ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И СЕТЕЙ

Методические рекомендации к лабораторным работам для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника» дневной формы обучения

Могилев 2018

УДК 004.4
ББК 32.973.202-018.2
А76

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «13» марта 2018 г., протокол № 11

Составитель ст. преподаватель В. М. Прудников

Рецензент А. Е. Науменко

Методические рекомендации предназначены к лабораторным работам для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника» дневной формы обучения. Приведены задания и список литературы для подготовки.

Учебно-методическое издание

АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И СЕТЕЙ

Ответственный за выпуск	А. И. Якимов
Технический редактор	А. А. Подошевко
Компьютерная верстка	М. М. Дударева

Подписано в печать . Формат 60x84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 16 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев.

Содержание

Введение

Целью преподавания дисциплины «Аппаратное и программное обеспечение ЭВМ и сетей» является приобретение специальных знаний, умений и навыков, необходимых инженеру по информационным технологиям в процессе разработки Web-приложений.

Методические рекомендации имеют целью помочь студентам в подготовке и выполнении лабораторных работ по дисциплине.

Лабораторная работа № 1. Работа в Internet

Цель работы:
– повторение и изучение терминов и понятий;
– изучение интерфейса и настроек браузеров;
– изучение возможностей глобальной сети Internet.

Порядок выполнения работы

1 Ознакомиться со списком литературы по дисциплине.

2 Изучить и выписать в конспект состав меню браузеров MS Internet Explorer и Mozilla Firefox (либо другого).

3 Найти в Internet или в литературе различные варианты определений (минимум из двух источников) следующих терминов и понятий:

– Internet;
– адрес IP v.4;
– маска IP-адреса;
– адрес IP v.6;
– DNS (Domain Name System);
– DNS (Domain Name Service);
– WWW (World Wide Web);
– сайт;
– HTML (Hypertext Markup Language);
– CSS (Cascading Style Sheets).

4 Оформить отчет с найденными определениями (со ссылками на источники) в виде документа MS Word с гиперссылками на них по документу. При оформлении отчета использовать возможности MS Word по форматированию текстового документа.

Вопросы для контроля

1 Структура Internet.
2 Состав служб (сервисов) Internet.
3 Состав FQDN (Fully Qualified Domain Name).
4 Структура DNS (Domain Name System).
5 Назначение HTML.
6 Как в браузере:

– настроить параметры соединения с Internet;
– просмотреть HTML-код страницы;
– изменить кодировку для интерпретации страницы;
– установить домашнюю страницу;
– просмотреть список загрузок;
– задать стили пользователя (форматирование пользователя)?

Лабораторная работа № 2. Введение в HTML

Цель работы:
– освоить синтаксис HTML;
– изучить текстовое оформление Web-страниц;
– научиться вставлять изображения в Web-страницы.

Порядок выполнения работы

1 Ознакомиться с основами HTML.
2 Выполнить задание, полученное у преподавателя.
3 Оформить отчет в виде html-файла.

Основные теоретические положения

Web-документы используют язык HTML (Hypertext Markup Language – язык разметки гипертекста).

HTML-документ – это файл, содержащий обыкновенный текст, структурно размеченный (разметкой). Такой файл может быть подготовлен в произвольном текстовом редакторе (существуют, однако, специальные программы-конверторы и HTML-редакторы).

HTML-документ состоит из содержимого и команд, задающих структуру (разметка).

Каждая управляющая конструкция HTML-документа (тег) должна заключаться в угловые скобки: <тег>. Чаще всего в документе встречаются парные теги (открывающий и соответствующий ему закрывающий), т. к. браузеру необходимо знать область действия тега.

Открывающий и закрывающий теги называются одинаково и отличаются друг от друга только символом «наклонная черта» или «слэш» – «/», который ставится сразу после открывающей угловой скобки закрывающего тега. Закрытие парных тегов выполняется так, чтобы соблюдались правила вложения.

```
<b><i>На этот текст воздействуют два тега</i></b>
```

Кроме того, тег может включать атрибут (атрибуты), дающий дополнительную информацию браузеру, т. е. уточняющий действие тега.

Атрибуты представляют собой дополнительные ключевые слова, отделяемые от ключевого слова, определяющего тег, и от других атрибутов пробелами и размещаемые до завершающего тег символа « > ». Значение атрибута отделяется от ключевого слова атрибута символом « = » (знак равенства) и заключается в кавычки.

```
<h1 align="left">
```

Вопросы для контроля

1 Что называют разметкой?
2 Приведите примеры парных и непарных тегов.
3 В каких единицах измерения задается значение атрибута `size`?
4 С помощью какого тега можно увеличить размер шрифта?
5 При использовании какого тега появляются вертикальные отступы?
6 Чем отличаются абсолютные и относительные ссылки?

Лабораторная работа № 3. HTML. Создание списков

Цель работы:
– освоить синтаксис HTML;
– изучить порядок использования списков.

Порядок выполнения работы

1 Ознакомиться с основами применения списков в HTML.
2 Выполнить задание, полученное у преподавателя.
3 Добавить на страницу список трех определений из лабораторной работы № 1.
4 Оформить отчет в виде html-файла.

Основные теоретические положения

В HTML используются списки следующих видов:

– упорядоченные (нумерованные);
– неупорядоченные (ненумерованные);
– определений.

Перед пунктами неупорядоченных списков обычно ставятся символы-маркеры (bullets), например, точки, ромбики и т. п., в то время как пунктам упорядоченных списков предшествуют их номера. Тэги, используемые для создания списков, показаны в таблице 1.

Таблица 1 – Тэги списков

Тэг	Назначение
** – **	Создает неупорядоченный список
** – **	Создает упорядоченный список
** – **	Создает пункт списка внутри тэгов **ol** или **ul**
<dl> – </dl>	Открывает и закрывает список определений
<dt> – </dt>	Создает термин в списке определений внутри элемента **dl**
<dd> – </dd>	Создает определение термина внутри элемента **dl**

Вопросы для контроля

1 Какие существуют виды списков?
2 Чем отличаются нумерованные и маркированные списки?
3 С помощью каких атрибутов можно изменить порядок нумерации списка?
4 Какие существуют значения атрибута `type` для маркированных списков?
5 Как создать маркированный список без вертикальных отступов?
6 Расшифруйте теги `<dl>`, `<dt>`, `<dd>`. Для чего они предназначены?

Лабораторная работа № 4. HTML. Создание и применение таблиц

Цель работы: изучить порядок создания и использования таблиц при разработке Web-страниц.

Порядок выполнения работы

1 Ознакомиться с основами применения таблиц в HTML.
2 Выполнить задание, полученное у преподавателя.
3 Оформить отчет в виде html-файла.
4 К выполненному заданию по лабораторной работе № 3 добавить рисунок со ссылкой на созданную страницу, при нажатии на которую будет открываться сформированная страница с таблицей.

Основные теоретические положения

Таблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Обычно таблицы используются для упорядочения и представления данных, однако возможности таблиц этим не ограничиваются. С помощью таблиц удобно верстать макеты страниц, расположив нужным образом фрагменты текста и изображений.

Для добавления таблицы на web-страницу используется тег-контейнер **`<table>`.** Таблица должна содержать хотя бы одну строку и колонку.

