно нужно ограничивать базовый тип, сужая один или более из его аспек-тов (facets).

Например, чтобы определить ограничения для элемента, со­держащего год издания книги или журнала, может использовать-ся следующий код:

*<xsd:simpleType name="YearType">*

*<xsd:restriction base="xsd:gYear">*

*<xsd:minInclusive value="1900"/>*

*<xsd:maxInclusive value="2012"/>*

*</xsd:restriction>*

*</xsd:simpleType>*

Таким образом, допустимые значения года издания нахо­дятся в интервале от 1900 до 2012.

Возможные аспекты базового типа XSD представлены в прил. 2.

Большинство аспектов неприменимы ко всем типам (некоторые имеют смысл только в определенных типах). Элемент *pattern* применяется, если ограничением выступает регулярное выражение.

Например, чтобы создать ограничение для элемента, содер­жащего информацию об авторе, нужно определить следующий простой тип:

*<xsd:simpleType name="AuthorType">*

*<restriction base=" xsd:string">*

*<pattern value="^[а-яА-Я\. ]+ "/>*

*</restriction>*

*</xsd:simpleType>*

Базовым является тип *string*(строка), ограничение по базовому типу создается с использованием регулярных выражений. В примере информация об авторе может содержать один или более символов, включающих строчные или прописные буквы рус-ского алфавита, точку, а также пробел. Символ "^" указывает на начало строки.

Развернутое описание синтаксиса регулярных выражений приведено в прил. 3.

*Комплексный* (*составной*) *тип* определяется для элементов, содержащих вложенные элементы, и элементов с атрибутами.

Чтобы определить новый комплексный тип в целевом про­странстве имен схемы, используется элемент ***xsd:complexType*:**

*<xsd:complexType name="…">*

*<!--описание модели содержимого комплексного типа -->*

*</xsd:complexType>*

Описание модели содержимого комплексного типа обычно включает объявления элементов, ссылки на элементы (<*xsd:ele-ment ref="имя элемента, объявленного ранее"*>) и объявления атрибутов.

При описании модели содержимого также используются конструкции ***xsd:sequence****,* ***xsd:choice*** и ***xsd:all*, называемые со­ставителями. Контейнер** *xsd:sequence* определяет порядок указа­ния дочерних элементов, контейнер *xsd:choice* указывает на необходимость выбора только одного дочернего элемента из указанных, а контейнер *xsd:all* предписывает, что дочерние элементы могут размещаться в произвольном порядке. Описания атрибутов помещаются не в составителе, а после него в конце определения составного типа.

В примере 3.2 приведена возможная XSL-схема для доку­мента из примера 1.1.

***Пример 3.2* (***sch.xsd***)**

*<xsd:schema targetNamespace="http://example.org/publishing"*

*xmlns:xsd="http://www.w3.org/2001/XMLSchema"*

*xmlns:tns="http://example.org/publishing">*

*<xsd:element name="documents" type="tns: DocsType"/>*

*<xsd:element name="author" type="tns:AuthorType"/>*

*<xsd:element name="title" type="xsd:string"/>*

*<xsd:element name="year\_publication" type="tns:YearType "/>*

*<xsd:complexType name="DocsType">*

*<xsd:sequence>*

*<xsd:element name="books"/>*

*<xsd:complexType>*

*<xsd:element name="book" type="tns: BookType">*

*</xsd:complexType>*

*<xsd:element name="articles"/>*

*<xsd:complexType>*

*<xsd:element name="article" type="tns: ArticleType"/>*

*</xsd:complexType>*

*</xsd: sequence>*

*</xsd:complexType>*

*<xsd:complexType name="BookType">*

*<xsd:sequence>*

*<xsd:element ref="author"/>*

*<xsd:element ref="title"/>*

*<xsd:element name="translate" type="tns:AuthorType"/>*

*<xsd:element ref=" year\_publication"/>*

*</xsd: sequence>*

*<xsd:attribute name="id" type="xsd:string" use="required"/>*

*</xsd:complexType>*

*<xsd:complexType name="ArticleType">*

*<xsd:sequence>*

*<xsd:element ref="author"/>*

*<xsd:element ref="title"/>*

*<xsd:element ref=" year\_publication"/>*

*</xsd:sequence>*

*<xsd:attribute name="id" type="xsd:string" use="required"/>*

*</xsd:complexType>*

*<xsd:simpleType name="AuthorType" >*

*<xsd:restriction base="xsd:string">*

*< xsd:pattern value="^[а-яА-Я\. ]+ "/>*

*</ xsd:restriction>*

*</xsd:simpleType>*

*<xsd:simpleType name="YearType">*

*<xsd:restriction base="xsd:gYear">*

*<xsd:minInclusive value="1900"/>*

*<xsd:maxInclusive value="2012"/>*

*</xsd:restriction>*

*</xsd:simpleType>*

*</xsd:schema >*

В начале документа определяется корневой элемент и эле­менты, на которые далее будет выполняться ссылка. Затем по порядку определяются комплексные и простые типы элементов.

Если элементы документа не принадлежат никакому про­странству имен и записаны без префикса, то для связи доку­мен-та XML со схемой XSD в корневом элементе документа за­писывается атрибут *noNamespaceSchemaLocation*:

*<documents xmlns:xsi="http://www.w3.org/2001/XMLSchema"*

*xsi:noNamespaceSchemaLocation="sch.xsd">*

Если же элементы документа относятся к некоторому про­странству имен, то применяется атрибут *schemaLocation*, в кото­ром через пробел перечисляются пространство имен и располо­жение файла со схемой, описывающей это пространство имен:

*<documents xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"*

*xsi: schemaLocation=*

*"http://some.firm.com /someNames A.xsd*

*http://some.firm.com /anotherNames B.xsd”*

*xmlns:pr1= http://some.firm.com/someNames*

*xmlns:pr2= “http://some.firm.com /anotherNames">*

Значением атрибута *xsi:schemaLocation* является список пар имен пространства имен и размещения URI, разделенных про­белами, который показывает список, где искать определенный файл схемы.

Таким образом, в документе можно использовать имена, определенные в схемах *A.xsd* и *B.xsd*, снабжая их префиксами *pr1* и *pr2* соответственно.

**Задание**

Разработайте XSD-схему XML-документа, созданного в лабораторной работе 1, которая должна включать:

- описание элементов;

- описание атрибутов элементов;

- определение типа элементов;

- определение последовательности использования эле­ментов.

**Контрольные вопросы**

1. В чем отличие XSD от DTD?

2. Как определить эле­мент в XSD?

3. Как определить атри­бут элемента в XSD?

4. Как подключить XSL-схему к XML-документу?

5. В каких случаях исполь­зуется сложный (комплексный) тип данных?

**Отчет по лабораторной работе**

1. Название и цель работы. Листинг программного кода.

2. Файлы выполненного задания на диске

ЛАБОРАТОРНАЯ РАБОТА 4

**Форматирование вывода XML-документа**

*Цель работы:* формирование практических умений исполь­зования XSL для форматирования вывода XML-документа.

**Теоретические сведения**

Расширяемый язык таблиц стилей (XSL) основан на XML и предназначен для трансформации документа XML в другой до­кумент XML или отображения в виде HTML.

Язык XSL основан на каскадных таблицах стилей CSS (Сas-cading Style Sheets) и на языке DSSSL (Document Style Semantics and Specification Language – язык спецификации и семантики стиля документа). По мере развития язык XSL становится все ближе к CSS и дальше от DSSSL.

Документ XSL (файл с расширением *.xsl*) фактически пред­ставляет собой документ XML, который может содержать теги HTML.

XSL-таблица стилей является мощным и гибким инстру­ментом для отображения XML-документов. Используя XSL-таб­лицы стилей, можно не только задать формат для каждого эле­мента XML, как при использовании CSS, но и обеспечить сред­ства контроля над выводимыми данными: выбрать для отобра-жения конкретные данные XML, представить эти данные в любом порядке, модифицировать или добавлять информацию.

Вместе с тем, технология XSLT является более сложной для понимания, чем CSS. Кроме того, это новая технология, следова-тельно, имеет меньшую степень поддержки, то есть меньшую степень унификации среди современных браузеров.

Документ XSL может быть создан в любом текстовом ре­дакторе.

Корневым элементом таблицы стилей является элемент *<xsl: stylesheet>*:

*<xsl: stylesheet version="1.0"*

*xmlns:xsl="http://www.w3.org/1999/XSL/Transform">*

Первым атрибутом этого элемента является *version*, опреде­ляющий версию XSL, вторым – атрибут *xmlns:xsl*, определяю­щий пространство имен.

Элемент *xsl:stylesheet* должен содержать один или несколько шаблонов элементов, создаваемых с использованием элемента *<xsl:template>*.

*<xsl:template match="/">*

*<!--дочерниеэлементы... -->*

*</xsl:template>*

Браузер использует шаблон для отображения определенной ветви элементов в иерархии XML-документа, с которым связана таблица стилей. Атрибут *match* шаблона указывает на опреде­ленную ветвь (он аналогичен селектору в правиле CSS). Значе­ние атрибута *match* носит название образца (pattern). Образец в данном примере ("/") представляет корневой элемент всего XML-документа. Таким образом, этот шаблон содержит инст­рукции для отображения всего XML-документа.

Для того чтобы связать таблицу стилей XSL с XML-доку­ментом, используется инструкция *xml-stylesheet*, которая имеет следующую обобщенную форму записи:

*<?xml-stylesheet type="text/xsl" href="путь к файлу XSL"?>*

В качестве значения атрибута *href* может использоваться абсолютный (URL полностью) или относительный путь.

Пример указания абсолютного пути:

*<?xml-stytesheet type="text/xsl"*

*href="http:/www.my\_domain.com/lnventory.xsl"?>*

Пример указания относительного пути (при условии, что *lnventory.xsl* находится в той же папке, что и текущий XML-до­кумент):

*<?xml-stylesheet type="text/xsl" href="lnventory.xsl"?>*

Обычно инструкция *xml-stylesheet* добавляется в пролог XML-документа вслед за объявлением XML (например, *<?xml version="1.0" encoding="Windows-1251"?>*).

Если XML-документ связан с таблицей стилей XSL, то при его открытии в браузере отобразится содержимое, оформленное в соответствии с инструкциями по преобразованию, содержащи-мися в таблице стилей. В отличие от CSS, если XML-документ связан с несколькими таблицами стилей XSL, браузер исполь-зует первую таблицу и игнорирует все остальные. Если вы свяжете с XML-документом и CSS-таблицу, и XSL-таблицу стилей, браузер использует только XSL-таблицу стилей.

Если XML-документ не связан ни с CSS-таблицей, ни с XSL-таблицей стилей, браузер отобразит документ с помощью встроенной XSL-таблицы, которая используется по умолчанию. Эта таблица стилей отображает исходный XML-текст в виде де­рева с возможностью свертывания/развертывания уровней.

**Основные дескрипторы XSL**

**для преобразования XML- документа в HTML**

Шаблон ***<****xsl:template****>*** содержит два вида элементов:

- элементы, представляющие собой HTML-разметку. На­пример:

*<H2>Заголовок</H2>*

- XSL-элементы.Например:

*<xsl:value-of select="/books/book/author"/>*

Браузер отличает XML-элемент от элемента, представляю­щего HTML, поскольку первый имеет в качестве префикса опи­сание пространства имен *xsl*:. XSL-элементы в шаблоне не копи­руются на выход HTML. Они лишь содержат инструкции по вы­бору и модификации данных XML либо используются для вы­полнения других задач.

XSL-элемент *value-of* добавляет текстовое содержимое оп­ределенного XML-элемента и его дочерних элементов в выход­ной модуль HTML. Конкретный элемент задается атрибутом *select* XSL-элемента *value-of*.

