№ 27–28 (364–365) 16–31 июля 2002

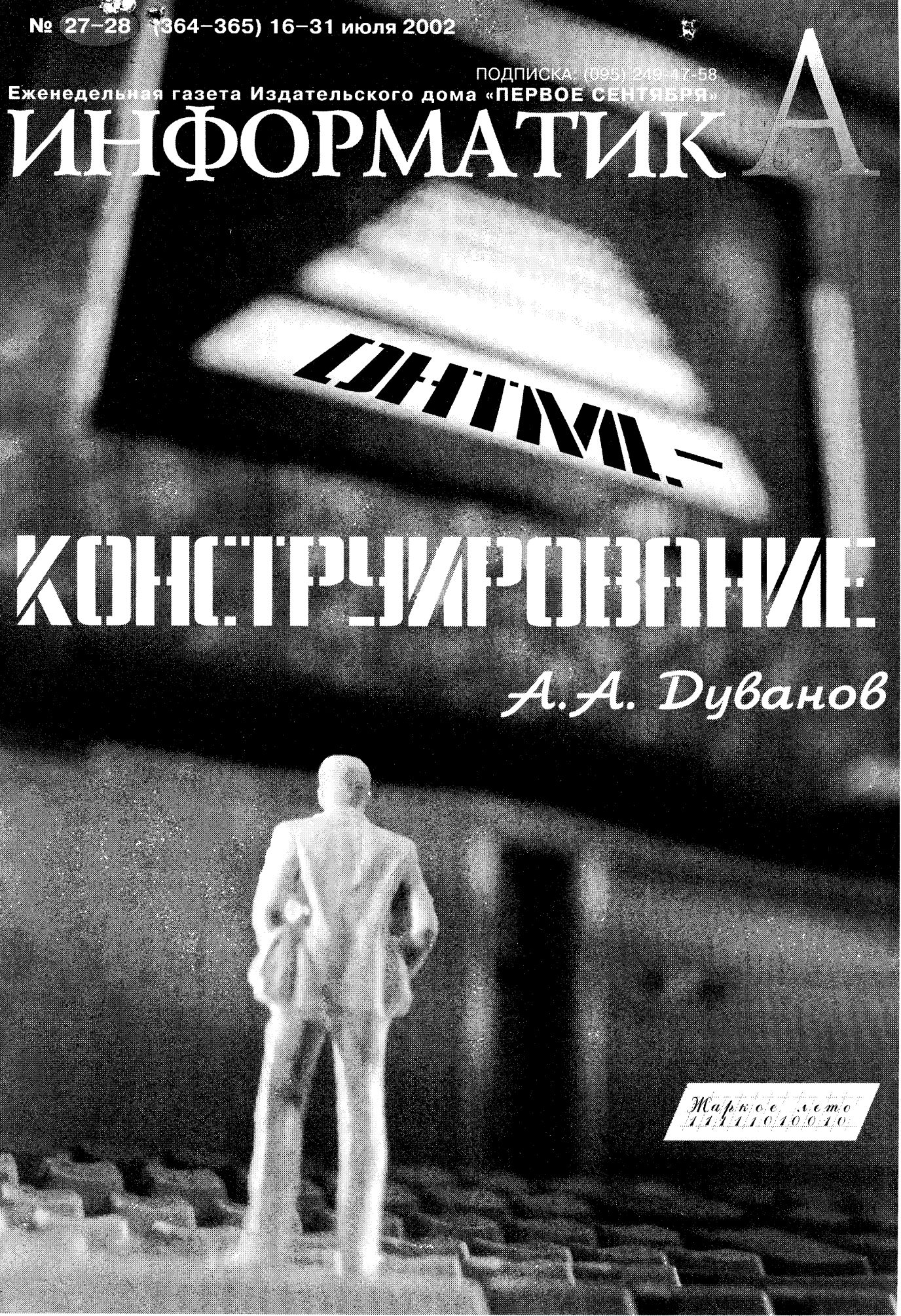
ПОДПИСКА: (095) 249-47-58

Еженедельная газета Издательского дома «ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИКА

ДНЕВНИК КОНСТРУИРОВАНИЯ

А.А. Дуванов



Жаркое лето
1111101001010

DHTML.RU • конструирование

РОБОТЛАНДСКИЙ УНИВЕРСИТЕТ © А.А.ДУВАНОВ

СОДЕРЖАНИЕ

Посвящается Олегу Какаулину в память о первых совместных работах.

Введение

"DHTML-конструирование" — это учебник Роботландского университета. В нем изложены основы CSS (*Cascading Style Sheets* — каскадные таблицы стилей) и показаны способы управления содержимым гипертекстовой страницы при помощи динамических воздействий на гипертекстовую модель документа скриптами, написанными на языке JavaScript.

Предполагается, что читатель знаком с основами ручной гипертекстовой разработки (книга "HTML-конструирование", № 21, 22/2000), а также умеет программировать на JavaScript (книга "JavaScript-конструирование", № 21, 25, 29, 33/2001).

Что такое DHTML

DHTML — это обозначение технологий для построения гипертекстовых приложений, практически не отличимых по своей динамике и интерактивности от обычных компьютерных программ.

Одна из технологий DHTML (которая и будет рассмотрена в книге) — это: DHTML = HTML 4.0 + + JavaScript + CSS + браузер 4.0 (и старше).

Версия HTML 4.0 кардинально не отличается от версии 3.2 по тегам (хотя есть и новинки). Самое главное в HTML 4.0 — **все** элементы страницы без исключения (графика, заголовки, таблицы, текст, все теги и их атрибуты) доступны для управления. То есть изменения больше касаются не языка, а объектной модели браузера.

Используя CSS и JavaScript, можно управлять любым элементом на странице, меняя его локально, не перерисовывая всю страницу целиком.

Технология CSS + JavaScript позволяет, например, применять абсолютное позиционирование, то есть выводить элементы на экран, используя реальные координаты. Программное изменение координат в объектной модели вызывает изменение положения элемента на экране.