Для добавления строк используются теги **`<tr>`** и **`</tr>`**. Чтобы разделить строки на колонки применяются теги **`<td>`** и **`</td>`.**

Для изменения вида и свойств таблицы используются атрибуты, которые добавляются в тег **`<table>`** (таблица 2)**.**

`<table атрибут1=... атрибут 2=...>`

Таблица 2 – Атрибуты таблицы и их значения

Атрибут	Значение	Описание	Пример
align=	**left** **right** **center**	Выравнивание таблицы	`align=center`
background=	**URL**	Фоновый рисунок	`background="pic.gif"`
bgcolor=	**#rrggbb**	Цвет фона таблицы	`bgcolor=#FF9900`
border=	**n**	Толщина рамки в пикселях	`border=2`
cellpadding=	**n**	Расстояние между ячейкой и ее содержимым	`cellpadding=7`
cellspacing=	**n**	Дистанция между ячейками	`cellspacing=3`
colspan=	**n**	Число ячеек, объединяемых по горизонтали.	`colspan=2`
rowspan=	**n**	Число ячеек, объединяемых по вертикали	`rowspan=3`
nowrap		Запрещает переносы строк в тексте	`<table nowrap>`
frame=	**void** **above** **below** **lhs** **rhs** **hsides** **vsides** **box**	Задание типа рамки таблицы	`frame=hsides`
valign=	**top** **bottom**	Выравнивание по высоте	`valign=top`
width=	**n, n%**	Минимальная ширина таблицы, в пикселях или процентах	`width=90%`
height=	**n, n%**	Минимальная высота таблицы, в пикселях или процентах	`height=18`

Вопросы для контроля

1 Какие теги предназначены для создания заголовка и подвала таблицы?

2 Каким способом можно создать рамку вокруг ячейки таблицы?

3 Для чего применяются атрибуты `colspan` и `rowspan`?

4 С помощью какого атрибута можно задать пространство между границей и содержимым ячейки?

5 Что произойдет, если для двух смежных ячеек задать разные значения ширины или высоты?

6 Для чего предназначен тег `<caption>`?

Лабораторная работа № 5. HTML. Создание и применение фреймов

Цель работы: изучить порядок создания и использования фреймов при разработке Web-страниц.

Порядок выполнения работы

1 Ознакомиться с основами применения фреймов в HTML.
2 Выполнить задание, полученное у преподавателя.
3 Содержимое фреймов № 1, 2, 3, 4 – HTML-страницы из лабораторных работ № 1, 2, 3, 4 соответственно.
4 Оформить отчет в виде html-файла.

Основные теоретические положения

Фреймы – это окна независимого просмотра HTML-документов. Иногда бывает очень удобно использовать навигацию по странице в виде меню, оформленного в отдельном фрейме, и основного окна, где будет представлена вся основная информация, определяемая действиями пользователя в области меню.

Для создания фрейма используется тег **<frameset>**, который заменяет тег **<body>** в документе и используется для разделения экрана на области. Внутри данного тега находятся теги **<frame**>, которые указывают на HTML-документ, предназначенный для загрузки в область.

Пример – Создание простого фрейма.

```
<html>
   <frameset cols="30%, 70%" border=0>
      <frame src="menu.html" name="Menu">
      <frame src="main.html" name="Main">
   </frameset>
</html>
```

В приведенном примере присутствует объявление двух фреймов, которые будут располагаться вертикальными полосами и занимать соответственно 30 и 70 % рабочей области.

Вертикальное расположение устанавливается параметром `cols="..."`, а для горизонтальных полос используют параметр `rows="..."`.

Параметр `border="..."` определяет границу между фреймами.

Вопросы для контроля

1 Что называют фреймами?
2 Какие теги участвуют в создании фреймов?
3 Какие атрибуты можно задать для тега `<frame>`?
4 Как можно управлять полосами прокрутки в фреймах?

5 Какие атрибуты можно задать для тега `<frameset>`?
6 С помощью каких тегов можно задать границу между фреймами?

Лабораторная работа № 6. HTML. Создание и применение форм

Цель работы: изучить порядок создания и использования форм при разработке Web-страниц.

Порядок выполнения работы

1 Ознакомиться с основами применения форм в HTML.
2 Выполнить задание, полученное у преподавателя.
3 Оформить отчет в виде html-файла.

Основные теоретические положения

HTML-формы предназначены для пересылки данных от удаленного пользователя к Web-серверу. С их помощью можно организовать простейший диалог между пользователем и сервером, например, регистрацию пользователя на сервере или выбор нужного документа из представленного списка. Формы поддерживаются всеми популярными браузерами.

Форма в HTML-коде ограничивается тэгами **`<form>`** и **`</form>`**, которые создают контейнер для размещения элементов управления. Элементы управления могут размещаться в таблице, которая, в свою очередь, полностью расположена в форме. Тэг **`<form>`** не создает какой-либо отображаемой структуры. Он, скорее, предназначен для внутренней группировки объектов.

Тэг **`<form>`** обладает целым рядом параметров (атрибутов), которые задают свойства создаваемой формы.

Параметр **`action`** является обязательным. Его значение – URL, указывающий на расположение того CGI-приложения, которое будет обрабатывать данные, введенные пользователем с помощью элементов управления искомой формы.

Атрибут **`method`** предназначен для указания способа, которым данные передаются обрабатывающему приложению. В качестве значения атрибута используется одно из двух предустановленных ключевых слов: **`get`** или **`post`**. По умолчанию используется значение **`get`**.

Атрибут **`enctype`** указывает тип данных, передаваемых из формы. Обычно нет необходимости его использовать, т. к. значение **`application/x-www-form-uriencoded`**, применяемое по умолчанию, идеально подходит для подавляющего большинства CGI-приложений.

Атрибут **`accept-charset`** задается в тех случаях, когда пользователь передает из формы приложению не только данные, но и файлы. В этом случае можно явно указать кодировки передаваемых файлов. В качестве значения данного атрибута используется текстовая строка, в которой записывается одно или несколько названий кодировок. Если применяется несколько кодировок, их

наименования разделяются пробелами или запятыми. По умолчанию используется значение **unknown**, которое указывает серверу, что он должен сам разобраться с применяемыми кодировками.

Атрибут **accept** описывает типы передаваемых файлов. Обычно не используется, т. к. сервер вполне в состоянии сам корректно распознать тип принимаемого файла.

Атрибут **name** определяет уникальное имя формы. На одной Web-странице может находиться несколько форм. В этом случае значения атрибутов **name** у них не должны совпадать.

Вопросы для контроля

1 Для чего предназначены формы в html?

2 Перечислите известные параметры тега `<form>`. Для чего они предназначены?

3 Какие значения можно передать в атрибут `type`?

4 Как можно назвать элементы, значение атрибута `type` которых равно `checkbox`?

5 Какой элемент управления называется зависимым переключателем?

6 С помощью какого атрибута в теге `<select>` можно позволить пользователю выбирать несколько значений сразу?

Лабораторная работа № 7. CSS. Основные понятия

Цель работы: изучить порядок создания и использования CSS при разработке Web-страниц.

Порядок выполнения работы

1 Ознакомиться с основами применения CSS.

2 Оформить лабораторные работы № 2–4 с использованием CSS.

3 Использовать ***все типы*** селекторов и ***все способы*** подключения CSS.

4 Сделать вывод о размере кода HTML с применением CSS, а также о целесообразности применения CSS в конкретном случае.

Основные теоретические положения

CSS (Cascading Style Sheets, Каскадные Таблицы Стилей) – это набор правил форматирования Web-страницы, который может быть применен к различным элементам страницы.

В HTML для присвоения какому-либо элементу определенных свойств, таких как цвет, размер, положение на странице и т. п., приходится каждый раз описывать эти свойства.

Применяя CSS, можно один раз описать свойства и их значения и определить это описание как *стиль*, а в дальнейшем просто указывать, что элемент,

который надо оформить соответствующим образом, должен принять эти свойства стиля.

Описание стиля можно сохранить не в тексте страницы, а в отдельном файле – это позволит использовать описание стиля на любом количестве Web-страниц.

Описания стилей в Web-странице должны находиться в тегах **<style></style>**, которые размещаются между тегами **<head></head>**.

`Example.css` – это CSS файл, содержащий описание применяемых стилей. Если он находится в другом каталоге, нужно указать к нему путь. Создается CSS файл в любом текстовом редакторе, например, в Блокноте, нужно только изменить расширение текстового файла на CSS. В CSS файле не должны указываться теги **<style></style>**.