Адресация к XML-данным строится на выражениях языка [xPATH](http://ru.wikipedia.org/wiki/XPath). Самые простые выражения xPATH похожи на пути в файловых системах. Путь в файловой системе состоит из цепо­чек директорий, разделенных слэшем. xPATH-пути выглядят аналогичным образом:

*"/"*– путь к корневому узлу;

*" /books"*– путь к элементу *books*;

*"/books/book"*– путь к элементу *book*, содержащему инфор­мацию о книге;

*"/books/book/author"*– путь к элементу *author*, содержащему фамилию и имя автора книги.

Так же как и в файловых системах, в [xPATH](http://ru.wikipedia.org/wiki/XPath) есть понятия «относительного пути» и «абсолютного пути». Абсолютный путь отсчи­тывается от корневого узла и начинается с одинарного слэша  "/" как в примерах выше. Относительный путь отсчиты-вается от текущего положения, определяемого значением атрии-бута *match* элемента <*xsl:template*>.

Для того чтобы добавить в выходной модуль HTML значе­ние атрибута, используется символ *@*. Например, для вывода значения атрибута *id* тега <*book id="Book1*"> может исполь-зоваться следующая запись:

*<xsl:value-of select="/books/book/author/@id"/>*

Чтобы вывести содержимое элемента, включающее HTML-те-ги, используется конструкция  *<xsl:copy-of select="название эле-мента****/****\*"****/>***. Значение атрибута *select* предполагает выбор всех дочерних узлов (элементов, атрибутов, текста, комментариев) конкретного элемента.

В XSL предусмотрены два условных оператора: *xsl:if* и *xsl:choose*.

Рассмотрим использование оператора *xsl:if*. Например, что-бы проверить, издана ли книга позднее 1990 года из XML-до-кумента (пример 1.1 в описании лабораторной работы 1) может использоваться конструкция

*<xsl:if test="year\_publication > 1990"> Действия при вы­полнении условия</xsl:if>*

Если нужно проверить обратное, то есть, что книга издана ранее 1990 года, нужно использовать не символ <, а его *amp*-по-следовательность (*&lt;*):

*<xsl:if test="year\_publication &lt; 1990"> Действия при выполнении условия</xsl:if>*

Так как XML-документ чаще всего содержит не одну запись, а набор записей одинаковой структуры, требуется использова-ние специального оператора для их перебора. Таким оператором является *xsl:for-each*.

Рассмотрим пример вывода XML-документа из примера 1.1 в виде таблицы на HTML-страницу.

**Пример 4.1(***table.xsl***)**

*<?xml version="1.0" encoding="WINDOWS-1251" ?>*

*<xsl:stylesheet*

*xmlns:xsl="http://www.w3.org/1999/XSL/Transform">*

*<xsl:template match="/">*

*<html>*

*<head>*

*</head>*

*<body>*

*<table border="1">*

*<tr bgcolor="#CCCCCC">*

*<td align="center"><strong>Автор</strong></td>*

*<td align="center"><strong>Название</strong></td>*

*<td align="center"><strong>Дата</strong></td>*

*</tr>*

*<tr>*

*<td align="center" colspan="3">*

*<strong>Книги</strong></td>*

*</tr>*

*<xsl:for-each select="documents/books/book">*

*<tr bgcolor="#F5F5F5">*

*<td><xsl:value-of select="author"/></td>*

*<td><xsl:value-of select="title"/></td>*

*<td><xsl:value-of select="translate"/></td>*

*<td><xsl:value-of select="year\_publication"/></td>*

*</tr>*

*</xsl:for-each>*

*<tr>*

*<td align="center" colspan="3">*

*<strong>Журналы</strong></td>*

*</tr>*

*<xsl:for-each select="documents/articles/article">*

*<tr bgcolor="#F5F5F5">*

*<td><xsl:value-of select="author"/></td>*

*<td><xsl:value-of select="title"/></td>*

*<td><xsl:value-of select="year\_publication"/></td>*

*</tr>*

*</xsl:for-each>*

*</table>*

*</body>*

*</html>*

*</xsl:template>*

*</xsl:stylesheet>*

Здесь оператор *xsl:for-each* организует перебор всех элементов *book* и *article*. Далее все, что находится внутри этих элементов, адресуется относительно текущего элемента, заданного атрибутом *select*инструкции *xsl:for-each*.

Конструкция *xsl:sort* служит для сортировки XML-элемен-тов*.* Например, чтобы сортировать книги по дате рождения ав-тора, нужно модифицировать пример 4.1 следующим образом:

***Пример 4.2***(*table\_sort.xsl*).

*<?xml version="1.0" encoding="WINDOWS-1251" ?>*

*<xsl:stylesheet*

*xmlns:xsl=" http://www.w3.org/1999/XSL/Transform ">*

*<xsl:template match="/">*

*<html>*

*<head>*

*</head>*

*<body>*

*<table border="1">*

*<tr bgcolor="#CCCCCC">*

*<td align="center"><strong>Автор</strong></td>*

*<td align="center"><strong>Название</strong></td>*

*<td align="center"><strong>Дата</strong></td>*

*</tr>*

*<tr>*

*<td align="center" colspan="3">*

*<strong>Книги</strong></td>*

*</tr>*

*<xsl:for-each select="documents/books/book">*

*<xsl:sort select=" year\_publication"/>*

*<tr bgcolor="#F5F5F5">*

*<td><xsl:value-of select="author"/></td>*

*<td><xsl:value-of select="title"/></td>*

*<td><xsl:value-of select="translate"/></td>*

*<td><xsl:value-of select="year\_publication"/></td>*

*</tr>*

*</xsl:for-each>*

*<tr>*

*<td align="center" colspan="3">*

*<strong>Журналы</strong></td>*

*</tr>*

*<xsl:for-each select="documents/articles/article">*

*<tr bgcolor="#F5F5F5">*

*<td><xsl:value-of select="author"/></td>*

*<td><xsl:value-of select="title"/></td>*

*<td><xsl:value-of select="year\_publication"/></td>*

*</tr>*

*</xsl:for-each>*

*</table>*

*<html>*

*<head>*

*</head>*

*<body>*

*</xsl:template>*

*</xsl:stylesheet>*

Опция *select э*лемента *xsl:sort* определяет, по какому ключу идет сортировка(в примере – по году издания книги).

Элемент *xsl:sort* имеет также дополнительные параметры:

- порядок сортировки (*убывание/возрастание в зависимости от регистра букв, атрибут case-order= "lower-first|upper-first"*);

- тип данных (числовой/текстовый, атрибут *data-type="num-ber|text"*).

**Задание**

Выведите на HTML-страницу содержимое XML-документа, созданного в лабораторной работе 1, используя XSL. Выполните сортировку записей по одному из возможных критериев и фильтрацию данных по одному из возможных показателей.

**Контрольные вопросы**

1. Для чего используется язык XSL?

2. Как связать таблицу стилей XSL с XML-документом?

3. Как создаются шаблоны элементов в XSL?

4. Для чего используется XSL-элемент *value-of*?

5. Как осуществляется адресация к XML-данным?

6. Какие условные операторы предусмотрены в XSL? В чем особенности их использования?

7. Для чего используется оператор *for-each*?

8. Как выполняется сортировка XML-данных?

**Отчет по лабораторной работе**

1.Название и цель работы. Листинг программного кода.

2. Файл *lr4.хsl*, содержащий выполненное задание.

ЛАБОРАТОРНАЯ РАБОТА 5

**Создание простейших PHP-сценариев**

*Цель работы:* формирование практических умений создания и запуска простейших PHP-сценариев.

**Теоретические сведения**

PHP – это язык серверных скриптов (server scripting langua-ge), встраиваемый в HTML. Он интерпретируется и выполняется на сервере.

Для встраивания PHP-кода в HTML используются специальные открывающий и закрывающий теги (<*?php* и *?*>). Для PHP тег <*?php* означает начало блока команд, которые надо обработать и выполнить. Заканчивается блок тегом *?*>. Иными словами, символы <*?php* и *?*> выполняют роль «скобок». Все, что находится вне их, интерпретатор PHP пропускает и отправ-ляет клиенту без всякой обработки. PHP-код можно вставлять в любое место HTML-страницы. Вместо «скобок» <*?php* … *?*> можно использовать и сокращенную запись <*?* … *?*>.

Рассмотрим встраивание кода PHP в HTML-страницу. Так выглядит web-страница с элементами PHP:

<html>  
<head>  
<title>Пример</title>  
</head>  
<body>  
<?php echo "Привет, я PHP-программа!"; ?>  
</body>  
</html>

Оператор *echo* осуществляет вывод информации. Для этого могут использоваться также функции *print* и *printf.*

Для разделения операторов PHP используется точка с запя­той (;).

Прежде чем запустить программу, ее необходимо правильно сохранить на сервере. Для этого нужно сохранить сценарий в виде имя файла.php, например, first.php и скопировать его в ка-талог, определенный в директиве DocumentRoot вашего сервера (обычно это каталог www).

Для выполнения скрипта в строке браузера нужно ввести http://localhost/first.php.

В результате загрузки получим страницу, представленную на рис 5.1.

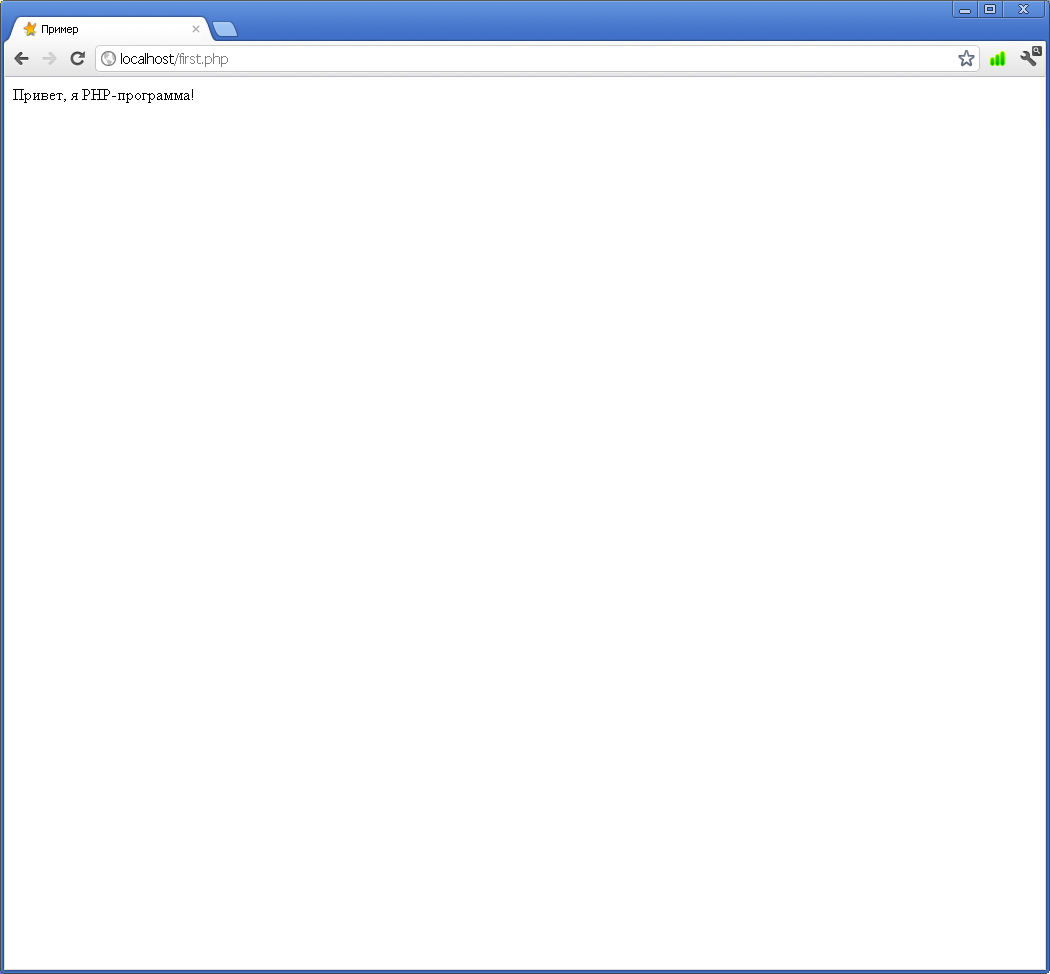


Рис. 5.1. Результат запроса *http://localhost/first.php*

Переменные в PHP обозначаются знаком *$* с последующим именем переменной. Имя переменной чувствительно к регистру символов. Имена переменных следуют тем же правилам, что и другие метки в PHP. Правильное имя переменной начинается с буквы или символа подчеркивания с последующими (в любом количестве) буквами, числами или символами подчеркивания.