CSS позволяет наряду с двумя координатами x и y использовать еще и третью координату — z-индекс. Третья координата определяет номер слоя, в который помещается элемент. Таким образом, при движении в многослойном пространстве одни элементы могут проходить над или под другими.

CSS позволяет вводить на странице графические фильтры, которые добавляют к графике и тексту эффекты мультимедиа (отражение, прозрачность, тени, движение пятен, проявление в "рентгеновских лучах").

Целевые установки

В гипертекстовом проектировании можно выделить два направления:

— сайты,

— локальные приложения.

Сайты "живут" в Интернете, локальные приложения — на компьютере индивидуального или в локальной сети коллективного пользователя (например, школьного класса).

Жанровые особенности определяются разницей в скорости передачи информации. Интернет работает очень медленно (средняя скорость передачи информации — несколько килобайт в секунду). На стадии становления глобальной сети скорость передачи росла каждый год, внушая оптимизм разработчикам и пользователям. Но сейчас, к сожалению, этот процесс прекратился, и новый виток разгона сети возможен только после кардинального пересмотра существующих способов передачи. Мир замер в ожидании Эйнштейна связи. Замер и принял на вооружение пессимистические правила, продиктованные суровой действительностью: сейчас хороший сайт — это простой сайт. Он должен:

— иметь посредственную малоразмерную графику,
— не использовать звук и видео,
— иметь минимум интерактивности и динамики.

Помимо проклятия скоростью, сайтостроение переживает тяжелое время анархии в среде разработчиков средств просмотра: каждый браузер требует особого программирования (даже на уровне разных версий, не говоря уже об уровне разных компаний) — никто не желает соблюдать стандарты.

Со временем, конечно, технические и политические трудности Интернета будут преодолены. Сейчас же те, кто не хочет ждать сложа руки, могут направить усилия в гипертекстовый жанр, отличный от сайтостроения, в котором можно накручивать любые скриптовые спирали, выкидывать любые стилевые коленца, грузить "толстые" звуковые треки, графику, видео без риска усыпить пользователя, ждущего окончания загрузки у монитора.

Речь идет о жанре гипертекстового приложения, примерами которого служат гипертекстовые учебники Роботландского университета. Такие гипертекстовые изделия живут на локальном компьютере или в локальной сети и, следовательно, не имеют "родовых" пятен ужасно медленного Интернета. Кроме того, они могут позволить себе работать только на одном браузере и игнорировать все остальные.

Хотя технология CSS и воздействие на объектную модель документа при помощи JavaSript являются универсальными, то есть годятся как для сайтостроения, так и для разработки локальных приложений, в книге "DHTML-конструирование" практическая часть имеет уклон в сторону локальных приложений с учетом отмеченных выше особенностей.

Инструментарий

Для работы с книгой необходимы текстовый редактор (в нем записываются коды) и браузер (в нем происходят отладка кодов и работа с созданным продуктом).

В качестве текстового редактора хорошо подходит МикроМир (привязанность автора), но можно обойтись и блокнотом Windows или другим редактором, который умеет работать в кодировке Windows.

Какой браузер необходим? Разработчик сайтов должен всегда иметь под рукой несколько браузеров, минимум два: Microsoft Internet Explorer (IE) и Netscape Navigator (NN). Ведь пользователь сети может "заехать" на сайт, используя любой транспорт! И мы должны знать, что он увидит.

К сожалению (и большой головной боли интернет-программиста), браузеры разных фирм интерпретируют HTML-коды и скрипты по-разному. Технические различия порой столь велики, что приходится создавать две версии сайта: отдельно для IE и отдельно для NN.

С другой стороны, принципы построения динамических документов остаются практически одинаковыми, и это позволяет в учебном курсе ориентироваться только на один браузер, так как постоянное распараллеливание вносит дополнительный пласт технических подробностей, которые, накладываясь на концептуальное ядро, рассеивают внимание, усложняют изложение и запутывают новичка.

Автором курса принято решение в пользу браузера IE (версии 4.0 и старше):

— IE-браузер не надо устанавливать специально, он входит в состав Windows;

— с точки зрения программирования IE (как интерпретатор программного кода) устроен более логично, и тем самым на нем легче построить учебный курс для новичка.

Итак, для работы с книгой необходим браузер Microsoft Internet Explorer 4 (или старше). В подключении к Интернету необходимости нет.

Как устроена электронная книга

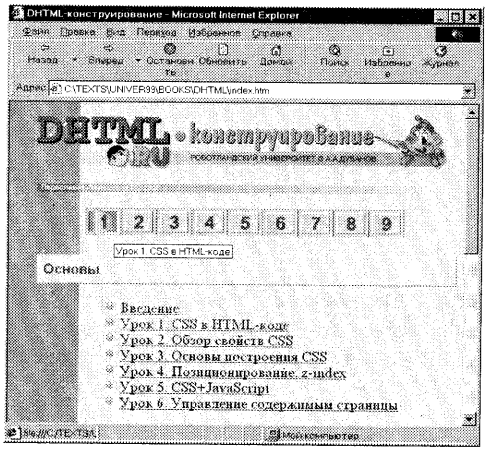
Работа с электронной книгой начинается с загрузки в браузер файла index.htm (он находится в корне пакета) и продолжается переходом по гиперссылкам оглавления к нужному разделу.

Документ index.htm представляет собой оглавление-меню книги и содержит:

— Верхний ряд "горячих" кнопок для быстрого входа в разделы (уроки) книги.

— Гипертекстовое оглавление-меню для выбора нужного тематического раздела.

— Справочник по свойствам CSS, снабженный многочисленными испытательными стендами, которые помогают понять практический смысл той или иной конструкции.



Электронные адреса

Демо-версию гипертекстовой книги (180 Кб) можно скопировать с адреса: **ftp://ftp.botik.ru/rented/robot/univer/dhtmldem.zip**.

Гипертекстовое приложение "Конюх" (127 Кб), которое подробно разбирается в книге, можно скопировать с адреса: **ftp://ftp.botik.ru/rented/robot/univer/horse.zip**.