Можно определить стиль для любого тега отдельно. Для этого нужно в тег добавить атрибут **style=""** и описать его стиль в кавычках.

Следующий пример отображает слово «Пример» шрифтом Verdana, размером 150 процентов и красным цветом.

```
<h3 style="font-family:Verdana; font-size:150%; color:red">Пример</h3>
```

Некоторые свойства CSS и их назначение показаны в таблице 3.

Таблица 3 – Свойства CSS

Свойства CSS	Назначение
Свойства шрифта	
`font-family`	Используется для указания шрифта или шрифтового семейства, которым будет отображаться элемент. Пример: `p {font-family: Verdana, sans-serif;}`
`font-weight`	Определяет степень насыщенности шрифта: bold|bolder|lighter|normal|100|200|300|400|500|600|700|800|900 Пример: `b {font-weight: bolder;}`

`font-size`	Устанавливает размер шрифта. Параметр может указываться в процентах, пикселях или сантиметрах. Примеры использования для тегов `h1`, `h2`, `h3`:
	`h1 {font-size: 200%;}`
	`h2 {font-size: 150px;}`
	`h3 {font-size: 400pt;}`

Свойства текста	
`text-decoration`	Устанавливает эффекты оформления шрифта, такие как подчеркивание или зачеркивание текста. Пример использования для тега `h4`: `h4 {text-decoration: underline;}` (подчеркивание) `h4 {text-decoration: line-through;}` (зачеркивание)

Окончание таблицы 3

Свойства CSS	Назначение
text-align	Определяет выравнивание элемента. Пример: p {text-align: left} (выравнивание по левому краю) p {text-align: right} (выравнивание по правому краю) p {text-align: justify} (выравнивание по ширине) p {text-align: center} (выравнивание по центру)
text-indent	Устанавливает отступ первой строки текста. Чаще всего используется для создания параграфов с табулированной первой строкой. Пример использования: h1 {text-indent: 60pt;}
line-height	Управляет интервалами между строками текста. Пример: p {line-height: 50 %}
Цвет	
color	Определяет цвет элемента. Пример использования для тега h3: h3 {color: #0000FF;}
background-color	Устанавливает цвет фона для элемента. Пример использования для тега h3: Пример
Свойства границ	
margin-left margin-right margin-top margin-bottom	Устанавливают значения отступов вокруг элемента. Пример использования для рисунка: img {margin-left: 20pt} img {margin-right: 20pt} img {margin-top: 20pt} img {margin-bottom: 20pt}

Вопросы для контроля

1 Какова структура правила CSS?
2 Что такое стиль и таблица стилей?
3 Способы подключения таблиц стилей CSS.
4 Какие существуют типы селекторов?
5 Для чего используется группирование в CSS?
6 У какого селектора комбинатор #?

Лабораторная работа № 8. CSS. Каскадирование

Цель работы: изучить механизм каскадирования в CSS.

Порядок выполнения работы

1 Подготовить Web-документ(-ты) (за основу можно взять предыдущие работы), при форматировании которых применяется механизм каскадирования.

2 В применяемых CSS механизм каскадирования пояснить комментариями.

Основные теоретические положения

В соответствии со спецификацией CSS имеется несколько способов подключения таблиц стилей к документу. И может получиться, что возникнет конфликт – на одно свойство конкретного элемента претендуют несколько разных значений, пришедших из разных правил (таблиц стилей).

Для разрешения конфликтов значений свойств в CSS определен механизм (порядок, алгоритм) каскадирования:

– по важности (явной приоритетности);
– по источнику;
– по специфичности;
– по расположению.

1 Важность (явная приоритетность).

Важность объявления настолько велика, что перевешивает все остальные факторы. CSS2.1 называет их (по вполне понятным причинам) *важными объявлениями (important declarations)* и предоставляет возможность отмечать их путем введения в объявление ключевого слова **!important** прямо перед завершающей точкой с запятой:

```
p.dark {color: #333 !important; background: white;}
```

Здесь значение цвета **#333** отмечено как **!important**, тогда как цвет фона **white** – нет. Если надо сделать важными оба объявления, каждому из них понадобится собственный маркер **!important**:

```
p.dark {color: #333 !important;
  background: white !important;}
```

Необходимо правильно размещать **!important**, иначе объявление будет признано недействительным. Ключевое слово **!important** всегда располагается *в конце объявления*, прямо перед точкой с запятой. Это особенно важно, когда дело доходит до свойств (например, **font**), значения которых могут состоять из нескольких ключевых слов:

```
p.light (color: yellow;
  font: smaller Times,
  serif !important;}
```

Если бы **!important** было расположено где-либо в другом месте объявления **font**, то все объявление было бы признано недействительным и ни один из его стилей не был бы применен.

Объявления, отмеченные как **!important**, не имеют особого значения специфичности, они рассматриваются отдельно от остальных. Фактически все объявления **!important** группируются вместе, и тогда уже их конфликты специфичностей разрешаются относительно друг друга. Аналогично группируются все остальные объявления, и конфликты свойств разрешаются с помощью специфичностей.

2 Источник правила.

Существует *три* возможных источника правил: ***автор***, ***читатель*** и ***браузер*** пользователя. Причем на их приоритетность влияет **инструкция !important**. Таким образом, с точки зрения *приоритетности* объявлений выделяют *пять* уровней. В порядке ***уменьшения приоритетности*** это следующие уровни.

1 Важные объявления пользователя.
2 Важные объявления автора.
3 Обычные объявления автора.
4 Обычные объявления пользователя.
5 Объявления браузера пользователя.

3 Специфичность.

Для каждого правила браузер пользователя вычисляет специфичность *(specificity) селектора* и прикрепляет ее к каждому *объявлению* правила.

Специфичность селектора определяется компонентами селектора.

Значение специфичности состоит из четырех частей ***a***, ***b***, ***c*** и ***d***: `0,0,0,0`.

Если стиль встроенный (*inline*), ***a*** = 1.

Значение ***b*** равно общему количеству селекторов идентификаторов.

Значение ***c*** равно количеству классов, псевдоклассов и селекторов атрибутов.

Значение ***d*** равно количеству селекторов типов и псевдоэлементов.

4 Порядок расположения.

Если два правила имеют совершенно одинаковые *приоритетность* и *специфичность*, то «побеждает» то из них, которое расположено в таблице стилей *позже*.

С этих позиций принимается, что стили, определенные в атрибуте **style** элемента, находятся в самом конце таблицы стилей документа, т. е. размещаются после всех остальных правил и поэтому имеют более высокую *специфичность*, чем любой селектор таблицы стилей.

5 Упрощенный порядок каскадирования.

Упрощенный порядок определения приоритетность правил (от низшего к высшему):

1) связанная таблица стилей (элемент <link>);
2) импортируемая таблица стилей (инструкция @import);
3) правило с элементом HTML в качестве селектора (элемент <style>);
4) правило с параметром class в качестве селектора;
5) правило с параметром id в качестве селектора;
6) встроенное в тэг HTML правило (атрибут style);

Вопросы для контроля

1 Когда применяется механизм каскадирования?
2 Как задать явную приоритетность?
3 По какому компоненту правила вычисляется специфичность?
4 Какие существуют источники правил?
5 Как подключить пользовательский стиль в браузере?
6 Назовите упрощенный порядок каскадирования от низшего к высшему.

Лабораторная работа № 9. CSS. Модель визуального форматирования

Цель работы: изучить возможности модели визуального форматирования в CSS.

Порядок выполнения работы

1 Ознакомиться с основами применения блочной модели в CSS.
2 Выполнить задание, полученное у преподавателя.
3 При этом использовать:
– максимальное количество CSS-свойств форматирования текста;
– различные типы и комбинации селекторов;
– возможности свойства display для отображения элементов.
4 Оформить отчет в виде html-файла.

Основные теоретические положения

1 Типы отображения элементов.