Например:

$var = "Bob";   
$Var = "Joe";   
echo "$var, $Var";   
// выводит "Bob, Joe"   
$4site = 'str'; // неправильно; начинается с числа   
$\_4site = 'notyet'; // правильно; начинается с символа подчеркивания

Язык PHP поддерживает следующие типы данных:

- Integer,

- Double,

- String,

- Array,

- Object,

- Boolean.

В PHP тип переменной отдельно определять не нужно, он определяется присвоенным ей значением.

PHP поддерживает следующие виды комментариев:

<?php

echo "This is a test"; // Это однострочный комментарий в стиле С++

/\* Это многострочный комментарий,

это еще одна его строка \*/

echo "This is yet another test";

echo "One Final Test"; # Это комментарий в shell-стиле

?>

Оператором условия в PHP является оператор *if*.

Общий формат:

if (условие) {

набор команд

}

Оператором цикла в PHP является оператор *while*.

Общий формат:

while (условие) {

набор команд

}

При использовании языка PHP программисту не требуется специальным образом извлекать данные из формы. На момент начала выполнения PHP-скрипта уже существуют и определены переменные, соответствующие одноименным полям формы.

Например, если форма имеет вид:

<form action="test.php">  
<input type="text" name="f1">  
<input type="text" name="f2">  
<input type="submit" name="do" value="OK">  
</form>

то при старте скрипта *test.php* в нем уже будут определены переменные с содержимым полей *do*, *f1* и*f2*.

К ним можно получить доступ, используя так называемые суперглобальные массивы *$\_GET[ ]* или *$\_POST[ ]* (в зависимо­сти от метода передачи данных), а также обращаясь к ним, как к обычным глобальным переменным.

Например, в файле *form.html* определена простейшая форма регистрации пользователя:

<html>  
<head>  
</head>  
<body>  
<form action="second.php" method="post">  
<b>Give us some information!</b><br>  
Your Name:<br>  
<input type="text" name="name" size="20" maxlength="20" value=""><br>  
Your Email:<br>  
<input type="text" name="email" size="20" maxlength="40" value=""><br>  
<input type="submit" value="go">  
</form>  
</body></html>

Внешний вид страницы *form.html* представлен на рис. 5.2.

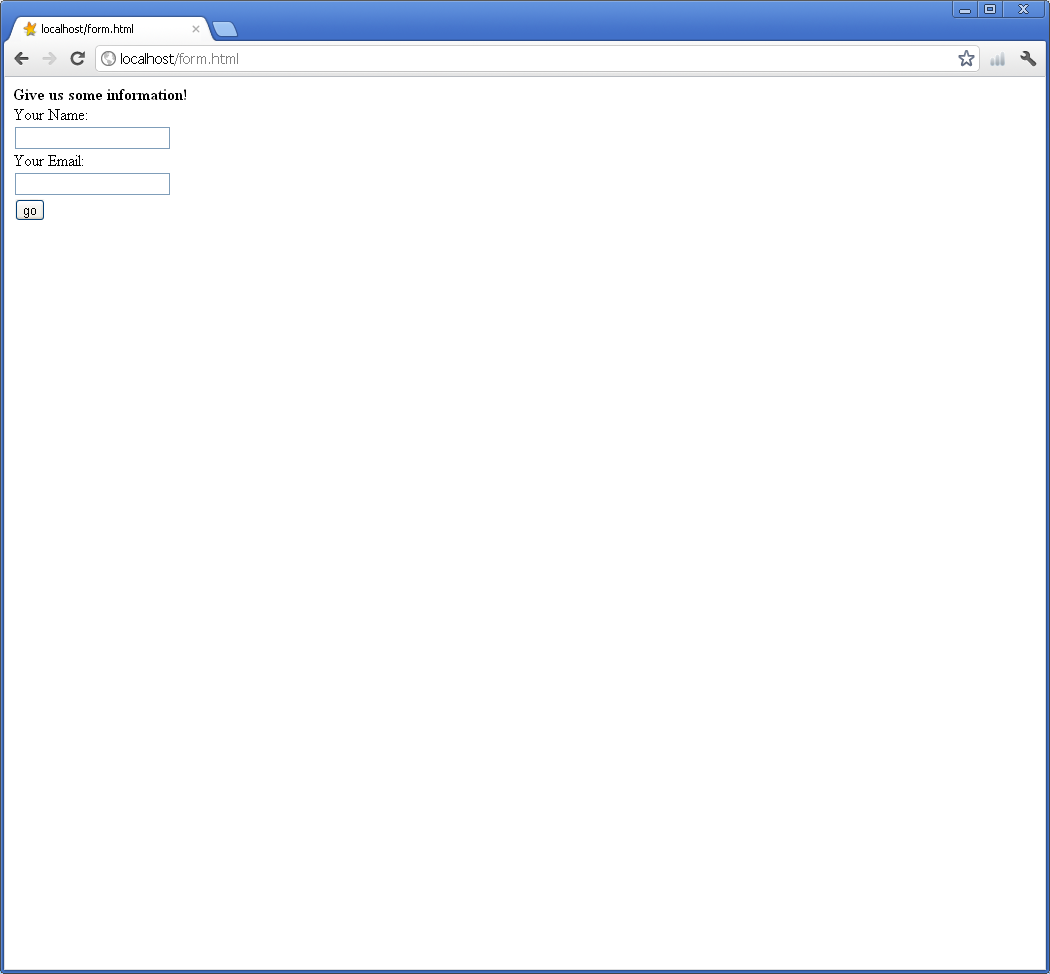


Рис. 5.2. Страница *form.html*

Когда пользователь щелкает на кнопке отправки данных (кнопка *go*), форма обращается к странице *second.php*,  код кото­рой представлен ниже. В свою очередь, *second.php* выводит пе­ременные *$name* и *$email*, переданные с запросом.

<html><head>  
</head>  
<?php  
// Вывести имя и адрес электронной почты.  
print "Hi, $name! Youremailaddressis $email";  
?>  
</body></html>

Получение значений переменных могло быть выполнено и с использованием массива *$\_POST*[ ]:

$n=$\_POST['name'];

$e=$\_POST['email'];

print "Hi, $n! Your email address is $e";

**Задание**

1. Используя PHP, создайте на HTML-странице простейший калькулятор, выполняющий следующие операции:

- суммирование,

- вычитание,

- умножение,

- деление.

Для каждого из операндов и результата должно быть опре­делено свое поле ввода. Калькулятор следует оформить с ис­пользованием таблицы. Файл сохраните под именем *lr5.php*.

2. Реализуйте проверку корректности введенных пользова­телем данных.

3. Просмотрите результат выполнения *lr5.php* в браузере.

4. Просмотрите исходный *html*-код страницы в браузере.

**Контрольные вопросы**

1. В чем отличие серверных скриптов от клиентских?

2. Как осуществляется встраивание PHP-кода в HTML-страницу?

3. Как выполнить PHP-код?

4. Каков будет исходный код страницы, загруженной в резуль-тате запроса *http://localhost/first.php*?

6. Как обозначаются переменные в PHP?

7. Как обозначаются комментарии в PHP?

8. Как получить значение переменных, переданных в сценарий

РНР?

**Отчет по лабораторной работе**

1. Название и цель работы. Листинг *lr5.php*.

2. Файлы *lr5.php*на диске.

ЛАБОРАТОРНАЯ РАБОТА 6

**Создание и использование функций**

*Цель работы:* формирование практических умений создания и использования функций в PHP.

**Теоретические сведения**

Функции могут создаваться в любом месте программы РНР, однако размещать все функции, используемые сценарием, реко-мендуется в самом начале файла.

Обобщенный синтаксис функций РНР выглядит так:

*function имя\_функции ([$параметр1, $параметр2, .... $параметрn]) {*

*тело функции*

*}*

В зависимости от области видимости переменные РНР делятся на четыре типа:

- локальные переменные;

- параметры функций;

- глобальные переменные;

- статические переменные.

*Локальной* называется переменная, объявленная внутри функ-ции, и на нее можно ссылаться только в этой функции.

*Параметры функций* объявляются в круглых скобках после имени функции. К параметрам можно обращаться в той функ-ции, в которой они были объявлены, и выполнять с ними необходимые операции, но после завершения функции параметры уничтожаются.

*Глобальные переменные*, в отличие от локальных, доступны в любой точке программы. Но, чтобы изменить значение гло-бальной переменной, необходимо специально объявить ее как глобальную в соответствующей функции. Для этого перед именем переменной ставится ключевое слово *GLOBAL*.

Например:

*$somevar = 15;*

*function addit() {*

*GLOBAL $somevar;*

*$somevar++;*

*print "Somevar is $somevar";*

*}*

*addit();*

Будет выведено значение *$somevar*, равное 16.

Альтернативный способ объявления глобальных перемен­ных связан с использованием массива РНР *$GLOBALS*. Вернем-ся к предыдущему примеру и воспользуемся этим массивом для объявления глобальной переменной *$somevar*:

*$somevar = 15;*

*function addit() {*

*$GLOBALS["somevar"];*

*$somevar++;*

*}*

*addit();*

*print "Somevar is $somevar";*

*Статическая переменная* сохраняет свое значение при по­вторном вызове функции. Для объявления статической перемен­ной перед ее именем ставится ключевое слово *STATIC*:

*STATIC $somevar;*

Рассмотрим пример 6.1.

***Пример 6.1.***

*function keep\_track() {*

*STATIC $count = 0;*

*$count++;*

*print $count;*

*print "<br>";*

*}*

*keep\_track();*

*keep\_track();*

*keep\_track();*

Если бы переменная *$count* не была объявлена статической (то есть являлась локальной), результат выглядел бы так:

*1*

*1*

*1*

Но поскольку переменная *$count* является статической, при каждом вызове функции будет сохраняться ее предыдущее зна­чение. Результат выполнения функции приведен на рис. 6.1.