Заказ на электронную версию можно послать с сайта **www.botik.ru/~robot** или письмом по адресу: **kurs@robotland.botik.ru**.

РАЗДЕЛ 1 ОСНОВЫ

УРОК 1. CSS В HTML-КОДЕ

Стиль для отдельного тега

CSS — *Cascading Style Sheets* (каскадные таблицы стилей) — это средство, позволяющее задавать различные свойства HTML-тегам.

Например, можно указать, как должен выглядеть конкретный абзац **P**:

```
<P style="font-size:1.5cm; color:green">
```

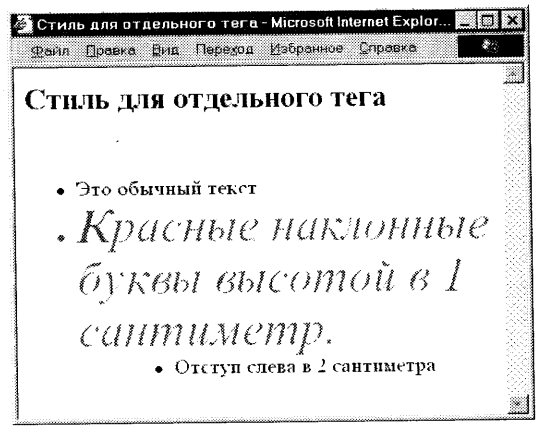
К этому абзацу применено стилевое определение.

Стиль задается атрибутом **style**. Браузеру дано указание вывести абзац зелеными буквами размером в 1,5 сантиметра. Стилевые определения задаются при помощи указаний вида

характеристика: величина;

Указания отделяются друг от друга символом ";".

Атрибут **style** можно использовать практически в каждом теге, задавая его специфические свойства. Посмотрите еще один пример внедрения стилей в теги HTML-программы.



Страница построена кодом:

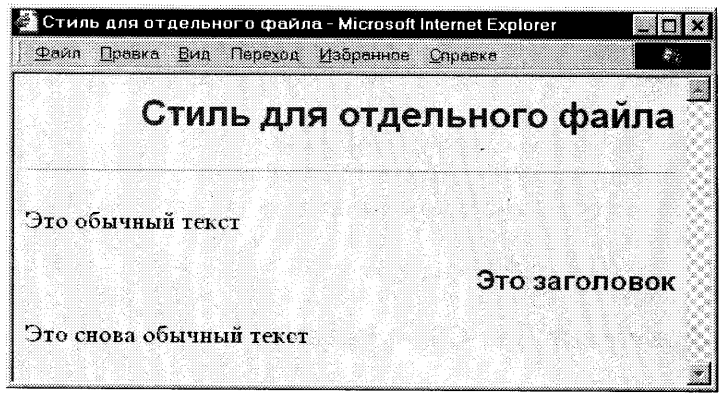
```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type" content="text/html;
        charset=windows-1251">
  <TITLE>Стиль для отдельного тега</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
<H2>Стиль для отдельного тега</H2>
<HR>
 <UL>
   <LI>Это обычный текст.
   <LI style="color:red; font-size:1cm; font-style:italic">
     Красные наклонные буквы высотой в 1 сантиметр.
   <LI style="margin-left:2cm"> Отступ слева в 2 сантиметра.
 </UL>
 </BODY>
</HTML>
```

Стиль для отдельного HTML-файла

Можно задавать стиль для тега или группы тегов так, чтобы определение работало на протяжении всего HTML-документа. Например, можно указать вид всех заголовков. Для этого достаточно написать определение в головной части документа:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Стиль для отдельного файла</TITLE>
  <STYLE type="text/css">
  <!--
    H1,H2,H3,H4,H5,H6
    {
     text-align:right;
     color:maroon;
     font-family:"Arial Cyr",Geneva,Helvetica,sans-serif;
    }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=#DFF0D5 text=black>
 <H2>Стиль для отдельного файла</H2>
 <HR>
```

```
<P>Это обычный текст
<H3>Это заголовок</H3>
<P>Это снова обычный текст
</BODY>
</HTML>
```



Браузер отображает заголовки в этом документе рубленым шрифтом цвета "maroon" и выравнивает их по правому краю экрана:

Такое поведение браузера соответствует инструкции:

```
<STYLE type="text/css">
  <!--
    H1,H2,H3,H4,H5,H6
    {
     text-align:right;
     color:maroon;
     font-family:"Arial Cyr",Geneva,Helvetica,sans-serif;
    }
  -->
  </STYLE>
```

Стилевые определения (**селекторы**) располагаются внутри тегов **<STYLE>...</STYLE>**, "запакованные" в HTML-комментарий (для браузеров, которые не поддерживают CSS).

Стилевое определение имеет вид:

Имя тега (или *имена тегов* через запятые)
{*характеристика*: *величина;*
...
характеристика: *величина;*
}

В приведенном выше примере использованы три характеристики:

`text-align:right;` — задает выравнивание по правому краю;

`color:maroon;` — задает цвет "maroon";

`font-family:"Arial Cyr", Geneva, Helvetica, sans-serif;` — задает рубленый шрифт.

Заголовки будут выводиться шрифтом "Arial Cyr", если, конечно, этот шрифт есть на компьютере пользователя. Если шрифта нет, браузер последовательно ищет шрифты "Geneva", "Helvetica" или в конце концов какой-нибудь рубленый шрифт (указание "sans-serif"). Если ни одного из этих шрифтов на компьютере не окажется, браузер выведет текст шрифтом по-умолчанию, то есть, как правило, шрифтом "Times New Roman".

Стиль для нескольких HTML-файлов

Обычной практикой является указание стилей в отдельном файле. Для таких файлов используют расширение "css". Например, можно в файле style.css записать:

```
BODY { margin-left:40px; }
H1,H2,H3,H4,H5,H6
{
 text-align:right;
 color:maroon;
 font-family:"Arial Cyr",Geneva,Helvetica,sans-serif;
}
```

Для подключения этих указаний в разделе **<HEAD>...</HEAD>** HTML-файла нужно поместить ссылку:

```
<LINK rel=stylesheet type="text/css" href=style.css>
```

Понятно, что такой способ расположения стилевых определений очень удобен. На один и тот же стилевой файл могут ссылаться много HTML-документов. Изменения в этом единственном файле скажутся на внешнем виде десятка (а то и сотни) документов.

Обратите внимание на стилевое указание

```
BODY {margin-left:40px;}
```

Оно задает отступ слева в 40 пикселей для всего документа в целом. В силу такого определения визуальные элементы страницы не будут "наползать" на вертикальную полосу слева страницы. Как видите, для задания страничного отступа можно обойтись без таблиц и "распорок", о которых говорилось в книге "HTML-конструирование".

Комбинирование стилей

Выше были показаны три способа внедрения стилевых указаний в HTML-коды:
— на уровне отдельного тега;
— на уровне одного HTML-файла;
— на уровне нескольких HTML-файлов.

А что если начать комбинировать эти способы? Какой из них окажется самым "сильным" для конкретного тега?

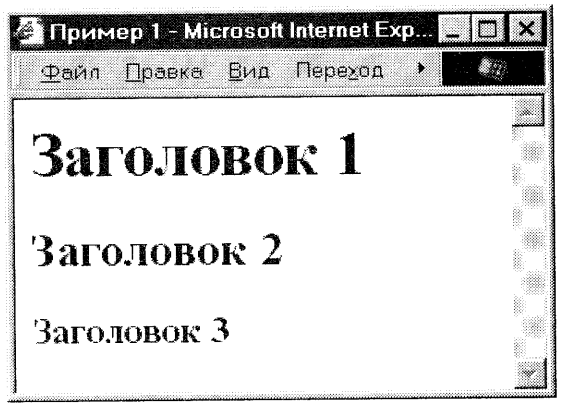
Проделаем серию опытов.

Документ без CSS-указаний

Заголовки выводятся черным по белому.

Код примера:

```
<HTML>
 <HEAD>
  <TITLE>Пример 1</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Заголовок 1</H1>
  <H2>Заголовок 2</H2>
  <H3>Заголовок 3</H3>
 </BODY>
</HTML>
```



CSS-указания в отдельном теге

```
<HTML>
 <HEAD>
  <TITLE>Пример 2</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Заголовок 1</H1>
  <H2>Заголовок 2</H2>
  <H3 style="color:red">Заголовок 3</H3>
 </BODY>
</HTML>
```

Код отобразит первые два заголовка черным, а последний — красным.

CSS-указания в HEAD

```
<HTML>
 <HEAD>
  <STYLE type="text/css">
   <!--
     H1,H2,H3
     {
      color:blue;
     }
   -->
  </STYLE>
  <TITLE>Пример 3</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Заголовок 1</H1>
  <H2>Заголовок 2</H2>
  <H3 style="color:red">Заголовок 3</H3>
 </BODY>
</HTML>
<HTML>
```

Код отобразит первые два заголовка синим, а последний — красным.

CSS-указания в CSS-файле

Файл prim.css содержит:

```
H1,H2,H3
{
 color:green;
}
```

HTML-файл имеет вид:

```
<HTML>
 <HEAD>
  <LINK rel=stylesheet type="text/css" href=prim.css>
  <STYLE type="text/css">
   <!--
     H1,H2,H3
     {
      color:blue;
     }
   -->
  </STYLE>
  <TITLE>Пример 4</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Заголовок 1</H1>
  <H2>Заголовок 2</H2>
  <H3 style="color:red">Заголовок 3</H3>
 </BODY>
</HTML>
<HTML>
```

Код отобразит первые два заголовка синим, а последний — красным.
Рассмотрим другой порядок расположения ссылки на файл prim.css:

```
<HTML>
 <HEAD>
  <STYLE type="text/css">
   <!--
     H1,H2,H3
     {
      color:blue;
     }
   -->
  </STYLE>
  <LINK rel=stylesheet type="text/css" href=prim.css>
  <TITLE>Пример 5</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Заголовок 1</H1>
  <H2>Заголовок 2</H2>
  <H3 style="color:red">Заголовок 3</H3>
 </BODY>
</HTML>
<HTML>
```

Код отобразит первые два заголовка зеленым, а последний — красным.

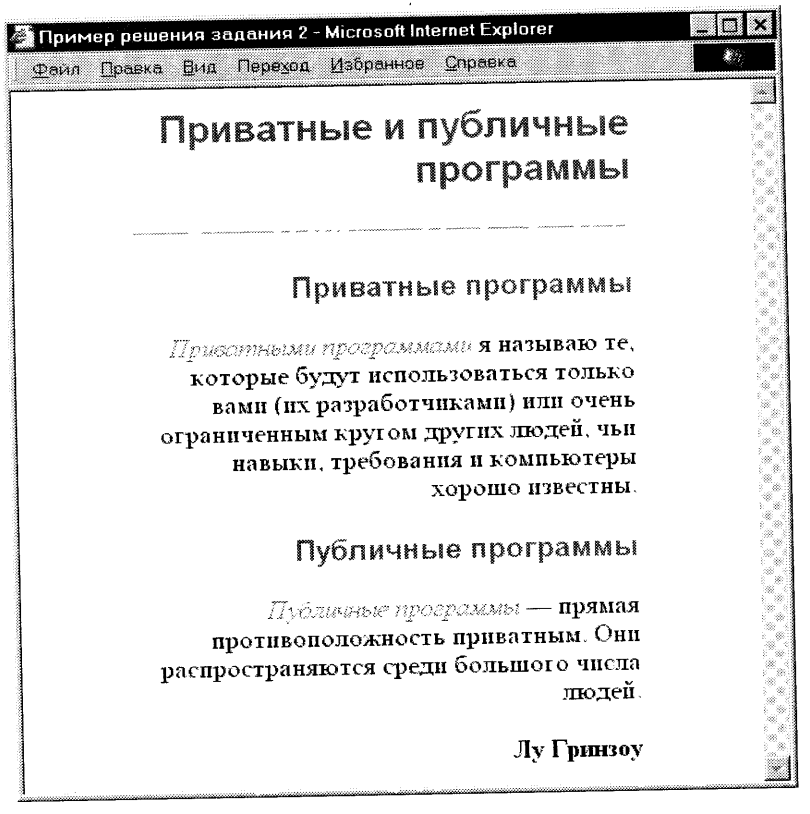
Задания

1. После внимательного изучения исходных текстов приведенных выше примеров ответьте на следующие вопросы:

— Какое CSS-указание главнее: описанное в отдельном теге или размещенное в разделе **HEAD** HTML-документа?

— Какое указание главнее: описанное в разделе **HEAD** HTML-документа или размещенное в отдельном CSS-файле и связанное с документом при помощи тега **LINK**, помещенного в раздел **HEAD**? Зависит ли результат от порядка следования этих предписаний?

2. Используя стили, постройте документ "Приватные и публичные программы", представленный на рисунке.



В этом документе:

— Основные цвета: черный текст на белом фоне.

— Отступ всех элементов на странице слева и справа равен по 2 см.

— Абзацы выравниваются справа.

— Заголовки выравниваются справа и записываются рубленым шрифтом красного цвета.

— Термины, выделяемые курсивом, записываются зеленым цветом.

За что мы любим CSS

Используя CSS, можно:

— Задавать поля, отступы, размер и тип шрифта, цвета текста и фона для отдельных элементов страницы (абзацев, слов, букв), оформлять красные строки, буквицы. В CSS представлен ряд свойств, с помощью которых можно создавать выпадающие меню, накладывать один элемент на другой, заставлять текст отбрасывать тень, проявляться в рентгеновских лучах и использовать другие эффекты. Можно выводить элементы на экран браузера с точностью до одного пикселя, динамически перемещать их по экрану в разных слоях (одни элементы будут проходить над или под другими). Иными словами, CSS вместе с JavaScript позволяет монтировать на экране браузера приложение, практически ничем не отличающееся от компьютерной программы, написанной на профессиональном языке программирования, таком, как, например, Си.

— Изменять оформление целого сайта, состоящего из сотен файлов, не прикасаясь к HTML-коду, редактируя всего лишь один CSS-файл.

— Уменьшать количество тегов в HTML-документе, по-возможности отделяя информационное наполнение HTML-файла от его внешнего представления на экране браузера.

УРОК 2. ОБЗОР СВОЙСТВ CSS

Существует более 70 свойств, которые можно использовать для составления стилевых определений. Много! Для практической работы, конечно, необходим справочник, особенно на первых порах. Справочник с испытательными стендами приводится в электронной книге. Однако для успешной работы необходимо иметь общее представление о том, чем можно управлять в принципе, и выполнить несколько простых учебных заданий.

Именно такому вводному (но не полному!) обзору с изрядной долей тренировочных упражнений посвящен второй урок.

Рассмотрение свойств структурировано (так же, как и в справочнике) по следующим разделам:

— шрифт;
— цвет;
— текст;
— поля и рамки;
— вид.

Начнем обзор стилевых свойств с единиц измерения, которые в них используются.

Единицы измерения

Обозначение	Название	Пример
in	дюймы	`<P style="font-size:1in">`
cm	сантиметры	`<P style="font-size:1cm">`
mm	миллиметры	`<P style="font-size:1mm">`
px	пиксели	`<P style="font-size:1px">`
pt	пункты	`<P style="font-size:1pt">`
pc	пики	`<P style="font-size:1pc">`
%	проценты	`<P style="font-size:1%">`

Пункты и пики — это типографские термины, которыми обозначают размер шрифта или кегль.

Например, в Word этот параметр задается в специальном окошке числами, как правило, от 8 до 72. Это величина кегля в пунктах. Один типографский пункт равен 0,375 мм. Причем это размер не самих символов шрифта, а размер так называемого "очка". То есть размер по вертикали матрицы, на которой гравируется в типографии символ (литера). Этот размер больше, чем размер самой литеры. Так, в шрифте кегля 10 заглавные буквы имеют размер около 7 пунктов.

При печати книг, как правило, для основного текста выбирается кегль в 10 или 12 пунктов. Для заголовков — более крупные кегли, а для ссылок и примечаний — более мелкие (обычно 8 пунктов).

Пика — более крупная единица измерения. Одна пика равна 12 пунктам.

Процентный отсчет ведется от какого-либо базового значения. В случае шрифта — от текущего размера.

Задание процентной величины очень привлекательно для разработчика.

Если пользователь меняет в браузере размер шрифта, размеры, указанные в стилевых указаниях, меняются пропорционально.

Шрифт

font-family

Это стилевое свойство задает название гарнитуры шрифта (например, "Arial") или родовое название:

`serif` — шрифт с засечками (серифный);
`sans-serif` — шрифт без засечек (рубленый);
`monospace` — моноширинный шрифт (буквы имеют одинаковую ширину).

Это шрифт с засечками.

Это рубленый шрифт.

Это моноширинный шрифт.

Гарнитура — это набор начертаний символов одного шрифта.

Этот набор может включать в себя прямое и курсивное начертания, различаться по степени жирности (толщине штрихов), ширине литер (сжатые, нормальные, растянутые) и кеглю.

Шрифты подразделяются на серифные (с засечками) и рубленые (без засечек). Пример шрифта с засечками — гарнитура "Times", без засечек — "Helvetica".

Серифные шрифты посредством засечек (серифов) в нижней части литер улучшают читаемость документа. Нижние серифы создают впечатление слитности написания слова, объединяя соседние буквы.

Рубленые шрифты, в отличие от серифных, имеют гладкие края букв и не имеют завитков.

Можно предложить еще одно деление шрифтов на два класса: пропорциональные шрифты и моноширинные.

Моноширинные шрифты имитируют литеры пишущих машинок и матричных принтеров. В этих шрифтах ширина всех литер одинакова. Именно таким шрифтом (если шрифт не задан CSS) браузер выводит текст на экран при использовании тегов **PRE**, **CODE**, **TT**, **SAMP**, **KBD**.

Обычные же тексты браузер выводит пропорциональным шрифтом. В пропорциональных шрифтах каждая буква занимает столько места, сколько ей действительно нужно.

Гарнитуры "Times" и "Helvetica" — пропорциональные. К моноширинным шрифтам относится гарнитура "Courier".

Любая графическая операционная система имеет три набора стандартных гарнитур; для Windows это:

"Times Roman" (серифный шрифт);
"Arial" (рубленый шрифт);
"Courier" (моноширинный шрифт).

Браузер по умолчанию использует шрифт "Times Roman" для вывода обычного текста и шрифт "Courier" для моноширинного.

Если гипертекстовый продукт — сайт, лучше не задавать собственную гарнитуру. Указанный шрифт на компьютере пользователя может не содержать русских букв, и текст станет похож на "древненорвежский":

Â øðèôòå íåò ðóññêèõ áóêâ.

Для локальных гипертекстовых приложений можно использовать любые шрифты. Такое приложение передается пользователю вместе с набором шрифтов и инструкцией по их установке.

При помощи свойства `font-family` можно указать не один, а несколько шрифтов через запятые в порядке предпочтения:

```
H1,H2,H3,H4,H5,H6
{
  font-family:"Arial Cyr",Geneva,Helvetica,sans-serif;
}
```

Сначала браузер будет пытаться найти шрифт "Arial Cyr", затем "Geneva", потом "Helvetica" и, наконец, в случае полной неудачи, какой-нибудь рубленый шрифт (указание `sans-serif`).

Если название шрифта состоит из нескольких слов, оно заключается в кавычки.

font-size

Свойство `font-size` используется для задания абсолютного или относительного размера текущего шрифта.

В качестве относительного размера используется процентное задание (по отношению к размеру текущего шрифта) или ключевые слова:

`larger` — крупнее;
`smaller` — мельче.

В качестве абсолютного размера используется указание числа с одной из следующих единиц измерения: in, cm, mm, px, pt, pc. Можно использовать и такие ключевые слова:

`xx-small` — сверхмелкий;
`x-small` — очень мелкий;
`small` — мелкий;
`medium` — средний;
`large` — крупный;
`x-large` — очень крупный;
`xx-large` — сверхкрупный.

Следует отдавать предпочтение относительным величинам, для того чтобы пользователь, используя настройки браузера, смог установить для себя комфортный размер текста.

Цвет и фон

В списках стилей поддерживаются три формы указания цвета:

— ключевое слово, например, `white`;
— шестнадцатеричный RGB-код, например, `#EEE5DB`;
— десятичный RGB-код, например, `rgb(255,64,0)`.

color

Определяет цвет элемента.

background-color

Определяет цвет фона, на который выводится элемент.

```
<P style="color: white;
   background-color: black">
```

Тестирование — это защитная сеть вокруг вашей программы, последняя и порой единственная надежда на серьезное повышение ее качества до того, как она выйдет в свет. (Лу Гринзоу, "Философия программирования")

Текст

letter-spacing

Определяет добавочное расстояние между буквами. Допускается использовать ключевое слово `normal` или задавать произвольное расстояние. Процентная запись не допускается.

```
<P style="letter-spacing: 10px">
```

Тестирование — это защитная сеть вокруг вашей программы, последняя и порой единственная надежда на серьезное повышение ее качества до того, как она выйдет в свет (Лу Гринзоу, "Философия программирования")

line-height

Определяет расстояние между строками (интерлиньяж). Можно задавать в количествах строк (1,4), высотой (14 pt) или в процентах от текущей высоты строки (200%). Например, двойной интервал между строками задается `line-height:2` или `line-height:200%`. Допускается задание ключевого слова `normal`.

```
<P style="line-height: 200%">
```

Тестирование — это защитная сеть вокруг вашей программы, последняя и порой единственная надежда на серьезное повышение ее качества до того, как она выйдет в свет (Лу Гринзоу, "Философия программирования")

text-align

Свойство определяет способ горизонтального выравнивания текста.
Допустимы следующие значения:

`left` — по левому краю;
`right` — по правому краю;
`center` — по центру;
`justify` — по левому и правому краям.

```
<P style="text-align: right">
```

Тестирование — это защитная сеть вокруг вашей программы, последняя и порой единственная надежда на серьезное повышение ее качества до того, как она выйдет в свет (Лу Гринзоу, "Философия программирования")

К сожалению, значением `justify` практически невозможно пользоваться на текстах с длинными словами (например, на русском или немецком языках).

Браузер не умеет пока переносить слова по слогам, и поэтому на экране при выравнивании `justify` часто образуются большие "дыры":

Электрификация с помощью электромеханического привода, расположенного на синусоидальном сердечнике магнитооптического дросселя

Поля и рамки

border-style

Определяет вид рамки элемента. Допустимы следующие значения:

`none` — рамки нет (по умолчанию);
`solid` — обычная сплошная граница;
`double` — двойная линия;
`groove` — "вдавленная" граница;
`ridge` — "выпуклая" граница;
`inset` — "вдавленный" элемент;
`outset` — "выпуклый" элемент.

border-color

Определяет цвет рамки. Свойство работает, если задано свойство `border-style`.

border-width

Определяет ширину рамки. Свойство работает, если задано свойство `border-style`. Ширину можно указывать числовыми единицами (процентной записи нет) или при помощи следующих ключевых слов:

`thin` — тонкая;
`medium` — средняя;
`thick` — толстая.

```
<P  style="border-style: solid ;
    border-color: #009900 ;
    border-width: 20px ">
```

Тестирование — это защитная сеть вокруг вашей программы, последняя и порой единственная надежда на серьезное повышение ее качества до того, как она выйдет в свет (Лу Гринзоу, "Философия программирования")

margin

Задает ширину поля (отступа) элемента от краев блока. Допускается числовая, процентная (от ширины блока) запись или ключевое слово `auto`.

В случае `auto` предполагается, что браузер будет подыскивать оптимальное значение самостоятельно.

padding

Задает расстояние между содержимым и рамкой элемента. Допускается числовая или процентная (от ширины элемента) запись.