Css-свойство display задаёт тип отображения элемента. На русский язык его можно перевести фразой: «Веди себя как». То есть берём любой элемент html и говорим ему: веди себя как… блок, строка, таблица или вообще как будто тебя нет.

Наиболее часто используемые значения свойства display:

– none;
– block;
– inline;

– inline-block;
– list-item;

Свойство **display** изменяет поведение элемента, но не классовую принадлежность.

2 Замещаемые и незамещаемые элементы.

CSS определяется элементами, но не все элементы создаются одинаково. Например, изображения и абзацы – это элементы разных типов, так же как и . В CSS элементы разделяются на *замещаемые* и *незамещаемые*.

Замещаемыми (replaced) называются те элементы, содержимое которых замещается чем-то, что не содержится непосредственно в документе. Наиболее очевидный пример из HTML – элемент img, замещаемый файлом изображения, который является внешним по отношению к документу. Он фактически не имеет содержимого, как видно из простого примера:

```
<img src="howdy.gif">
```

Этот фрагмент разметки не имеет реального содержимого, а только имя элемента и атрибут. Данный элемент ничего не представляет, пока нет указания на внешнее содержимое (в данном случае изображение, заданное атрибутом src).

Элемент input замещается кнопкой, переключателем или полем ввода текста в зависимости от его типа. Замещаемые элементы также генерируют блоки в своем визуальном представлении.

Основная масса элементов HTML – *незамещаемые* (nonreplaceable). Например, элемент эй там – незамещаемый элемент, и агент пользователя (user agent) будет отображать текст «эй там». Это выполняется для абзацев, заголовков, ячеек таблиц, списков и почти всех остальных элементов HTML.

3 Схлопывание (сворачивание) вертикальных полей (margin).

Есть две ситуации в верстке, когда вертикальные поля схлопываются:

1) соседство двух элементов с вертикальными полями;

2) наличие вертикальных полей у родительского и дочернего элемента.

Внешне такой эффект выглядит так, будто поля накладываются друг на друга.

4 Строчные (внутристрочные, inline) элементы.

Строчными называются элементы web-страницы, которые являются непосредственно частью строки. К строчным элементам относятся элементы <img>, , <a>, <q>, <code> и другие. В основном они используются для изменения вида текста.

Характерные особенности строчных элементов:

– внутрь строчных элементов допустимо помещать текст или другие строчные элементы. Вставлять блочные элементы внутрь строчных ***запрещено***;

– эффект схлопывания полей не действует. Если рядом стоят два строчных элемента с определенными полями, то эти поля ***суммируются***;

– свойства, связанные с размерами (width, height), ***не применимы***. Их просто нет. На то она и строка;

– ширина элемента равна содержимому плюс значения отступов, полей и границ;

– несколько строчных элементов идущих подряд располагаются на одной строке друг за другом и переносятся на следующую строку при необходимости;

– строчные элементы можно выравнивать по вертикали при помощи css свойства `vertical-align`.

Вопросы для контроля

1 Какие существуют концепции визуального форматирования?
2 В чем отличие замещаемого элемента от незамещаемого?
3 Какие есть типы отображения элементов?
4 Что нужно сделать, чтобы не происходило схлопывание полей?
5 Как сделать элемент невидимым?
6 Можно ли в строчные элементы вкладывать блочные? А наоборот?

Лабораторная работа № 10. CSS. Позиционирование и свободное перемещение

Цель работы: изучить возможности основных типов позиционирования в CSS.

Порядок выполнения работы

1 Ознакомиться с основами позиционирования в CSS.
2 Выполнить задание, полученное у преподавателя.
3 Использовать все типы позиционирования.
4 Оформить отчет в виде HTML и CSS файлов.

Основные теоретические положения

Сущность позиционирования состоит в том, чтобы любой элемент расположить в необходимом месте web-страницы. Позиционирование определяется свойством `position`, которое может иметь следующие значения:

`position: static;` (значение по умолчанию для всех элементов в нормальном потоке)

```
position: absolute;
position: fixed;
position: relative;
```

Указание точного месторасположения элементов осуществляется ***свойствами смещения***: `top`, `right`, `bottom`, `left`.

`top` – расстояние от верхнего края окна блока-контейнера (или браузера).

`bottom` – расстояние от нижнего края окна блока-контейнера (или браузера).

`left` – расстояние от левого края окна блока-контейнера (или браузера).

`right` – расстояние от правого края окна блока-контейнера (или браузера).

Свойства смещения работают со всеми позиционированными элементами, кроме имеющих значение `static`.

Прежде чем приступать к работе с позиционированием, необходимо разобраться с таким понятием, как *нормальный поток*.

Нормальный поток:

1) *текста* (*контента*) – это размещение символов слева направо и сверху вниз (для западных языков);

2) *порядок вывода элементов* на экран определяется порядком, в котором они написаны в html-коде документа;

3) *блочные элементы* (`display: block`):

– выводятся сверху вниз по странице;

– занимают всю доступную ширину родительского элемента;

– всегда начинаются с новой строки;

– могут содержать *блочные* и *строковые* элементы. К блочным элементам относятся `<div>`, `<h1>`, `<p>` и др.

Высота элемента определяется его содержимым.

4) *строковые элементы* (`display: inline`):

– выводятся слева направо по строке;

– располагаются друг за другом в одной строке;

– могут содержать *только строковые* элементы.

При необходимости строка переносится. До и после строчного элемента отсутствуют переносы строки. Ширина и высота строчного элемента зависит только от его содержания, задать размеры с помощью CSS нельзя.

Абсолютное позиционирование

Абсолютное позиционирование задаётся при установке свойства элемента «`position: absolute`».

При абсолютном позиционировании элемент:

– выводится из общего потока;

– становится независимым от других элементов;

– может накладываться на другие элементы.

При абсолютном позиционировании положение элемента задаётся только свойствами смещения этого элемента (`bottom`, `left`, `right`, `top`) относительно блока-контейнера.

Блоком-контейнером считается ближайший предок (любой), значение свойства `position` которого отлично от `static`. Если таких предков нет, то блоком-контейнером считается начальный блок-контейнер.

Важные моменты:

– абсолютно позиционированные элементы перемещаются вместе с документом при прокрутке;

– свойства `left` и `top` имеют приоритет выше, чем свойства `right` и `bottom`.

Например, если свойства `left` и `right` противоречат друг другу, то свойство `right` будет проигнорировано. Это же относится и к свойствам `top/bottom`;

– элементы можно спрятать. Для этого можно задать отрицательным либо `left`, либо `top`. Элемент выйдет за границы окна браузера, полоса прокрутки при этом не возникнет;

– если задать `left` больше, чем ширина окна браузера либо отрицательное значение `right`, то возникнет горизонтальная полоса прокрутки. Аналогично будет и с `top`, только полоса прокрутки будет вертикальной.

Фиксированное позиционирование

Фиксированное позиционирование задаётся «`position:fixed`» и по своей сути очень похоже на абсолютное позиционирование. Элемент с фиксированным позиционированием также выводится из общего потока и не зависит от расположения других элементов. Положение элемента **не** изменяется при прокрутке. Элемент так же, как и при абсолютном позиционировании, может накладываться на другие. Ещё одной особенностью фиксированного позиционирования является то, что выходе содержимого за пределы видимой области **не** возникает полос прокрутки.

Фиксированное позиционирование используется для создания меню, вкладок, заголовков, т. е. таких элементов, которые всегда должны быть видны пользователю.

Относительное позиционирование

Относительное позиционирование задается свойством «**`position: relative`**». При таком способе позиционирования положение элемента определяется относительно краёв элемента родителя и сам элемент не выводится из основного потока. Расстояние до краёв родительского элемента задаётся свойствами: `bottom`, `left`, `right`, `top`.

Важные моменты:

– данный тип позиционирования не применяется к элементам таблицы;

– если при смещении элемента образовалось пустое место, оно не занимается ниже- или вышележащими элементами;

– относительно позиционированный элемент образует блок-контейнер для других элементов.