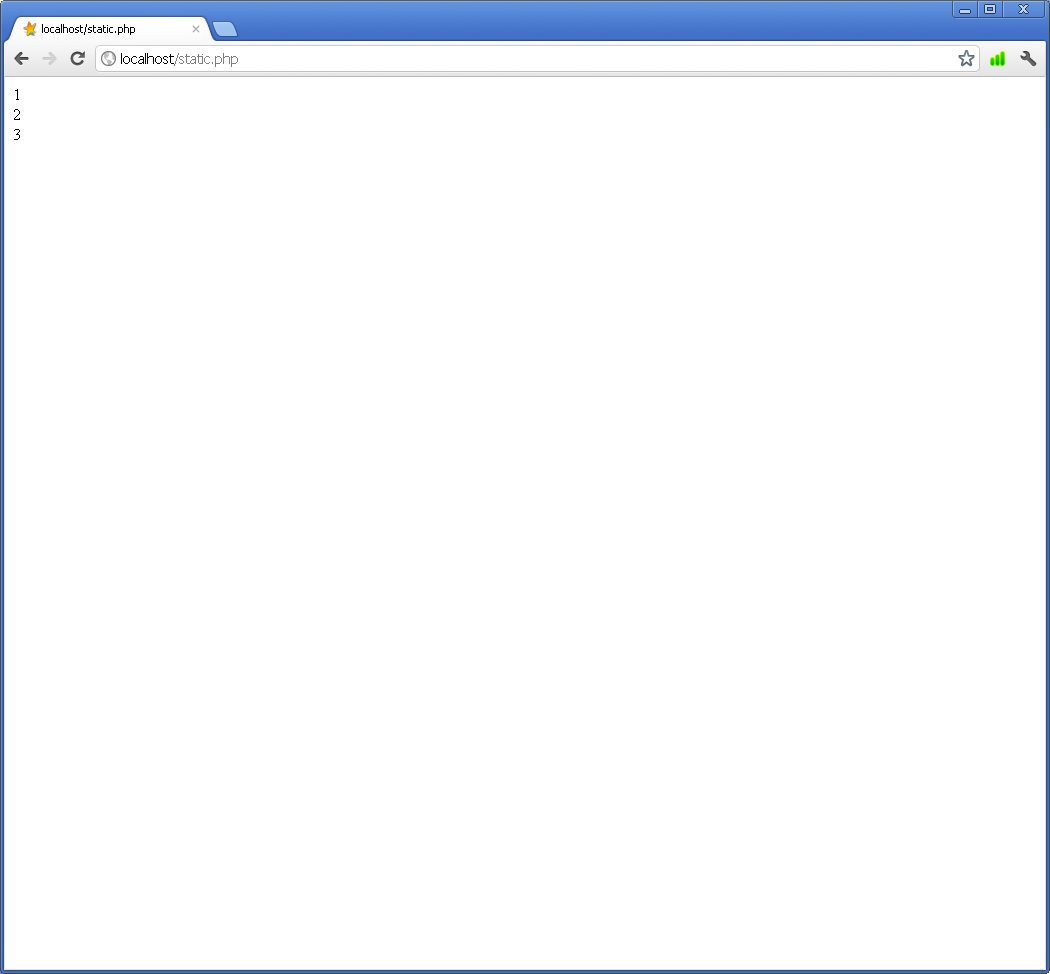


Рис. 6.1. Результат выполнения функции *keep\_track*( )

**Задание**

Создайте РНР-тест по языку программирования JavaScript, состоящий из десяти вопросов. Для ввода ответов используйте текстовые поля ввода, выпадающие списки, переключатели (*radio*), флажки (*checkbox*). При завершении теста (нажатии соответствующей кнопки) должна вызываться функция подсчета и вывода результата. Вывод результата осуществите в соответствии с вариантом.

Результат выполнения лабораторной работы сохраните в файле *lr6.php.*

***Вариант 1***. Результат выводится на той же странице: созда-ется новое поле ввода, в которое помещается отметка за тест.

***Вариант 2*.**Результат выводится на той же странице в теле документа после теста: выводятся отметка, номера вопросов, на которые дан правильный ответ, и номера вопросов, на которые дан неправильный ответ.

***Вариант 3.***Результат выводится на той же странице в теле документа после теста: выводятся отметка, номера вопросов, на которые дан неправильный ответ, и правильные варианты ответа на данные вопросы.

***Вариант 4.***Результат выводится на той же странице в теле документа после теста: выводится процентное соотношение пра-вильных и неправильных ответов.

***Вариант 5.***Результат выводится на новой странице: выво-дятся отметка за тест, текст вопросов, на которые был дан неверный ответ, и правильные варианты ответа.

***Вариант 6.***Результат выводится на новой странице: выводятся процентное соотношение правильных и неправильных ответов, текст вопросов, на которые был дан неверный ответ, и правильные варианты ответа.

***Вариант 7.***Результат выводится на той же странице в теле документа перед формой теста: выводится процентное соотно-шение правильных и неправильных ответов.

***Вариант 8.*** Результат выводится на той же странице в теле документа перед формой теста: выводятся отметка, номера вопро-сов, на которые дан неправильный ответ, и правильные варианты ответа на данные вопросы.

***Вариант 9.***Результат выводится на той же странице в теле документа перед формой теста: выводятся отметка, номера воп-росов, на которые дан правильный ответ, и номера вопросов, на которые дан неправильный ответ.

**Контрольные вопросы**

1. Как классифицируются переменные PHP по области види­мости?

2. Как указать, что переменная является глобальной? Приведите два способа.

3. В чем особенность использования статических переменных?

**Отчет по лабораторной работе**

1. Название и цель работы. Листинг *lr6.php*.

2. Файл *lr6.php* на диске.

ЛАБОРАТОРНАЯ РАБОТА 7

**Создание и обработка массивов**

*Цель работы:* формирование практических умений создания и обработки массивов.

**Теоретические сведения**

Существуют два типа массивов, различающихся по способу идентификации элементов. В массивах первого типа, называе­мых *индексированными*, элемент определяется индексом в по­следовательности. Массивы второго типа имеют ассоциативную природу: для обращения к их элементам используются ключи, логически связанные со значениями. Они соответственно назы­ваются *ассоциативными*.

По размерности массивы делятся на одномерные и много­мерные.

Одномерные индексированные массивы создаются следую­щим образом:

$a[0] = "first";

$a[1] = "second";

При выполнении команды *print $a[1];*в браузере выводится строка *second*.

При создании массивов можно воспользоваться функцией *array ().Такой же* массив *$a* может быть создан и следующим образом:

$a = аrrау("first", "second");

Чтобы включить новый элемент в конец массива, можно просто присвоить значение переменной массива без указания индекса. Следовательно, массив *$a* можно создать еще одним способом:

$a[] = "first";

$a[] = "second";

Основные функции поиска элементов в массиве:

in\_array(элемент, массив) – проверяет, присутствует ли в мас-сиве заданный элемент; возвращает true или false;

*array\_values(массив)*– возвращает массив, состоящий из всех значений исходного массива, переданного в качестве параметра;

array\_keys (массив [,искомый\_элемент]) –возвращает мас­сив, содержащий все ключи исходного массива, переданного в качестве параметра. Если при вызове передается дополнитель­ный параметр искомый\_элемент, возвращаются только ключи, которым соответствует заданное значение; в противном случае возвращаются все ключи массива.

Основные функции добавления и удаления элементов:

array\_push(массив, элемент 1[,элемент 2 [, ...]) –присоединяет к концу массива один или несколько новых элементов;

аrrау\_рор(массив) –удаляет последний элемент из массива; функцией возвращается извлеченный элемент;

array\_shift(массив) –удаляет первый элемент из массива; функцией возвращается извлеченный элемент;

array\_unshift(массив, переменная1 [,переменная2]) –добав-ляет в начало массива новый элемент, а остальные элементы сдвигаются на одну позицию вправо.

Функции определения количества элементов массива:

sizeof (массив) –возвращает количество элементов в массиве;

count (переменная) –возвращает количество значений, со­держащихся в массиве.

В некоторых ситуациях функция *count ( )* также возвращает дополнительную информацию, если:

- переменная существует и является массивом, *count( )* возвращает количество элементов в массиве;

- переменная существует, но не является масси­вом, функция возвращает значение 1;

- переменная не существует, возвращается значение 0.

Функции сортировки массивов:

sort (массив)–сортирует по возрастанию значений;

rsort (массив)–сортирует по убыванию значений;

*asort(массив)*– сортирует по возрастанию значений, сохра-няя исходную ассоциацию индексов с элементами;

*arsort (массив)* – сортирует по убыванию значений, сохраняя исходную ассоциацию индексов с элементами;

ksort (массив)– сортирует массив по ключам (по возраста-нию), сохраняя исходные ассоциации ключей со значениями;

krsort (массив)–сортирует массив по ключам (по убыва-нию), сохраняя исходные ассоциации ключей со значениями.

Другие функции работы с массивами:

*explode (разделитель, строка [,порог])*–делит строку на элементы и возвращает эти элементы в виде массива; разбиение выполняется по каждому экземпляру разделителя, при этом количество полученных фрагментов может ограничиваться необя-зательным параметром *порог*;

*array\_reverse(массив)* – изменяет порядок следования эле­ментов на обратный;

array\_flip(массив) –меняет местами ключи и значения элементов массива;

array\_merge(массив1, массив2, ..., массивN) –объединяет от 1 до N массивов в соответствии с порядком перечисления в параметрах;

array\_slice(массив, смещение [,длина]) –возвращает часть массива, начальная и конечная позиция которой определяется смещением от начала и необязательным параметром длина;

array\_splice(входной\_массив, смещение [,длина] [,заменяю-щий\_массив]) –заменяет часть массива, определяемую началь-ной позицией и необязательной длиной, элементами необязатель-ного параметра-массива.

**Конструкция *foreach***

Конструкция *foreach* представляет собой разновидность *for*, использующуюся для упрощения перебора элементов массива. Существуют две разновидности команды *foreach*, предназна-ченные для разных типов массивов:

foreach (массив as $элемент) {

блок

}

foreach (массив as $ключ => $элемент) {

блок

}

Например, при выполнении следующего фрагмента:

$menu = аrrау("first", "second", "third", "fourth");

foreach ($menu as $item) {

print "$item <br>";

}

будет выведен такой результат:

first

second

third

fourth

Конструкция *foreach* автоматически возвращается в начало массива (в других циклических конструкциях этого не происхо­дит). К тому же, нет необходимости явно увеличивать счетчик или иным способом переходить к следующему элементу массива – это происходит автоматически при каждой итерации *foreach*.

**Создание и использование библиотек функций**

Повысить эффективность программирования и способство­вать многократному использованию кода позволяет выделение функций в отдельный файл, называемый *библиотекой.* Биб­лиотеки удобны тем, что их функции можно использовать в раз­ных приложениях, не создавая лишних копий и не рискуя допус­тить ошибки в процессе копирования.

Файл библиотеки функций содержит описания функций, за­ключенные в «скобки» *<?php* … ?*>*. Он обычно сохраняется с расширением .*inc*.

Библиотеку функций можно включить в сценарий при по­мощи конструкций РНР *include* и *require*.

В общем виде синтаксис использования этих конструкций выглядит следующим образом:

*include "путь/имя\_файла";*

*require "путь/имя\_файла";*

Конструкция *require* позволяет включать файлы в сценарий PHP *до* исполнения сценария PHP. В отличие от конструкции [*require*](http://php.su/learnphp/cs/?ic#require), конструкция *include* включает файлы в код PHP-скрипта *во время* выполнения сценария. Конструкцию [*require*](http://php.su/learnphp/cs/?ic#require) целесооб­разнее использовать там, где не требуется динамическое вклю­чение файлов в сценарий, а конструкцию ***include*** только с целью динамического включения файлов.

**Задания**

**Задание 1.**Осуществите обработку одномерного массива в соответствии с вариантом (функции обработки должны хранить-ся в отдельном файле *func.inc*). Элементы массива должны вводиться в текстовое поле и разделяться пробелом. Результат выдается на той же странице. Сохраните основной файл под именем *lr7.php****.***

***Вариант 1.*** Вычисление среднего арифметического положи-тельных элементов; проверьте, являются ли элементы массива невозрастающей последовательностью.

***Вариант 2.*** Вычисление среднего арифметического отрицательных элементов, проверьте, являются ли элементы массива возрастающей последовательностью.

***Вариант 3.*** Проверьте наличие одинаковых значений; про-верьте, являются ли элементы массива неубывающей последо-вательностью.

***Вариант 4.***Поиск наибольшего по модулю элемента; про-верьте, являются ли элементы массива убывающей последова-тельностью.