```
<P  style="margin: 10pt ;
    padding: 5mm ">
```

Тестирование – это защитная сеть вокруг вашей программы, последняя и порой единственная надежда на серьезное повышение ее качества до того, как она выйдет в свет (Лу Гринзоу, "Философия программирования")

Вид

width

Устанавливает ширину элемента. Допускаются числовые, процентные значения и `auto`. Свойство используется для элементов **DIV** и **SPAN**. Эти теги используют для объединения элементов в блок. Подробнее о них рассказано в следующем уроке.

height

Устанавливает высоту элемента. Допускаются числовые значения и `auto`. Свойство используется для элементов **DIV** и **SPAN**.

```
<DIV  style="width: 80%;
      height:      200px">
Тестирование — это защитная сеть вокруг вашей программы,
последняя и порой единственная надежда на серьезное повышение ее
качества до того, как она выйдет в свет. (Лу Гринзоу, "Философия
программирования")
</DIV>
```

Задания

Разработайте код для вывода на экран следующего текста:

Стойкий оловянный солдатик

Было когда-то на свете двадцать пять оловянных солдатиков. Все они были сыновьями одной матери — старой оловянной ложки — и, значит, приходились друг другу родными братьями. Они были очень красивы: ружье на плече, грудь колесом, мундир красный с синим. Чудо что за солдатики!

Г.-Х.Андерсен

Приведенная ниже таблица содержит 7 заданий. Попробуйте получить красивое визуальное представление вашего документа.

Задания	01	02	03	04	05	06	07
Шрифт	Academy	Antiqua	Arial	Courier	Garamond	New York	Parsek
Размер	medium	large	xx-small	x-small	x-large	xx-large	14pt
Цвет	черный	подобрать	зеленый	подобрать	красный	подобрать	желтый
Фон	подобрать	синий	подобрать	голубой	подобрать	серый	подобрать
Рамка	подобрать	подобрать	подобрать	подобрать	подобрать	подобрать	подобрать
Поля	подобрать	подобрать	подобрать	подобрать	подобрать	подобрать	подобрать
Отступ	подобрать	подобрать	подобрать	подобрать	подобрать	подобрать	подобрать

УРОК 3. ОСНОВЫ ПОСТРОЕНИЯ CSS

Наследование

HTML-код имеет иерархическую структуру. Весь документ кодируется внутри тегов **<BODY>...</BODY>**. Внутри абзаца **P** могут содержаться элементы, размеченные тегами **STRONG** и так далее.

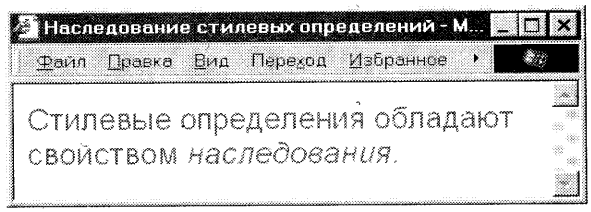
Наследование стилей означает, что если определен какой-то стиль для тега **BODY**, то этот стиль будет относиться ко всем тегам документа, без повторных определений. Соответственно, если задан стиль для некоторого тега, то все теги, расположенные внутри него в HTML-программе, также будут этим стилем обладать.

Пусть для тега **P** задан стиль:

```
P {
    color: red;
    font-size:14pt;
    font-family:Arial,sans-serif;
  }
```

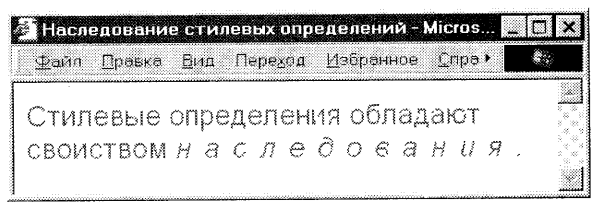
Тогда содержимое тега **EM**, помещенного внутрь абзаца, будет также выведено на экран красным рубленым шрифтом размером в 14 пунктов:

```
<P> Стилевые определения обладают свойством
   <EM>наследования</EM>.
```



Если нужно, например, добавить для тега **EM** разрядку текста, то такую модификацию нужно задать дополнительно:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Наследование стилевых определений</TITLE>
  <STYLE type="text/css">
  <!--
    P { color: red;
        font-size:14pt;
        font-family:Arial,sans-serif;
      }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
<P> Стилевые определения обладают свойством
<EM style="letter-spacing:6pt">наследования</EM>.
 </BODY>
</HTML>
```



Как видите, содержимое тега **EM** выводится так же, как и содержимое тега **P**, внутри которого он содержится (красным рубленым шрифтом в 14 пунктов), но благодаря дополнительному определению **style**="letter-spacing:6pt" между буквами появляются дополнительные промежутки в 6 пунктов.

Внутри тега-потомка можно не только ввести дополнительные стилевые определения, но и переопределить стилевые свойства родителя:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Наследование стилевых определений</TITLE>
  <STYLE type="text/css">
  <!--
    P { color: red;
        font-size:14pt;
        font-family:Arial,sans-serif;
      }
  -->
  </STYLE>
```

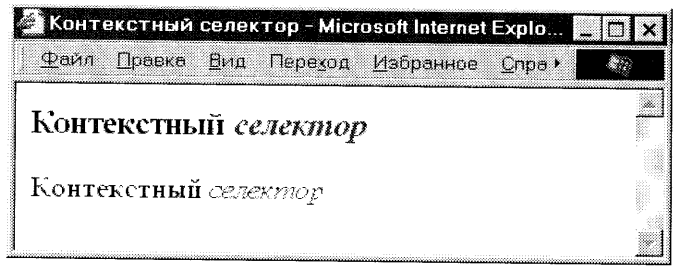
```
 </HEAD>
 <BODY bgcolor=white text=black>
<P> Стилевые определения обладают свойством
<EM style="letter-spacing:6pt;color:blue">наследования</EM>.
 </BODY>
</HTML>
```

Теперь слово "наследования" будет выведено синим цветом.

Контекстные селекторы

Можно написать стилевое определение, которое будет работать только при определенной комбинации вложенности тегов. Например, можно установить цвет **EM** синим только для случая, если этот тег расположен внутри тега **H3**:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Контекстный селектор</TITLE>
  <STYLE type="text/css">
  <!--
    H3 EM { color: blue }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
<H3>Контекстный <EM>селектор</EM></H3>
<P>Контекстный <EM>селектор</EM></P>
 </BODY>
</HTML>
```



Слово "селектор" выводится синим только в первой строке:

Вы видите, что **EM** внутри **H3** записывается синим, а внутри **P** — черным.