Плавающие элементы

Плавающее поведение элемента осуществляется заданием свойства `float`. Оно применятся для создания красивого эффекта обтекания какого-либо элемента содержимым. Для этого закрепляется одна сторона, позволяя другим элементам располагаться вокруг.

Возможные значения свойства `float`:

`none` – без обтекания;

`left` – выравнивание по левому краю, обтекание по правому краю;

`right` – выравнивание по правому краю, обтекание по левому краю.

Свойство z-index

Свойство `z-index` предназначено для работы с элементами страницы в трехмерном пространстве. Для этого вводится третья ось – ось Z. Так как страница видится как двумерная, элементы накладываются слой за слоем друг на друга. Управлять подобным наложением можно с помощью свойства `z-index`.

Возможные значения свойства `z-index`:

`auto` – элементы накладываются друг на друга в том порядке, в каком они были указаны в коде HTML;

`целое число` – чем больше число, тем более высокую позицию он занимает по оси Z и, соответственно, перекрывает все элементы, расположенные ниже по этой оси.

Важные моменты:

– числовое значение `z-index` может быть отрицательным;

– при равном значении `z-index` на переднем плане будет находиться тот элемент, который идёт ниже в html-коде. Это же правило действует и при `z-index`, равном `auto`.

Вопросы для контроля

1 В чем сущность позиционирования? Каким свойством оно определяется?

2 Какие свойства являются свойствами смещения?

3 Какие базовые схемы размещения элементов есть в CSS?

4 Что такое нормальный поток?

5 В чем отличие абсолютного позиционирования от относительного позиционирования?

6 Каким свойством задаются плавающие элементы?

Лабораторная работа № 11. Создание XML-документа и форматирование CSS

Цель работы:
– изучить структуру XML-документа;
– изучить возможности форматирования XML-документа CSS.

Порядок выполнения работы

1 Ознакомиться с основами синтаксиса языка XML и построения XML-документа.
2 Выполнить задание, полученное у преподавателя.
3 Оформить отчет в виде XML- и CSS- файлов.

Основные теоретические положения

XML (*e**X**tensible **M**arkup **L**anguage*, расширяемый язык разметки).

Особенности XML:
– XML является языком разметки так же, как и HTML;
– XML создан для описания данных;
– в XML автор сам определяет собственные тэги;
– для описания данных в XML используется DTD (Document Type Definition – Определение типа документа) или XML-Shema.

Основные различия между XML и HTML:
– XML создавался для описания данных и фокусируется на том, чем являются данные;
– HTML создавался для демонстрации данных и фокусируется на том, как данные выглядят;
– HTML предназначен для показа информации, а XML – для описания информации.

1 Структура XML-документа.

XML-документ – последовательность символов, в которой ни одна из групп этих символов (в т. ч. и вся последовательность целиком) не используется отдельно от какого-либо обозначения (то есть никак не обозначенные наборы отсутствуют), причем для каждого такого набора могут быть заданы правила его интерпретации.

Документы XML включают:
– разметку;
– текстовые данные.

Разметка:
– может начинаться и заканчивается символами < и >;
– может обозначаться символом **&,** а конец – знаком ";" ;
– может начинаться с символа **%** и заканчиваться символом " ;" ;
– образует структуру документа и состоит из:
а) начальных и конечных тегов;

б) тегов пустых элементов;
в) определений DTD;
г) объектных и текстовых ссылок (сущностей);
д) комментариев;
е) секции CDATA;
и) инструкций по обработке.

Текстовые данные – текст (содержимое, content) в XML-документе, который не относится к категории разметки.

2 Синтаксис XML.

1 Все теги парные.

```
<книга> Текст параграфа. </книга >
```

Открывающий тег должен иметь соответствующий закрывающий тег. Все элементы должны быть закрыты.

2 Пустой элемент (его тег) должен быть закрыт:

```
<br />
```

3 Элементы (их теги) не могут пересекать друг друга.
Все элементы должны быть соответствующим образом вложены:

```
<parent ont = "12">
<child> text </child>
    </parent>
```

4 XML-документ может иметь только ***один*** корневой элемент (**`root element`**) (даже так **`<docum />`**).

5 Тег может иметь сколько угодно атрибутов.

6 Все значения атрибутов должны заключаться в кавычки.

7 Имена *элементов* и *атрибутов* должны подчиняться соглашениям XML о названиях:

– регистрозависимы

Тег **`<Book>`** отличается от тега **`<book>`**;

– имя должно начинаться с буквы или с символа подчеркивания (**`_`**);
– имена не могут начинаться:
а) с цифры или знака препинания;
б) с букв **`xml`** (или **`XML`**, или **`Xml`** и т. д.);
– в имени не должно быть пробелов;
– имена могут содержать буквы, цифры и символы: точки (**`.`**), тире (**`-`**), подчеркивания (**`_`**), двоеточие (**`:`**).

8 Для специальных (служебных) символов (**<**, **>**, **&**, **%** и др.) должны использоваться сущности.

9 XML сохраняет *пробелы* внутри текста.

3 Связывание XML-документа с таблицей CSS.

Чтобы связать таблицу каскадных стилей с XML-документом, надо вставить в документ зарезервированную инструкцию по обработке `xml-stylesheet`. Эта инструкция по обработке имеет следующую обобщенную форму записи, где `CSSFilePath` есть путь, задающий местонахождение файла таблицы стилей:

```
<?xml-stylesheet type="text/css" href=CSSFilePath?>
```

Чаще используется частичный URL, который задает местонахождение относительно местонахождения XML-документа, содержащего инструкцию по обработке `xml-stylesheet` , например:

```
<?xml-stylesheet type="text/css" href="Inventory.css"?>
```

Относительный URL используется чаще, поскольку обычно файл таблицы стилей хранится в той же папке, что и XML-документ, либо в одной из вложенных папок.

Вопросы для контроля

1 Назначение XML.
2 Что входит в структуру XML-документа?
3 Что включает разметка XML-документа?
4 Требования синтаксиса XML.
5 Как связать XML-документ с таблицей CSS?
6 Что такое настраиваемые языки разметки?

Лабораторная работа № 12. Разработка DTD и действительного XML-документа

Цель работы: изучить структуру и синтаксис DTD XML-документа.

Порядок выполнения работы

1 Ознакомиться с основами построения DTD XML-документа.
2 Выполнить задание, полученное у преподавателя.
3 В DTD необходимо использовать все типы элементов, типы атрибутов, значений атрибутов по умолчанию и сущностей.
4 В работе должны присутствовать внутреннее и внешнее DTD.
5 Оформить отчет в виде XML-документа.

Основные теоретические положения

1 Уровни соответствия.

Согласно спецификации языка XML 1.0, объект данных именуется ***XML-документом***, если является *формально корректным (well formed – хорошо оформленным, правильным*), т. е. имеет синтаксис, соответствующий стандарту (спецификации).

XML-документ считается *действительным* (*valid – валидным, состоятельным*), если это условие выполняется, а также документ имеет в *своем составе* описание или *ссылку* на описание, определяющее типы используемых данных и возможную структуру документа, причем сам документ *не противоречит* этому описанию.

Такое описание задает корректный синтаксис и может быть реализовано в виде:

– DTD (Document Type Definition) – определение типа документа;

– XML Schema (XML-схема);

– Relag NG.

XML Schema и Relag NG сами являются XML-документами и должны располагаться в отдельных файлах.

Оба эти формата (DTD и XML Schema) описывают, какие **элементы** должен содержать в себе XML-документ, какими **атрибутами** могут обладать эти самые элементы и какого **типа** должны быть значения атрибутов и элементов.

Программы, читающие и анализирующие XML-документы, часто называют *парсерами – синтаксическими анализаторами* (от англ. *parse* – анализировать, обрабатывать). Современные браузеры включают в себя парсеры XML-документов, отвечающие стандартам W3C.