***Вариант 5.***Поиск наименьшего по модулю элемента; оп-ределите количество нулевых элементов.

***Вариант 6.*** Определите модуль наименьшего элемента, и наи-больший элемент среди элементов, стоящих на нечетных местах.

***Вариант 7.***Определите количество элементов массива, являющихся нечетными числами, и наименьший элемент среди элементов, стоящих на нечетных местах.

***Вариант 8.*** Определите количество элементов массива, имею-щих нечетные номера и являющихся четными числами; наи-меньший среди элементов, стоящих на четных местах.

***Вариант 9.*** Определите количество элементов массива, имею-щих четные номера и являющихся нечетными числами; наиболь-ший элемент среди элементов, стоящих на четных местах.

**Задание 2.** Получите у преподавателя дополнительное зада-ние и выполните его. Сохраните под именем *lr7\_dop.php*.

**Контрольные вопросы**

1. Какие типы массивов поддерживаются PHP?

2. Что такое ассоциативный массив?

3. Как создать массив?

4. Как преобразовать строку в массив?

5. Укажите способы добавления элемента в конец массива.

6. Что такое библиотека функций? Для чего она используется?

7. Чем отличаются конструкции *include* и *require*?

**Отчет по лабораторной работе**

1. Название ицель работы. Листинг файлов*func.inc,lr7.php* и *lr7\_dop.php.*

2. Файлы *func.inc,lr7.php, lr7\_dop.php* на диске.

ЛАБОРАТОРНАЯ РАБОТА 8

**Обработка строк**

*Цель работы:* формирование практических умений обработ-ки строк.

**Теоретические сведения**

Длину строки в символах можно определить при помощи функции *strlen( )*, имеющей следующий синтаксис:

*strlen (строка)*

Функции удаления пробелов:

*chop(строка)*– возвращает строку после удаления из нее за­вершающих пропусков и символов новой строки;

*trim (строка)*– удаляет все пропуски с обоих краев строки и возвращает полученную строку;

*ltrim (строка)*–удаляет все пропуски и специальные сим­волы с левого края строки и возвращает полученную строку;

*rtrim (строка)* – удаляет все пропуски и специальные сим­волы с правого края строки и возвращает полученную строку.

Функции сравнения строк:

strcmp (строка1, строка2) –сравнивает две строки с учетом регистра символов.

После завершения сравнения *strcmp( )* возвращает одно из трех возможных значений:

- 0, если *строка1* и *строка2* совпадают;

- <0, если *строка1* меньше, чем *строка2*;

- 0, если *строка2* меньше, чем *строка1*.

*strcasecmp* – сравнивает строки без учета регистра символов. В остальном работает точно так же, как *strcmp*.

Функции деления и соединения строк:

*strtok (строка, разделители)*– разбивает строку по разде­лителям, заданным вторым параметром; чтобы полностью раз­делить строку, функцию необходимо последовательно вызвать несколько раз;

*parse\_str (строка)*– выделяет в строке пары «переменная-зна-чение» и присваивает значения переменным в текущей области видимости;

*implode (разделитель, массив)* – объединяет массив в строку.

Функции поиска и замены:

*strpos (строка, подстрока [,смещение])*– находит в строке позицию первого экземпляра заданной подстроки; необязатель­ный параметр *смещение* задает позицию, с которой должен на­чинаться поиск;

*strrpos (строка, символ)*– находит в строке последний эк­земпляр заданного символа;

*str\_replace (подстрока, замена, строка)*– ищет в строке все вхождения заданной подстроки и заменяет их новой подстрокой;

*strstr (строка, подстрока)* – возвращает часть строки, начи­нающуюся с первого вхождения заданной подстроки;

*substr (строка, начало [,длина])*– возвращает часть строки, начинающуюся с заданной начальной позиции и имеющую за­данную длину;

*substr\_count (строка, подстрока)*– возвращает количество вхождений подстроки в заданную строку;

*substr\_replace (строка, замена, начало [,длина])* – заменяет часть строки, которая начинается с заданной позиции; если за­дан необязательный параметр *длина*, заменяется фрагмент за­данной длины, в противном случае производится замена по всей длине заменяющей строки.

Функции преобразования строки к верхнему и нижнему ре­гистру:

*strtolower(строка)*–преобразует все алфавитные символы строки к нижнему регистру;

*strtoupper (строка )*– преобразует все алфавитные символы строки к верхнему регистру;

*ucfirst (строка)*– преобразует к верхнему регистру первый символ строки;

*ucwords (строка)*–преобразует к верхнему регистру первую букву каждого слова в строке.

**Регулярные выражения**

*Регулярное выражение*– это механизм, позволяющий задать шаблон для строки и осуществить поиск данных, соответст-вующих этому шаблону в заданном тексте. С помощью регу­лярных выражений можно изменить и удалить данные, разбить строку по шаблону на подстроки и многое другое. Одно из рас­пространенных применений регулярных выражений – это про­верка строки на соответствие каким-либо правилам.

Справочная информация о представлении символов в регу­лярных выражениях приведена в прил. 3.

В PHP существуют два различных механизма для обработки регулярных выражений: POSIX-совместимые и Perl-совместимые. Их синтаксис во многом похож, однако Perl-совместимые регуляр-ные выражения более мощные и работают намного быстрее.

Рассмотрим основные функции обработки Perl-совместимых регулярных выражений:

*preg\_match (рег. выражение, строка [, массив совпадений])*

*preg\_match\_all (рег. выражение, строка, массив совпадений [, порядок])*

*preg\_replace (рег.выражение, заменяющая строка, заменяе-мая строка [, предел])*

Функция *preg\_match* предназначена для проверки совпаде-ния заданной строки с заданным регулярным выражением. В ка-честве результата функция возвращает 1, если совпадения были найдены, и 0, если совпадений нет. Если при вызове функции был задан необязательный параметр *массив совпадений*, то после работы функции ему будет присвоен массив, содержащий результаты поиска по заданному регулярному выражению, при этом будет сохранено только первое совпадение.

Функция *preg\_match\_all* также предназначена для проверки совпадения строки с регулярным выражением. Однако она осу­ществляет поиск во всей заданной строке и возвращает *все* име-ющиеся совпадения.

Значение параметра *порядок* определяет структуру выход-ного массива с найденными совпадениями. Его значение может быть одним из следующих:

*PREG\_PATTERN\_ORDER* – результаты поиска будут сгруп-пированы по номеру регулярного выражения, которое возвратило этот результат (это значение используется по умолчанию);

*PREG\_SET\_ORDER* – результаты поиска будут сгруппиро-ваны по месту их нахождения в тексте.

Функция *preg\_replace ( )* производит замену текста по регу­лярному выражению. Задание необязательного параметра *предел* позволяет ограничить количество заменяемых фрагментов в тексте.

**Задания**

**Задание 1.** Выполните обработку строки в поле ввода, используя регулярные выражения, в соответствии с вариантом. Результат обработки должен выводиться рядом с исходной строкой. Сохраните файл выполнения задания под именем *lr8.php*.

***Вариант 1.*** Выделите все числа в строке синим цветом.

***Вариант 2***. Замените все пробелы из строки символами + и выделите их красным цветом.

***Вариант 3.*** Выделите все латинские буквы курсивом.

***Вариант 4.*** Замените все строчные английские буквы на соответствующие прописные.

***Вариант 5*.** Замените последовательности из двух или более пробелов одним пробелом.

***Вариант 6.*** Подчеркните все гласные русские буквы.

***Вариант 7.*** Замените все гласные строчные английские буквы на прописные.

***Вариант8.*** Выделите полужирным шрифтом все номера те-лефонов (формат +375(код)xxx-xx-xx).

***Вариант 9*.** Выделите курсивом все цифры и латинские буквы.

**Задание 2.** Получите у преподавателя дополнительное зада-ние и выполните его. Сохраните под именем *lr8\_dop.php*.

**Контрольные вопросы**

1. Как определить длину строки?

2. Какое значение может возвращаться функциями сравнения строк?

3. Что такое регулярное выражение? Для чего используются регулярные выражения?

4. Создайте регулярное выражение для поиска текста из шести символов, включающего строчные буквы русского алфавита и пробелы.

5. Для чего используется функция *preg\_match()*? Как отличается результат ее выполнения от результата выполнения функции *preg\_match\_all()*?

6. Какая функция выполняет замену текста, соответствующего ре-гулярному выражению?

7. Какие функции используются для преобразования символов строки к верхнему регистру?

**Отчет по лабораторной работе**

1. Файлы *lr8.php* и *lr8\_dop.php* на диске.

2. Название и цель работы. Листинг файлов *lr8.php* и *lr8\_dop.php*.

ЛАБОРАТОРНАЯ РАБОТА 9

**Использование функций для работы**

**с файлами и каталогами**

*Цель работы:* формирование практических умений исполь­зования функций работы с файлами и каталогами.

**Теоретические сведения**

Прежде чем начать работать с файлом, нужно убедиться в том, что он существует. Для решения этой задачи обычно ис­пользуются две функции: *file\_exists* и *is\_file*.

Функция *filе\_ехists* проверяет существование заданного фай-ла. Если файл существует, функция возвращает *true*, в про­тивном случае возвращается *false*. Синтаксис функции:

file\_exists(файл)

Функция *is\_file* проверяет не только существование задан­ного файла, но и возможность выполнения с ним операций чте­ния/записи. Синтаксис функции:

is\_file(файл)

Для определения размера файла используется функция *filesize( )*. Она возвращает размер (в байтах) файла с заданным именем или *false* в случае ошибки. Синтаксис функции:

filesize(имя\_файла)

**Открытие и закрытие файлов**

Прежде чем выполнять операции ввода/вывода с файлом, необходимо открыть его функцией *fopen( )*.

Функция *fopen* открывает файл (если он существует) и воз­вращает целое число –так называемый *файловый манипулятор* (*file handle*). Синтаксис функции:

fopen (файл, режим [,включение\_пути])

Если необязательный третий параметр *включение\_пути* ра­вен 1, то путь к файлу определяется по отношению к каталогу включаемых файлов, указанному в файле *php.ini*.

Функция *fclose* закрывает файл с заданным манипулятором. При успешном закрытии возвращается *true*, при неудаче –*false*. Синтаксис функции:

fclose(файловый манипулятор)

Параметр *режим* определяет возможность выполнения чте­ния и записи в файл.

В табл. 9.1 перечислены некоторые значения, определяющие режим открытия файла.

Таблица 9.1

|  |  |
| --- | --- |
| Режим | Описание |
| *r* | Только чтение. Указатель текущей позиции устанавлива-ется в начало файла |
| *r+* | Чтение и запись. Указатель текущей позиции устанав-ливается в начало файла |
| *w* | Только запись. Указатель текущей позиции устанавлива-ется в начало файла, а все содержимое файла унич-тожается. Если файл не существует, функция пытается создать его |
| *w+* | Чтение и запись. Указатель текущей позиции устанав-ливается в начало файла, а все содержимое файла уничтожается. Если файл не существует, функция пытается создать его |
| *a* | Только запись. Указатель текущей позиции устанавлива-ется в конец файла. Если файлне существует, функция пытается создать его |
| *a+* | Чтение и запись. Указатель текущей позиции устанавливается в конец файла. Если файл не существует, функция пытается создать его |

**Запись в файл**

С открытыми файлами выполняются две основные опера­ции: чтение и запись.

Функция *is\_writeable* позволяет убедиться в том, что файл существует и для него разрешена операция записи. Возможность записи проверяется как для файла, так и для каталога. Синтаксис функции:

is\_writeable (файл)

Функция *fwrite* записывает содержимое строковой перемен-ной в файл, заданный файловым манипулятором. Синтаксис функции:

fwrite(файловый манипулятор, переменная [,длина])

Если при вызове функции передается необязательный параметр *длина*, запись останавливается либо после записи указанного количества символов, либо при достижении конца строки.

**Чтение из файла**

Функция *is\_readable* позволяет убедиться в том, что файл существует и для него разрешена операция чтения. Возможность чтения проверяется как для файла, так и для каталога. Синтаксис функции:

is\_readable (файл)

Функция *fread* читает из файла, заданного файловым манипулятором, указанное количество байт. Чтение прекращается после прочтения заданного количества байт или при достижении конца файла. Синтаксис функции:

fread(файловый манипулятор, длина)

Функция *fgetc* возвращает строку, содержащую один сим-вол из файла в текущей позиции указателя, или *false* при до-стижении конца файла. Синтаксис функции:

fgetc (файловый манипулятор)

Функция *fgets* возвращает строку, прочитанную от текущей позиции указателя в файле, определяемом файловым манипуля-тором. Синтаксис функции:

fgets (файловый манипулятор, длина)

Чтение прекращается при выполнении одного из следующих условий:

- из файла прочитана длина;

- из файла прочитан символ новой строки (включается в возвращаемую строку);

- из файла прочитан признак конца файла (EOF).

### Для выполнения всех функций чтения из файла файловый манипулятор должен ссылаться на открытый файл, доступный для чтения.

### Положение указателя текущей позиции

Функция *feof* возвращает указатель конца файла. Возвраща-ет *true*, если достигнут конец файла, то есть, если указатель файла установлен за концом файла. Синтаксис функции:

feof(файловый манипулятор)

Функция *fseek* устанавливает указатель файла на определен-ную позицию: со *смещением* на определенное количество байт (от начала файла, от его конца или от текущей позиции в зависимости от значения параметра *позиция отсчета*). В случае успешного завершения эта функция возвращает 0, а в случае неудачи ***–***1. Синтаксис функции:

*fseek(файловый манипулятор, смещение, позиция отсче-та=SEEK\_SET)*

Параметр *позиция отсчета* задает, с какого места отсчиты-вается *смещение*. В PHP для этого существуют три константы, равные, соответственно, 0, 1 и 2:

*SEEK\_SET****–*** устанавливает позицию с начала файла;

*SEEK\_CUR****–*** отсчитывает позицию относительно теку­щей позиции;

*SEEK\_END****–*** отсчитывает позицию относительно конца файла.

В случае использования последних двух констант параметр *смещение* может быть отрицательным.

Функция *ftell* возвращает положение указателя файла. Син­таксис функции:

*ftell(файловый манипулятор)*

**Чтение файла в массив**

Функция *file* загружает все содержимое файла в индексиру-емый массив. Каждый элемент массива соответствует одной стро-ке файла. Синтаксис функции:

file (файл [,включение\_пути])

Если необязательный параметр *включение\_пути* равен 1, то путь к файлу определяется по отношению к каталогу включаемых файлов, указанному в файле *php.ini*.

**Копирование и переименование файлов**

К числу других полезных системных функций, которые мо-гут выполняться в сценариях РНР, относятся копирование и переименование файлов на сервере. Эти операции выполняются двумя функциями: *сору* и *rename*.

Функция *сору* пытается скопировать файл *источник* в файл *приемник*; в случае успеха возвращается *true*, а при неудаче –*false*. Если файл *приемник* не существует, функция *сору( )*создает его. Синтаксис функции:

copy (источник, приемник)

Функция *rename* переименовывает файл. В случае успеха возвращается *true*, a при неудаче –*false*. Синтаксис функции:

rename (старое\_имя, новое\_имя)

**Удаление файлов**

Функция *unlink* удаляет файл с заданным именем. Синтак­сис функции:

unlink (файл)

При работе с РНР в системе Windows при использовании этой функции иногда возникают проблемы. В этом случае можно воспользоваться функцией *system( )* и удалить файл командой DOS *del*:

system ("del filename.txt");

Функции работы с каталогами:

dirname (путь) – извлекает путь из полного имени файла;

is\_dir (имя\_файла) – проверяет, является ли файл с задан­ным именем каталогом;

mkdir (путь, права\_доступа)– создает новый каталог; права доступа для каталогов указываются точно так же, как и для файлов (чаще всего параметр принимает значение 0770);

opendir (путь) – открывает манипулятор для работы с ката­логом;

closedir(манипулятор\_каталога) – закрывает манипулятор каталога, переданный в качестве параметра;

readdir(манипулятор\_каталога) –возвращает очередной эле-мент заданного каталога;

chdir (каталог)– осуществляет переход в каталог, заданный параметром;

rewinddir ( манипулятор\_каталога) – переводит указатель те-кущей позиции в начало каталога, открытого функцией opendir( ).

**Задания**

**Задание 1.**Выведите список файлов каталога, имя которого указывается в поле ввода формы. Сохраните результат выпол-нения под именем *lr9\_1.php*.

**Задание 2.**Создайте счетчик посещений страницы (коли-чество посещений должно храниться в текстовом файле).Сохраните результат выполнения под именем *lr9\_2.php*.

**Контрольные вопросы**

1. Какие функции выполняют проверку существования файла?

2. Что такое файловый манипулятор?

3. Перечислите режимы открытия файла.

4. Какие операции можно выполнять с каталогами?

5. Как удалить файл, используя PHP? Укажите два способа.

6. Какие возможности предоставляет PHP по работе с файлами?

**Отчет по лабораторной работе**

1. Название и цель работы.Листинг файлов *lr9\_1.php* и *lr9\_.php*.

2. Файлы *lr9\_1.php*и *lr9\_.php* на диске.

ЛАБОРАТОРНАЯ РАБОТА 10

**Использование функцииdate()**

*Цель работы:* формирование практических умений исполь­зования функции *date( )*.

**Теоретические сведения**

Практически все функции по работе с датой и временем, в том числе *date*, имеют дело с форматом данных, называемым временной меткой (*timestamp).*Она является целым числом, рав-ным количеству секунд между эпохой Unix (1 января 1970 года) и указанным временем.

Для получения временной метки, соответствующей конкретной дате, используется функция *mktime( )*.

Синтаксис функции:

mktime([часы] [,минуты] [,секунды] [,месяц] [,день] [,год ] [,переход на летнее время]);

Все параметры этой функции необязательны, но пропускать их можно только справа налево. Если какие-то параметры не за­даны, на их место подставляются значения, соответствующие текущей дате.

Аргумент *переход на летнее время* указывает, осуществлен ли переход на летнее время (1) или нет (0); если не известно, то аргумент равен ***–***1.

Правильность даты, переданной в параметрах, не проверя­ется.

Для форматирования даты/времени, представленных вре­менной меткой, используется функция *date(формат [, временная метка]).*Эта функция возвращает строку, содержащую дату и время, отформатированную согласно строке *формат*, используя *временную метку* или текущее локальное время, если не задана временная метка.

В параметре *формат* могут использоваться следующие сим-волы:

*a****–*** "до" и "после" полудня: "am" или "pm";

*A****–*** "До" и "После" полудня: "AM" или "PM";

*d****–*** день месяца, 2 цифры (на первом месте ноль) (от 01 до 31);

*D****–*** день недели, текстовый, 3 буквы (например, "Fri");

*j****–*** день месяца, 1***–***2 цифры без начальных нулей (от 1 до 31);

*F****–*** месяц, текстовый, длинный (например, "January");

*h****–*** час, 12-часовой формат (от 01 до 12);

*H****–*** час, 12-часовой формат без нулей (от 1 до 12);

*G****–*** час, 24-часовой формат без нулей (от 0 до 23);

*i****–*** минуты (от 00 до 59);

*I****–*** 1, если действует переход на летнее время, иначе 0;

*L****–*** 0, если год не високосный, или 1 в противном случае;

*l****–*** день недели, текстовый, длинный (например, "Friday");

*m****–*** месяц, две цифры с нулями (от 01 до 12);

*n****–*** месяц, одна-две цифры без нулей (от 1 до 12);

*M****–*** трехбуквенное английское сокращение месяца (напри-мер, "Jan");

*t****–*** число дней в указанном месяце (от 28 до 31) ;

*s****–*** секунды (от 0 до 59);

*S****–*** англоязычный порядковый суффикс числа из двух букв, текстовый, то есть "th", "nd";

*U****–*** целое число секунд, прошедших с начала века (доступно не всегда);

*Y****–*** год, цифровой, 4 цифры (1999);

*y****–*** год, цифровой, 2 цифры (99);

*w****–*** порядковое число дня в неделе (от 0 – воскресенье до 6 – суббота);

*z****–*** порядковое число дня в году (от 0 до 365).

Все остальные символы в строковом аргументе *формат* возвращаются в результирующей строке "как есть".

**Задания**

**Задание 1.** Создайте простейшую гостевую книгу (*lr10\_1.php*). На странице должна помещаться форма с полями для ввода име-ни пользователя и сообщения, а также все предыдущие сообщения, упорядоченные по времени (первым выводится сообщение, оставленное позже всех остальных). Следует выполнять проверку корректности даты. Сообщения должны храниться в текстовом файле.

**Задание 2.**Определите и выведите возраст человека по дате его рождения, введенной в поле формы (сохраните код под име-нем *lr10\_2.php*).

**Контрольные вопросы**

1. Что представляет собой временная метка?

2. Как получить временную метку?

3. Для чего используется функция *date( )*?

4. Можно ли при помощи функции *mktime( )* проверить пра-­

вильность даты?

**Отчетпо лабораторной работе**

1. Название и цель работы. Листинги файлов *lr10\_1.php* и

*lr10\_2.php*.

2. Файлы *lr10\_1.php*и*lr10\_2.php* на диске.

ЛАБОРАТОРНАЯ РАБОТА 11

**Работа с сессиями**

*Цель работы:* формирование практических умений работы с сессиями.

**Теоретические сведения**

*Сессия* – механизм, позволяющий хранить некоторые данные, индивидуальные для каждого пользователя (например: его логин и пароль), между запусками сценария.

**Инициализация сессии и регистрация переменных**

Функция *session\_start ( )* инициализирует механизм сессий для текущего пользователя, запустившего сценарий.

Если посетитель запускает программу впервые, у него уста­навливается Cookies с уникальным идентификатором, и созда­ется временное хранилище, ассоциированное с этим идентифи­катором. Если данный пользователь уже запускал программу, определяется, какое хранилище связано с текущим идентифика­тором пользователя. Если в хранилище имеются какие-то пере­менные, их значения восстанавливаются, то есть создаются гло­бальные переменные, которые были сохранены в сессии при предыдущем завершении сценария.

Обратите внимание, что до функции инициализации не долж-но быть никакого вывода в браузер – иначе PHP не сможет установить идентификатор сессии (SID) для пользователя. Функция всегда возвращает значение *true*.

Функция *session\_register* указывает PHP на то, что ту или иную переменную нужно сохранить в сессии.

Синтаксис функции:

*session\_register(переменная1 [,переменная2, ...])*

Функция принимает в качестве параметров одно или не-сколько имен переменных (имена задаются в скобках, без знака *$* слева), регистрирует их в текущей запущенной сессии и возвращает *true*, если регистрация прошла успешно. Повторная запись одной и той же переменной в сессии не приведет к ошибке.

Например,

*<?*

*session\_start();*

*session\_register("count");*

*$count=@$count+1;*

*?>*

*<h2>Счетчик</h2>*

*В текущей сессии работы с браузером вы открыли эту страницу.*

*<?=$count?> раз(а). Закройте браузер, чтобы обнулить счетчик.*

**Имя группы сессии**

На одном и том же сайте могут существовать сразу несколько сценариев, которые нуждаются в услугах поддержки сессий PHP. Временные хранилища для сессий должны выбираться не только на основе идентификатора пользователя, но и на основе того, какой из сценариев запросил обслуживание сессии. Для того чтобы разграничить данные сессии, принадлежащей одному сценарию, от сессии, принадлежащей другому сценарию, создаются группы сессий и им даются имена. Сценарий, «знающий» имя своей группы сессии, сможет получить к ней доступ. Таким образом, сценарии будут избавлены от проблем с пересечением имен переменных.