Если при чтении XML-файла браузер обнаружил ошибку, в окне появляется сообщение о найденной ошибке и содержимое документа не отображается.

XML-спецификация консорциума W3C указывает, что программа не должна продолжать обработку XML-документа, если она обнаружит ошибку. Это сделано для того, чтобы было легко создавать программное обеспечение для XML и чтобы все XML-документы были совместимы.

Для *действительных (*состоятельных) документов анализатор, кроме разбора тегов, может проверить правильность их вложения друг в друга, корректность следования, наличие обязательных атрибутов тега, допустимость их значений и т. п.

Согласно рекомендациям консорциума W3C (World Wide Web Consortium), наиболее важным ограничением является *формальная корректность*.

2 Типы DTD и способы их подключения.

Определение типа документа может размещаться внутри документа XML – в этом случае имеем *внутреннее* DTD – или может быть задано во внешнем файле – *внешнее* DTD.

Применительно к *внешним* DTD различают *системное* (system) и *общее* (public) определения типа документа. Системное DTD может использоваться одним или несколькими документами XML, общее DTD – в любом документе XML.

Для создания *внешнего* DTD следует поместить объявления всех размещаемых в нем элементов, атрибутов и информационных объектов в отдельном файле с расширением DTD, например `book.dtd`.

В состав внешнего DTD нужно поместить также инструкцию по обработке кода XML, включив строку:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Чтобы получить доступ к внешнему DTD, размещенному в сети, в XML-документ нужно включить следующую строку

```
<!DOCTYPE book SYSTEM "http://www.xom.com/dtds/book.dtd">
```

Здесь `http://www.xom.com/dtds/book.dtd` – адрес размещения файла DTD в сети.

Если DTD является общим, в приведенной строке вместо слова `SYSTEM` требуется указать `PUBLIC`.

Вопросы для контроля

1 Какие бывают уровни соответствия XML-документа?
2 Чем отличается *валидный* XML-документ от *формально корректного?*
3 Какие виды описаний, определяющих типы используемых данных и возможную структуру XML-документа ,применяются?
4 Что сделает парсер при обнаружении ошибки при разборе?
5 Какие бывают типы DTD?
6 Как подключаются внешние DTD?

Лабораторная работа № 13. Разработка составного XML-документа с использованием пространства имен (namespace)

Цель работы: изучить синтаксис и порядок применения пространств имен.

Порядок выполнения работы

1 Ознакомиться с основами синтаксиса и порядка применения пространств имен в XML-документах.
2 Выполнить задание, полученное у преподавателя.
3 Использовать все варианты подключения пространств имен.
4 Оформить отчет в виде XML-документа.

Основные теоретические положения

1 Конфликт имён.

Поскольку XML-документы могут быть *составными*, использовать несколько схем, спроектированных разными разработчиками, то возможна *проблема неоднозначности имен*. В одном XML-документе не исключено использование элементов и атрибутов с *одинаковыми* названиями, но несущими *разную* смысловую нагрузку. При объединении XML-документов, принадлежащих разным предметным областям, возможны *конфликты* имен.

Следующий код содержит описание бассейна (pool):

```
<pool>
    <length> 10m </length>
    <width> 6.5m </width>
    <depth> 2m </depth>
</pool>
```

А код другого документа содержит описание бильярдного стола (pool):

```
<pool>
    <length> 2.7m </length>
    <width> 1.35m </width>
    <woodtype> Фанера </woodtype>
</pool>
```

Если соединить эти два документа в один, то произойдет *конфликт имён*. Оба документа содержат элемент `<pool>`, но он имеет разное *значение*. XML-парсер не сможет *правильно* обработать такой элемент.

Для предотвращения таких конфликтов и применяются пространства имён.

2 Пространства имён. Атрибут xmlns.

Пространство имен XML – это идентифицируемая с помощью ссылки URI коллекция имен, используемых в XML-документах для обозначения типов элементов и именования атрибутов.

Оно *не определяет* набор доступных элементов и атрибутов. Использование пространств имен в XML позволяет связать имя элемента и атрибута с соответствующим пространством имен, тем самым исключить возможные коллизии имен и подсказать обработчику XML, как правильно интерпретировать XML документ. Пространство имён может быть задано с помощью атрибута `xmlns` в элементе, причем задаётся оно *только* в начальном элементе. Его декларация имеет следующий синтаксис:

```
<element xmlns ="URI"></element>
```

URI – уникальный идентификатор ресурса. По адресу ресурса не обязательно должно что-то находиться. Парсер *не будет* переходить по ссылке

и проверять наличие информации. Адрес лишь используется как *уникальный идентификатор*, чаще всего это адрес своего сайта.

3 Пространство имён по умолчанию.

Если пространство имён задают для элемента верхнего уровня без использования индекса, то оно называется пространством имён *по умолчанию*. Все вложенные элементы будут использовать пространство имён элемента верхнего уровня. Всего можно задать только одно пространство имён по умолчанию для конкретного элемента. В примере ниже показано его использование.

```
<pool xmlns="http://www.mywebsite.org/outside-
    structure/swimmingpool/">
  <length> 10m </length>
  <width> 6.5m </width>
  <depth> 2m </depth>
</pool>
<pool xmlns="http://www.mywebsite.org/fur/pooltbl/">
  <length> 2.7m </length>
  <width> 1.35m </width>
  <woodtype> Фанера </woodtype>
</pool>
```

Здесь пространства имён по умолчанию заданы для элементов **<pool>**. Все элементы, вложенные в свой элемент **<pool>**, будут разделять его пространство имён. Использование пространства имён по умолчанию позволяет избегать использования префиксов.

4 Пространства имён с использованием префиксов.

Конфликта одинаковых имён можно избежать, используя префикс.

```
<sp:pool>
  <sp:length> 10m </sp:length>
  <sp:width> 6.5m </sp:width>
  <sp:depth> 2m </sp:depth>
</sp:pool>
<pt:pool>
  <pt:length> 2.7m </pt:length>
  <pt:width> 1.35m </pt:width>
  <pt:woodtype> Фанера </pt:woodtype>
</pt:pool>
```

Здесь конфликта имён не произойдет, т. к. у элемента **<pool>** разные имена. Перед названием каждого элемента поставлены префиксы **sp** для первой части документа и **pt** – для второй.

Префиксы следует применять *только* с заданным пространством имён.

Синтаксис декларации пространства имён с использованием префикса следующий:

```
<element xmlns:prefix="URI"></element>
```

Пример декларации пространства имён с использованием префиксов:

```
<sp:pool xmlns:sp="http://www.mywebsite.org/outside-
    structure/swimmingpool/">
  <sp:length> 10m </sp:length>
  <sp:width> 6.5m </sp:width>
  <sp:depth> 2m </sp:depth>
</sp:pool>
<pt:pool
    xmlns:pt="http://www.mywebsite.org/fur/pooltbl/">
  <pt:length> 2.7m </pt:length>
  <pt:width> 1.35m </pt:width>
  <pt:woodtype> Фанера </pt:woodtype>
</pt:pool>
```

В примере выше связка вида **prefix:name** называется *квалифицированным именем* (qualified name). Квалифицированное имя содержит единственный символ двоеточия, который делит имя на префикс пространства имён и локальную часть.

На имя префикса пространства имён накладываются следующие ограничения:

– префикс не может иметь значение **xml**, т. к. оно зарезервировано за префиксом, привязанным к имени пространства имён http://www.w3.org/XML/1998/namespace;

– имя префикса не может начинаться с последовательности *x, m, l* (независимо от регистра*)*;

– префикс не может иметь значение `xmlns`, т. к. это имя зарезервировано за атрибутом объявления пространства имён.

Следует отметить, что если элемент имеет префикс, то закрываться элемент должен тоже с префиксом, ведь префикс – часть имени элемента. То есть код вида `<pt:length> 2.7m </length>` будет *ошибочным*.

Если для вложенного элемента не задан префикс пространства имён, то он будет использовать пространство имён по умолчанию, заданное в элементе высшего уровня.