Функция *session\_name* устанавливает или возвращает имя группы сессии, которая будет использоваться PHP для хранения зарегистрированных переменных.

Синтаксис функции:

*session\_name([$новое имя]);*

Если параметр *новое имя* не задан, то возвращается текущее имя. Если же этот параметр указан, то имя группы будет изменено на новое, при этом функция вернет предыдущее имя.

Функцию *session\_name* нужно вызывать до инициализации сессии – до вызова *session\_start*.

Если функция *session\_name* не была вызвана до инициали­зации, PHP будет использовать имя по умолчанию – *PHPSESSID*.

Например,

*<?*

*session\_name("CounterScript");*

*session\_start();*

*session\_register("count");*

*$count=@$count+1;*

*?>*

*В текущей сессии вы открыли эту страницу <?=$count?> раз(а).*

**Идентификатор сессии**

*Идентификатор сессии* (SID)является именем временного хранилища, которое будет использовано для хранения данных сессии между запусками сценария. Все SID уникальны и однозначно определяют сессию на компьютере, выполняющем сценарий, независимо от имени сессии.

Функция *session\_id* возвращает текущий идентификатор сес-сии SID.

Синтаксис функции:

*session\_id([$SID])*

Если задан параметр, то у активной сессии идентификатор изменяется на *$SID*.

Другие функции управления сессиями:

*session\_is\_registered($имя\_переменной)* – проверяет, зарегистрирована или нет та или иная переменная;

*session\_unregister($имя\_переменной)* – отменяет регистра­цию переменной, не уничтожая ее; возвращает *true* (если все прошло успешно) или *false*;

*session\_unset() –* отменяет регистрацию переменных сессии и уничтожает глобальные переменные.

**Задание**

Создайте тест по языку программирования PHP, состоящий из десяти вопросов. Перед прохождением теста пользователь должен зарегистрироваться (ввести имя, номер группы). Каждый вопрос должен выводиться на отдельной странице, промежу-точный результат – сохраняться в переменной сессии. По прохождении теста должен осуществляться вывод информации о пользователе и количество набранных им баллов.

**Контрольные вопросы**

1. Для чего используются сессии?

2. Как выполнить инициализацию сессии?

3. Что такое идентификатор сессии? Для чего он используется?

4. Как получить идентификатор сессии?

5. Как зарегистрировать переменную в сессии?

6. Чем отличаются функции *session\_unregister* и *session\_unset( )*?

**Отчет по лабораторной работе**

1. Название и цель работы. Листинг программного кода.

2. Файлы выполненного задания на диске.

ЛАБОРАТОРНАЯ РАБОТА 12

**Создание базы данных MySQL**

**и разработка Web-интерфейса**

**для ее просмотра и редактирования**

*Цель работы:* формирование практических умений созда­ния базы данных MySQL и разработки Web-интерфейса для ее просмотра и редактирования.

**Теоретические сведения**

Для создания и администрирования СУБД MySQL исполь-зуется *phpMyAdmin* – Web-приложение с открытым кодом, написанное на языке PHP и представляющее собой Web-интерфейс для администрирования. Для его вызова нужно в адресной строке браузера набрать *localhost/phpmyadmin*.

Разработка Web-интерфейса для просмотра и редактирова­ния базы данных MySQL осуществляется путем использования стандартных функций PHP.

Общая последовательность действий при взаимодействии с сервером MySQL следующая:

1) установить соединение с сервером MySQL. Если попытка завершается неудачей, вывести соответствующее сообщение и завершить процесс;

2) выбрать базу данных сервера MySQL. Если попытка вы-бора завершается неудачей, вывести соответствующее сообщение и завершить процесс. Допускается одновременное открытие нескольких баз данных для обработки запросов;

3) обработать запросы к выбранной базе (или базам);

4) после завершения обработки запросов закрыть соедине-ние с сервером баз данных.

**Подключение к серверу MySQL**

Функция *mysql\_connect* устанавливает связь с сервером MySQL. Синтаксис функции:

mysql\_connect (хост [:порт] [,имя пользователя] [,пароль])

В параметре *хост* передается имя хостового компьютера. Наряду с именем хоста могут указываться необязательные пара­метры, например, номер порта. Параметры *имя пользователя* и *пароль* должны соответствовать имени пользователя и паролю, заданным в таблицах привилегий MySQL. Все параметры являются необязательными, поскольку таблицы привилегий можно настроить таким образом, чтобы они допускали соединение без проверки. Если параметр *хост* не задан, *mysql\_connect( )* пытается установить связь с локальным хостом.

Пример открытия соединения с MySQL:

@mysql\_connect("localhost", "web", "4tf9zzzf") or die("Could not connect to MySQL server!");

Здесь *localhost*– имя компьютера, *web* – имя пользователя, а *4tf9zzzf* – пароль. Знак @ перед вызовом функции *mysql\_con-nect( )*подавляет все сообщения об ошибках, выдаваемые при неудачной попытке подключения, – они заменяются сообщением, указанным при вызове *die( )*.

Если программа устанавливает соединения с несколькими сер-верами MySQL на разных хостах, нужно сохранить идентификатор соединения, возвращаемый при вызове *mysql\_connect( )*, чтобы адресовать последующие команды нужному серверу MySQL.

**Выбор базы данных сервера MySQL**

После успешного соединения с MySQL необходимо выбрать базу данных, находящуюся на сервере. Для этого используется функция *mysql\_select\_db*. Синтаксис функции:

mysql\_select\_db (имя\_базы\_данных [,идентификатор\_соединения])

Параметр *имя\_базы\_данных* определяет выбираемую базу данных, идентификатор которой возвращается функцией *mysql\_select\_db*.Параметр *идентификатор\_соединения* необязателен лишь при одном открытом соединении с сервером MySQL. При наличии нескольких открытых соединений этот параметр должен указываться.

**Выполнение запроса к базе данных**

Функция *mysql\_query* обеспечивает интерфейс для обраще­ния с запросами к базам данных. Синтаксис функции:

mysql\_query (запрос [,идентификатор\_соединения])

Параметр *запрос* содержит текст запроса на языке SQL. Запрос передается либо соединению, определяемому необязательным параметром *идентификатор\_соединения*, либо, при отсутствии параметра, последнему открытому соединению.

При успешном выполнении команды *SQL SELECT* функцией возвращается идентификатор результата, который впоследст-вии передается функции *mysql\_result* для последующего форматирования и отображения результатов запроса. Если обработка запроса завершилась неудачей, функция возвращает *false*.

Чтобы узнать количество записей, участвующих в запросе SQL с командами *INSERT*, *UPDATE* или *DELETE*, используется функция *mysql\_affected\_rows*. Синтаксис функции:

mysql\_affected\_rows ([идентификатор\_соединения])

Для определения количества записей, возвращенных при вызове команды *SELECT*, используется функция *mysql\_num\_ro-ws(результат)*.

Для получения набора данных в сочетании с функцией *mysql\_query* используется функция *mysql\_result*. Синтаксис функции:

mysql\_result (идентификатор\_результата, запись [, поле])

В параметре *идентификатор\_результата* передается значе-ние, возвращенное функцией *mysql\_query*. Параметр *запись* ссылается на определенную запись набора данных, определяемого параметром *идентификатор\_результата*.В необязательном параметре *поле* могут передаваться:

- смещение поля в таблице;

- имя поля.

Для работы с большими наборами могут использоваться функции *mysql\_fetch\_row( )* и *mysql\_fetch\_array( )*.

Функция *mysql\_fetch\_row( )* присваивает значения всех полей записи элементам индексируемого массива (начиная с индекса 0). Использование функции *list( )* в сочетании с *mysql\_fetch\_row( )* позволяет сэкономить несколько команд, необходимых при исполь-зовании *mysql\_result( )*. Синтаксис функции:

mysql\_fetch\_row (результат)

Функция *mysql\_fetch\_array( )*аналогична *mysql\_fetch\_row( )*, однако по умолчанию значения полей записи сохраняются в *ассоциативном* массиве. Синтаксис функции:

mysql\_fetch\_array (идентификатор результата [, тип\_ин-дексации])

В параметре *идентификатор\_результата* передается значе-ние, возвращенное функцией *mysql\_query( )*. Необязательный параметр *тип\_индексации* принимает одно из следующих значений:

MYSQL\_ASSOC – функция *mysql\_fetch\_array( )* возвращает ассоциативный массив. Если параметр не указан, это значение используется по умолчанию;

MYSQL\_NUM – функция *mysql\_fetch\_array( )* возвращает масссив с числовой индексацией;

MYSQL\_BOTH – к полям возвращаемой записи можно обращаться как по числовым, так и по ассоциативным индексам.

**Закрытие соединения с сервером MySQL**

После завершения работы с сервером MySQL соединение необходимо закрыть. Функция *mysql\_close( )* закрывает соедине-ние, определяемое необязательным параметром. Если параметр не задан, функция *mysql\_close( )* закрывает последнее открытое соединение. Синтаксис функции:

mysql\_close ([идентификатор\_соединения])

Пример использования функции *mysql\_close( )*:

<?

@mysql\_connect("localhost", "web", "4tf9zzzf")

or die("Could not connect to MySQL server!");

@mysql\_select\_db("company") or die("Could not select company database!");

print "You're connected to a MySQL database!";

mysql\_close();

?>

**Задание**

1. Выполните проектирование базы данных, хранящей не-обходимую информацию конкретной предметной области, в соответствии с вариантом.

***Вариант 1.***Книжный магазин

***Вариант 2.***Библиотека

***Вариант 3.***Биржа труда

***Вариант 4.***Фирма по продаже компьютеров

***Вариант 5.***Салон по продаже автомобилей

***Вариант 6.***Ресторан

***Вариант 7.***Туристическая фирма

***Вариант 8.***Кондитерская фабрика

***Вариант 9.***Медицинский центр

2. Создайте соответствующую базу данных MySQL, используя PHPMyAdmin.

3. Разработайте Web-интерфейс базы данных, обеспечиваю-щий следующие возможности:

- вывод информации в виде таблицы;

- поиск информации по различным критериям (не менее трех);

- удаление, добавление данных;

- редактирование информации.

**Контрольные вопросы**

1. С помощью какого программного приложения выполняется создание и администрирование БД MySQL? Как его вызвать?

2. Перечислите основную последовательность действий при ра-

боте с базой данных MySQL.

3. Как установить соединение с сервером MySQL?

4. Для чего используется символ *@* при вызове команд?

5. Как осуществляется выбор базы данных?

6. Для чего используется функция *mysql\_query()*?

7. В чем различие функций *mysql\_affected\_rows( )* и

*mysql\_num\_rows( )*?

8. В чем различие функций *mysql\_result( )* и *mysql\_fetch\_row( )*?

9. Как представить результаты выполнения запроса в виде ассоциативного массива?

10. Как закрыть соединение с сервером mySQL?

**Отчет по лабораторной работе**

1. Название и цель работы. Схема БД. Листинг программного кода.

2. Файлы выполненного задания на диске.

**Рекомендуемая литература**

1. Гарольд, Э. XML : справочник : пер. с англ. / Э. Гарольд, С. Минс. – СПб. :Символ-Плюс, 2002. – 576 с.
2. Гилмор, Д. PHP : учебный курс /Д. Гилмор. –

<http://www.phpfaq.ru/txt/gilmor/>

1. Кищенко, [О. Н.](http://www.intuit.ru/lector/223.html) [Языки информационного обмена](http://www.intuit.ru/goto/course/lande/). Трансформация документа XML / О. Н. Кищенко.–

<http://www.intuit.ru/department/internet/lande/9/>

1. Колисниченко, Д. Н. Самоучитель PHP / Д. Н. Колисниченко. – СПб.: Наука и техника, 2004. – 578 с.
2. Основы использования XML-схем для определения элементов <http://www.ibm.com/developerworks/ru/library/xml-schema/>
3. Справочный портал по PHP и MySQL– http://php.su/
4. [Учебник по XML](http://www.realcoding.net/articles/internet-tekhnologii/uchebnik-po-xml)–<http://www.realcoding.net/articles/glava-10-xsl-tablmts-stilei.html>
5. [Шатохина, Н. А.](mailto:sna@uneta.org)Понимание XML Schema /Н.А.Шатохина.–

<http://www.vbnet.ru/articles/showarticle.aspx?id=151>

**Приложение 1**

(справочное)

**Возможные типы данных XSD**

|  |  |  |
| --- | --- | --- |
| Обозначение | Описание | Пример |
| String | Строка символов, включая симво-лы пробела, табуляции, возврата каретки и перевода строки | John Lennon |
| Boolean | Двоичное, логическое. Принимает значения: *true* или *false* (1 или 0) | false |
| Decimal | Десятичное число в виде последовательности десятичных цифр, включая дробную часть, отделяемую точкой | 3145.56 |
| Float | 32-битное число с плавающей точкой. Мантисса необязательна | 4.6E4 |
| Double | 64-битное число с плавающей точкой. Мантисса необязательна | 4.6E4 |
| hexBinary | Шестнадцатеричные данные в двоичном коде | 0FFF |
| base64Binary | Двоичные данные в коде *base*64 | GpM7 |
| anyURI | Универсальный индикатор ресурсов (URI) | http://cnet.com |
| QName | Квалифицированное имя XML согласно требованиям пространства имен (XMLNamespaces) | xs:element |
| Duration | Период времени в годах, месяцах, днях, часах, минутах, секундах | PnYnMnDTnHnMnS  P1M3DT4H (один месяц, 3 дня и 4 часа) |
| dateTime | Время в виде комбинации календарной даты и времени | CCYY-MM-DD THH:MM:SS  2012-03-26T15:30:01 |
| Time | Значение текущего времени | HH:MM:SS  15:30:01 |
| Date | Календарная дата | CCYY-MM-YY  2012-03-26 |
| gYearMonth | Месяц в году | CCYY-MM  2012-03 |
| Обозначение | Описание | Пример |
| gYear | Год | CCYY  2012 |
| gMonthDay | День в месяце в виде календарной даты | -MM-DD  -03-26 |
| gDay | День в виде календарного дня | -DD  -26 |
| gMonth | Месяц в виде календарного месяца | -MM  -03 |

**Приложение 2**

(справочное)

**Аспекты базового типа XSD**

|  |  |
| --- | --- |
| Аспект (Facet) | Описание |
| Enumeration | Определяет фиксированное значение, с которым должен совпадать тип |
| fractionDigits | Определяет максимальное количество десятичных знаков справа от десятичной точки |
| Length | Определяет количество символов в строковом типе, количество байтов в двоичном типе или количество элементов в списочном типе |
| maxExclusive | Определяет исключающую верхнюю границу области значений типа |
| maxInclusive | Определяет включающую верхнюю границу области значений типа |
| maxLength | Определяет максимальное количество символов в строковом типе, максимальное количество байтов в двоичном типе или максимальное количество элементов в списочном типе |
| minExclusive | Определяет исключающую нижнюю границу облас-ти значений типа |
| minInclusive | Определяет включающую нижнюю границу области значений типа |
| minLength | Определяет минимальное количество символов в строковом типе, минимальное количество байтов в двоичном типе или минимальное количество элементов в списочном типе |
| Pattern | Определяет шаблон, основанный на регулярном выражении, с которым должен совпадать тип |
| totalDigits | Определяет максимальное количество десятичных знаков для типов, унаследованных от *number* |
| whiteSpace | Определяет правила нормирования пробелов |

**Приложение 3**

(справочное)

**Представление символов в регулярных выражениях**

|  |  |
| --- | --- |
| Символ | Значение |
| **\** | Делает обычные символы специальными и наоборот. Например, выражение "s" ищет просто символ 's'. Если поставить слэш перед s, то "\s" уже обозначает пробельный символ. И наоборот, если символ специальный, например \*, то слэш сделает его просто обыч-ным символом «звездочка». Например, "a\*" ищет 0, или больше подряд идущих символов 'a'. Чтобы найти 'a\*', нужно поставить слэш перед специальным символом: "a\\*" |
| **^** | Обозначает начало входных данных. Если установ-лен флаг многострочного поиска ("m"), то также сработает при начале новой строки. Например, "^A" не найдет 'A' в "an A", но найдет первое 'A' в "An A" |
| $ | Обозначает конец входных данных. Если установлен флаг многострочного поиска, то также сработает в конце строки. Например, "t$" не найдет 't' в "eater", но найдет в "eat" |
| **\*** | Обозначает повторение 0 или более раз. Например, "bo\*" найдет 'boooo' в "A ghost booooed" и 'b' в "A bird warbled", но ничего не найдет в "A goat grunted" |
| **+** | Обозначает повторение 1 или более раз. Эквивалент-но {1,}. Например, "a+" найдет 'a' в "candy" и все 'a' в "caaaaaaandy" |
| ? | Обозначает, что элемент может как присутствовать, так и отсутствовать. Например, "e?le? " найдет 'el' в "angel" и 'le' в "angle" |
| **.**  (десятичная точка) | Обозначает любой символ, кроме перевода строки: \n \r \u2028 or \u2029. Например, ".n" найдет 'an' и 'on' в "nay, an apple is on the tree", ноне 'nay' |
| (*x*) | Находит *x* и запоминает. Это называется «запоми­наю­щие скобки». Например, "(foo) " найдет и запомнит 'foo' в "foo bar." Кроме того, скобки объединяют то, что в них находится, в единый элемент паттерна. Например, (abc)\* – повторение abc0 и более раз | |
|  |  | |
| Символ | Значение | |
| (?:*x*) | Находит *x*, но не запоминает найденное. Это называется «незапоминающие скобки». Как и все скобки, объединяют находящееся в них в единый паттерн | |
| *x*(?=*y*) | Находит *x*, только если за *x* следует *y*. Например, "Jack(?=Sprat)" найдет 'Jack', только если за ним следует 'Sprat'. "Jack(?=Sprat|Frost)" найдет 'Jack', только если за ним следует 'Sprat' или 'Frost'. Однако, ни 'Sprat' ни 'Frost' не войдут в результат поиска | |
| *x*(?!*y*) | Находит *x*, только если за *x* не следует y. Например, "\d+(?!\.)" найдет число, только если за ним не следует десятичная точка | |
| *x*|*y* | Находит *x* или *y*. Например, "green|red" найдет 'green' в "green apple" и 'red' в "red apple" | |
| {*n*} | Находит ровно *n* повторений предшествующего элемента (*n* – положительное целое число). Например, "a{2}" не найдет 'a' в "candy," но найдет оба 'a' в "caandy" и первые два 'a' в "caaandy" | |
| {*n*,} | Находит *n* и более повторений элемента (*n* – положи-тельное целое число). Например, "a{2,} " не найдет 'a' в "candy", но найдет все 'a' в "caandy" и в "caaaaaaandy" | |
| {*n*,*m*} | Находит от *n* до *m* повторений элемента (*n* и *m* – положительные целые числа) | |
| [*xyz*] | Находит любой из перечисленных символов. Можно указать промежуток, используя тире. Например, [abcd] – то же самое, что [a-d]. Найдет 'b' в "*brisket*", а также 'a' и 'c' в "ache" | |
| [^*xyz*] | Находит любой символ, кроме указанных в наборе. Также можно указать промежуток. Например, [^abc] – то же самое, что [^a-c]. Найдет 'r' в "brisket" и 'h' в "chop" | |
| [\*b*] | Находит символ backspace (не путать с \b) | |
| \*b* | Находит границу слов (латинских), например, про-бел. (Не путать с [\b]). Например, "\bn\w" найдет 'no' в "noonday"; "\wy\b" найдет 'ly' в "possibly yesterday*"* | |
| \*B* | Обозначает не границу слов. Например, "\w\Bn" найдет 'on' в "noonday", а "y\B\w" найдет 'ye' в "possibly yesterday" | |
| \*cX* | Находит контрольный символ в строке (*X* – буква от A до Z). Например, "\cM" обозначает символ Ctrl+M | |
| Символ | Значение | |
| \*D* | Находит нецифровой символ (все алфавиты). [^0-9] – эквивалент для обычных цифр. Например, "\D" или "[^0-9] " найдет 'B' в "B2 isthesuitenumber" | |
| \*n* | Находит перевод строки | |
| \*s* | Находит любой пробельный символ, включая пробел, табуляцию, переводы строки. Например, "\s\w\*" найдет ' bar' в "foo bar" | |
| \*S* | Находит любой символ, кроме пробельного. Например, "\S\w\*" найдет 'foo' в "foo bar" | |
| \*t* | Находит символ табуляции | |
| \*v* | Находит символ вертикальной табуляции | |
| \*w* | Находит любой словесный (латинский алфавит) символ, включая буквы, цифры и знак подчеркивания. Эквивалентно [A-Za-z0-9\_]. Например, "\w" найдет 'a' в "apple," '5' в "$5.28," и '3' в "3D" | |
| \*W* | Находит любой нелатинский словесный символ. Эквивалентно [^A-Za-z0-9\_]. Например, "\W" и "[^$A-Za-z0-9\_]" одинаково найдут '%' в "50%" | |
| \n | Обратная ссылка на *n*-ю запомненную скобками подстроку (*n* – целое число)  Например, "apple(,)\sorange\1" найдет 'apple, orange,' в "apple, orange, cherry, peach" | |
| \0 | Находит символ NUL | |

**Оглавление**

|  |  |
| --- | --- |
| Предисловие. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| Лабораторная работа 1  **Создание XML-документа**. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| Лабораторная работа 2  **Описание структуры XML-документа средствами DTD**. . . . . . | 10 |
| Лабораторная работа 3  **Описание схемы XML-документа на языке XSD**. . . . . . . . . . | 17 |
| Лабораторная работа 4  **Форматирование вывода XML-документа**. . . . . . . . . . . . . . . | 24 |
| Лабораторная работа 5  **Создание простейших PHP-сценариев**. . . . . . . . . . . . . . . . . . | 31 |
| Лабораторная работа 6  **Создание и использование функций**. . . . . . . . . . . . . . . . . . . . | 36 |
| Лабораторная работа 7  **Создание и обработка массивов**. . . . . . . . . . . . . . . . . . . . . . . . . . | 39 |
| Лабораторная работа 8  **Обработка строк** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 44 |
| Лабораторная работа 9  **Использование функций для работы с файлами**  **и каталогами** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 49 |
| Лабораторная работа 10  **Использование функций date( )** . . . . . . . . . . . . . . . . . . . . . . . . | 54 |
| Лабораторная работа 11  **Работа с сессиями**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 56 |
| Лабораторная работа 12  **Создание базы данных MySQL и разработка**  **Web-интерфейса для ее просмотра и редактирования**. . . . . . | 59 |
| Рекомендуемая литература. . . . . . . . . . . . . . . . . . . . . . . . . | 65 |
| Приложение 1 Возможные типы данных XSD . . . . . . . . . . . | 66 |
| Приложение 2 Аспекты базового типа XSD . . . . . . . . . . . . . | 68 |
| Приложение 3 Представление символов в регулярных  выражениях . . . . . . . . . . . . . . . . . . . . . . . . . . | 69 |