5 Область действия пространства имён.

Область действия пространства имен может распространяться:

– на *весь документ*, когда оно объявлено в корневом элементе документа;

– на *конкретный контейнер (элемент)* и *все* включенные в него элементы и атрибуты, когда оно объявлено в элементе верхнего уровня контейнера;

– на *отдельный элемент и его атрибуты*, когда оно объявлено непосредственно в данном элементе.

У пространств имён есть свой недостаток: описание DTD станет невозможным при использовании нескольких пространств имен. Поэтому для валидации XML-документа, содержащего пространства имён используется XML Schema.

Вопросы для контроля

1 Зачем нужны пространства имён?
2 Что понимается под пространством имен XML?
3 Как задать пространство имён *по умолчанию*?
4 Каковы ограничения на имя префикса пространства имён?
5 Что такое *квалифицированное имя* (qualified name)?
6 Чем определяется область действия пространств имён?

Лабораторная работа № 14. Расширяемый язык таблиц стилей XSL

Цель работы: изучить синтаксис и порядок применения расширяемого языка таблиц стилей XSL.

Порядок выполнения работы

1 Ознакомиться с основами синтаксиса и порядком применения расширяемого языка таблиц стилей XSL.
2 Выполнить задание, полученное у преподавателя.
3 Создать XSL-файл и применить его к XML-документу, полученному в предыдущей лабораторной работе.
4 Оформить отчет в виде XML-документа и XSL-файла.

Основные теоретические положения

В самом XML-документе нет никаких указаний на то, как его следует отображать. Значит, надо разработать правила преобразования XML-документа к удобочитаемому виду. Для HTML-документов существуют стилевые таблицы CSS, а для XML-документов – язык XSL.

XSL (extensible Stylesheet Language) – расширяемый язык стилей, применяется для преобразования и форматирования XML-документов.

XSL состоит из трех языков;

1) XSLT (eXtensible Stylesheet Language Transformations) – язык преобразований XML-документов;

XSLT – язык преобразования XML-документа в другие документы. Такое преобразование позволяет выделить нужную часть информации из документа и представить ее в удобном для чтения виде;

2) XPath – язык определения частей и путей к элементам XML;

3) XSL Formatting Objects – язык определения формата показа XML.

XSLT является наиболее важной частью стандарта XSL. Именно эта часть XSL применяется для того, чтобы преобразовывать XML-документ в другой XML-документ или в другие типы.

XSLT позволяет:

– преобразовывать XML-документ в формат, знакомый браузерам;

– добавлять совершенно новые элементы в выходной документ или удалять элементы;

– изменять порядок следования элементов;

– производить проверку и на ее основе решать, какие элементы показывать;

– хранить в одном формате, выводить в другом;

– делать документ более компактным;

– использовать документ как интерфейс для запроса к базам данных.

С точки зрения XSLT существует входной документ – XML-документ, подлежащий преобразованию, и выходной документ – результат преобразования.

В процессе преобразования XSLT использует XPath для определения тех частей во входном документе, которые соответствуют одному или нескольким заранее заданным шаблонам. Когда такое совпадение обнаруживается, XSLT преобразует соответствующую часть входного документа в часть выходного документа.

Те части входного документа, которые не соответствуют шаблону, передаются в выходной документ без изменений.

Связывание XSL-таблицы стилей с XML-документом

Существуют *два основных шага* для отображения XML-документа при использовании XSL-таблицы стилей:

1) ***создание*** файла XSL-таблицы стилей. XSL является приложением XML, т.е. XSL-таблица представляет собой корректно сформированный XML-документ, который отвечает правилам XSL;

2) ***связывание*** XSL-таблицы стилей с XML-документом. В XML-документ включается инструкция по обработке xml-stylesheet, которая имеет следующую форму записи:

```
<?xml-stylesheet type="text/xsl" href=xslFileURL?>
```

Здесь xslFileURL – URL файла XSL-таблицы стилей.

Если используется полный (не относительный) URL, таблица стилей должна размещаться в том же домене, что и сам XML-документ.

Инструкция по обработке xml-stylesheet добавляется в пролог XML-документа вслед за объявлением XML.

Если *связывается* с XML-документом:

– *более одной* XSL-таблицы стилей – браузер использует *первую* таблицу и *игнорирует* все остальные;

– *одновременно* CSS-таблица и XSL-таблица стилей – браузер использует *только* XSL-таблицу стилей.

Минимальная таблица стилей XSL, в которой находится только корневой элемент и ничего больше, будет выглядеть следующим образом:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Таблица стилей XSL является XML-документом, поэтому начинается с XML-объявления.

Она может иметь объявление типа документа (DTD или другое), хотя у большинства таблиц стилей его нет.

Корневым элементом этого документа являются либо `stylesheet`, либо `transform`. Они служат не только хранилищем всех других элементов, но и идентифицируют документ как XSL-таблицу стилей.

Шаблон (pattern) XSL

Синтаксис шаблона `xsl:template`:

```
<xsl:template
  match = образец(XPath, pattern) || name = имя
  priority = число
  mode = режим >
  <!-- шаблон -->
  <!-- content:(<xsl:param>*,template) -->
</xsl:template>
```

Каждое правило шаблона (***template rules***, шаблон) выполняет три задачи:

1) находит (`match` или `name`) узел, в данном случае – тип элемента;

2) определяет структуру результирующего поддерева;

3) явно указывает, нужно ли обрабатывать ветви поддерева.

С помощью этих компонентов каждое поддерево исходного документа может быть обработано от корня и до листьев, в результате чего создается каскад деревьев, при склеивании которых формируется результирующее дерево.

Вопросы для контроля

1 Что такое XSL?
2 Состав XSL.
3 Как работает XSL-файл?
4 Синтаксис шаблона XSL.
5 Каковы задачи шаблона XSL?
6 Назначение атрибутов шаблона XSL.

Лабораторная работа № 15. XSLT. Применение функций

Цель работы: изучить синтаксис и порядок применения функций.

Порядок выполнения работы

1 Ознакомиться с основами синтаксиса и порядка применения функций в XSL-файлах.
2 Выполнить задание, полученное у преподавателя.
3 Использовать все варианты подключения пространств имен.
4 Оформить отчет в виде XML-документа.

Основные теоретические положения

Логические структуры в языке XSLT служат для обработки контента с условием. В зависимости от того, какое именно условие выполнено, выбирается та часть шаблона, которая будет участвовать в трансформации исходного дерева.

В XSLT существует всего *два вида* логических структур.

Первый представляет собой реализацию простейшей обработки с условием `xsl:if`. Если условие выполнено, то предпринимаются какие-либо действия, если нет, то не происходит ничего.

Второй вид обработки с условием представлен директивой `xsl:choose`, с помощью которой реализуется выбор одного из многих вариантов. Для отбора этих вариантов служат директивы `xsl:when` и `xsl:otherwise`, которые могут быть использованы только внутри контента `xsl:choose`.

Пример преобразований

Создаём документ XML с именем "`l.xml`"

```
<?xml-stylesheet type="text/xsl" href="lab3.xsl"?>
  <catalog>
    <cd>
      <название>HP</название>
      <процессор>IntelCorei7</процессор>
```

```
      <озу>8 GB</озу>
      <видеокарта>NVideoGForce 8G</видеокарта>
      <price>1000</price>
      <гарантия>2 года</гарантия>
    </cd>
    .
    .
  </catalog>
```

Создаём файл "ml.xsl"

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
  <h2>Главнаятаблица</h2>
  <table border="2">
    <tr bgcolor="skyblue">
      <th>Название</th>
      <th>Процессор</th>
      <th>ОЗУ</th>
      <th>Видеокарта</th>
      <th>Цена</th>
      <th>Гарантия</th>

    </tr>
    <xsl:for-each select="catalog/*">
    <tr>
      <td><xsl:value-of select="Название"/></td>
      <td><xsl:value-of select="Процессор"/></td>
      <td><xsl:value-of select="ОЗУ"/></td>
      <td><xsl:value-of select="Видеокарта"/></td>
      <td><xsl:value-of select="Цена"/></td>
      <td><xsl:value-of select="Гарантия"/></td>
    </tr>
    </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>
```

Результат представлен на рисунке 1.

Главная таблица

Название	Процессор	ОЗУ	Видеокарта	Цена	Гарантия
HP	Intel Core i7	8 GB	NVideo GForce 8G	1000	2 года
Samsung	Athlon g8	512 MB	NVideo	20000	5 лет
Dell	Intel Core i5	4 GB	Intel	1500	2 года

Рисунок 1 – Результат выполнения преобразований

Путь адресации

Чтобы делать в XSLT что-либо сложное, нужно свободно перемещаться по дереву документа. В любой момент требуется точно знать: *где* мы находимся и *куда* пойдем дальше.

Должна также иметься возможность с предельной точностью отбирать *группы узлов* для обработки.

Такие возможности навигации предоставляет XPath – язык для пометки точек и отбора групп узлов в документе.

Путь адресации – это такое выражение XPath, которое в результате обработки возвращает значение типа «набор узлов».

Путь адресации состоит из *одного* или *нескольких* шагов адресации *(steps)*, которые отделены друг от друга прямым слешем (/), каждый из которых уводит дальше от начальной точки.

Пути адресации могут быть записаны с помощью ***полного*** и ***сокращенного*** синтаксиса.

Сам шаг адресации *(steps)* образуют три части:

1) *ось (axis)* – описывает направление перемещения;

2) *критерий узла (node test, правило отбора)* – указывает, какие узлы являются пригодными (отбираются);

3) набор необязательных *предикатов (predicates)* – используют булевы значения (истина/ложь) проверки для дополнительного отсеивания кандидатов.

Вопросы для контроля

1 Порядок работы шаблона XSL.
2 Приведите пример логических функций в XSL.
3 Каков синтаксис применения логических функций в XSL?
4 Назначение языка XPath.
5 Что такое путь адресации?
6 Какие варианты синтаксиса пути адресации?

Лабораторная работа № 16. Преобразование XML-документов средствами XSL

Цель работы: изучить порядок преобразования XML-документов средствами XSL.

Порядок выполнения работы

1 Ознакомиться с основами преобразования XML-документов средствами XSL.

2 Выполнить задание, полученное у преподавателя.

3 Создать WindowsForm приложение, которое будет обеспечивать добавление и просмотр данных из XML-документа с использованием XML-сериализации.

4 Оформить отчет в виде XML-документа.

Основные теоретические положения

Сериализация в программировании – это процесс преобразования объекта в поток байтов. Затем получившийся поток сохраняется на диск для *долговременного* хранения либо помещается в оперативную память для *временного* хранения. Последовательность байтов содержит всю информацию, необходимую для восстановления (реконструкции, десерилизации) объекта с целью последующего использования.

Применяя технологию сериализации, возможно достаточно просто и быстро организовать *сохранение данных*; во многих случаях сериализация требует гораздо меньшего объема кода, чем процессы чтения/записи в текстовый или бинарный файл.

Если при использовании какого-либо приложения существует возможность сохранения и последующей загрузки пользовательских настроек, таких как цвет и размер шрифта, расположение окон, то с большой долей вероятности можно утверждать, что была применена сериализация.

Для выполнения сериализации /десериализации в технологии **.NET** существуют три основные возможности:

1) сериализация в двоичный формат (BinnaryFormatter);

2) сериализация в формат SOAP (SoapFormatter);

3) сериализация в формат XML (XmlSerializer).

Сериализации подлежат только открытые члены класса. Если в классе есть поля или свойства с модификатором private, то при сериализации они будут игнорироваться.

Сериализация невозможна для некоторых типов. Одним из таких является тип `Color`. Сохранение настроек цвета возможно в виде набора цифр с использованием формата записи sRGB.

Вопросы для контроля

1 Этапы преобразования XML-документа средствами XSL (сериализации).
2 Варианты выполнения сериализации/десериализации?
3 Какова структура XML-документа?

Список литературы

1 **Бройдо, В. Л.** Вычислительные системы, сети и телекоммуникации: учебник / В. Л. Бройдо. – 4-е изд. – Санкт-Петербург : Питер, 2013. – 560 с.

2 **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрейн. – Санкт-Петербург : Питер, 2014. – 304 с. : ил.

3 **Гоше, Х. Д.** HTML5. Для профессионалов : пер. с англ. / Х. Д. Гоше. – 2-е изд. – Санкт-Петербург : Питер, 2015. – 560 с. : ил.

4 **Дронов, В. А.** HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. А. Дронов. – Санкт-Петербург : БХВ-Петербург, 2011. – 414 с.

5 **Дакетт, Дж.** HTML и CSS. Разработка и дизайн веб-сайтов / Дж. Дакетт.– 2-е изд. – Санкт-Петербург : Эксмо, 2017. – 923 с.

6 **Дунаев, В. В.** HTML, скрипты и стили / В. В. Дунаев. – Санкт-Петербург: БХВ-Петербург, 2011. – 810 с.

7 **Евсеев, Д. А.** Web-дизайн в примерах и задачах: учебное пособие / Д. А. Евсеев, В. В. Трофимов; под ред. В. В. Трофимова. – Москва : КНОРУС, 2010 – 272 с.

8 **Макфарланд, Д.** Большая книга CSS /Д. Макфарланд. – 2-е изд. – Санкт-Петербург: Питер, 2012. – 560 с. : ил.

9 **Мак-Дональд, М.** HTML5. Недостающее руководство : пер. с англ. / М. Мак-Дональд. – Санкт-Петербург : БХВ-Петербург, 2012. – 480 с. : ил.

10 **Матросов, А. В.** HTML 4.0 / А. В. Матросов, А. О. Сергеев, М. П. Чаунин. – Санкт-Петербург: БХВ-Петербург, 2007. – 672 с.: ил.

11 **Мейер, Э.** CSS – каскадные таблицы стилей. Подробное руководство : пер. с англ. / Э. Мейер. – 3-е изд. – Санкт-Петербург: Символ-Плюс, 2010. – 576 с. : ил.

12 **Комолова, Н.** HTML, XHTML и CSS / Н. Комолова, Е. Яковлева. – Санкт-Петербург: Питер, 2012. – 304 с: ил.

13 **Немцова, Т. И.** Компьютерная графика и web-дизайн: учебное пособие / Т. И. Немцова, Т. В. Казанкова, А. В. Шнякин. – Москва: ФОРУМ; ИНФРА-М, 2014. – 400 с.

14 **Прохоренок, Н. А.** HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. А. Прохоренок. – Санкт-Петербург : БХВ-Петербург, 2010. – 900 с.

15 **Гарольд, Э.** XML: справочник : пер. с англ. / Э. Гарольд, С. Минс. – Санкт-Петербург: Символ-Плюс, 2002. – 576 с.: ил.

16 **Рэй, Э.** Изучаем XML : пер. с англ. / Э. Рэй. – Санкт-Петербург: Символ-Плюс, 2001. – 408 с.: ил.

17 **Старыгин, А. А.** XML: разработка Web-приложений / А. А. Старыгин. – Санкт-Петербург: БХВ-Петербург, 2003. – 592 с.: ил.

18 **Хабибуллин, И. Ш.** Самоучитель XML / И. Ш. Хабибуллин. – Санкт-Петербург: БХВ-Петербург, 2003. – 336 с.: ил.

19 **Холзнер, С.** XML: энцикл. / С. Холзнер. – 2-е изд. – Санкт-Петербург: Питер, 2004. – 1101 с.: ил.

20 **Шапошников, И. В.** Справочник Web-мастера. XML / И. В. Шапошников. – Санкт-Петербург: БХВ-Петербург, 2001. – 304 с.: ил.