Обратите внимание, что в стилевом определении отсутствует запятая. Это и есть признак контекстного определения. Если же записать `H3, EM { color: blue }`, то синий цвет приобретут как теги **H3**, так и теги **EM**. То есть запятая определяет одинаковые стили для группы тегов.

Контекстные и групповые определения можно комбинировать. Запись

```
H3 EM, H2 I { color: blue }
```

равнозначна двум таким записям:

```
H3 EM { color: blue }
H2 I { color: blue }
```

Классы

Стилевые определения можно описывать без указания тега. В этом случае каждому определению присваивается имя, которое можно использовать для сопоставления заданного стиля конкретному тегу. Такие стилевые определения называются классами. Класс записывается следующим образом:

```
.имя
{
 характеристика: величина;
 ...
 характеристика: величина;
}
```

Иными словами, определение записывается, как обычно, но вместо указаний на теги размещается конструкция *.имя*. Например, можно определить стилевой класс с именем `def`:

```
.def
{
  color:red;
  font-size: 16pt;
  font-family:Geneva, Helvetica, sans-serif;
  border:solid 0.2cm blue;
}
```

Сопоставлять стилевой класс с тегом можно при помощи атрибута **class**:

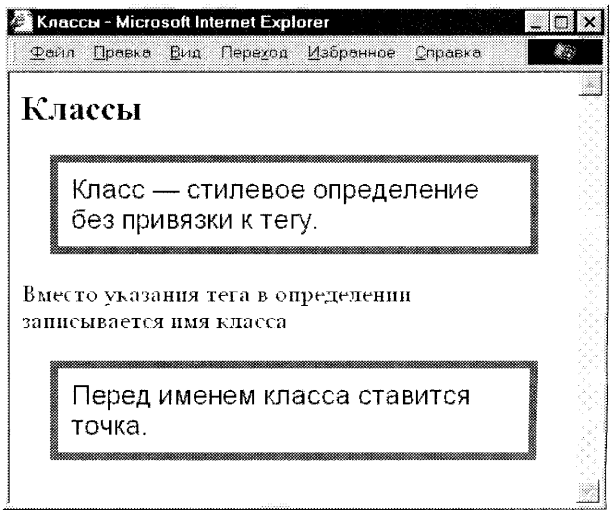
```
<P class=def>текст
```

Посмотрите, как работает такой код:

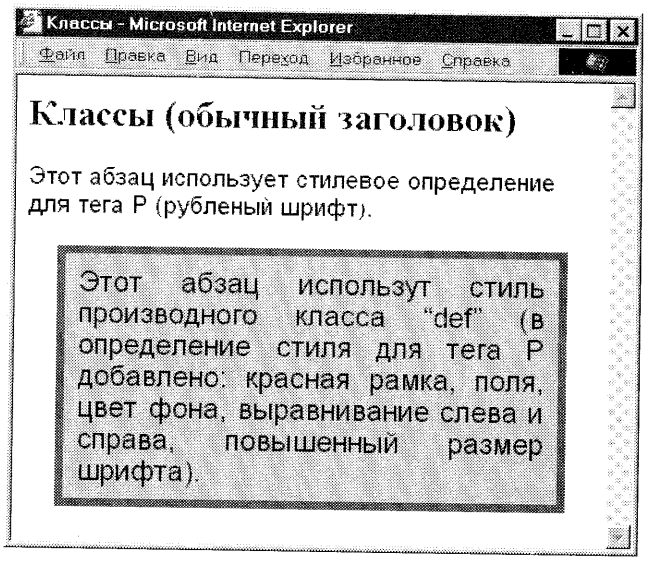
```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Классы</TITLE>
  <STYLE type="text/css">
  <!--
    .def
    {
      font-family: Helvetica, sans-serif;
      font-size: 14pt;
      border:solid 4pt red;
      padding: 6pt;
      margin-left: 5%;
      margin-right: 5%;
    }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <H2>Классы</H2>
   <P class=def>
    Класс — стилевое определение без привязки к тегу.
   <P>
    Вместо указания тега в определении записывается имя класса.
   <P class=def>
    Перед именем класса ставится точка.
 </BODY>
</HTML>
```

Можно образовывать классы на основе существующих стилевых определений. Посмотрите, как в следующем примере определен класс def, — за основу взято определение стиля для тега **P** и добавлены новые свойства:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
      content="text/html; charset=windows-1251">
  <TITLE>Классы</TITLE>
  <STYLE type="text/css">
  <!--
    P
    {
      font-family: Helvetica, sans-serif;
    }
    P.def
    {
      text-align: justify;
      background:#CFB597;
      font-size: 14pt;
      border:solid 4pt red;
      padding: 6pt;
      margin-left: 5%;
      margin-right: 5%;
    }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <H2>Классы (обычный заголовок)</H2>
```



```
    <P>
    Этот абзац использует стилевое определение
    для тега P (рубленый шрифт).
    <P class=def>
    Этот абзац использует стиль производного
    класса (в определение стиля для тега P
    добавлены: красная рамка, поля, цвет фона,
    выравнивание слева и справа, повышенный
    размер шрифта).
  </BODY>
</HTML>
```



Теги DIV и SPAN

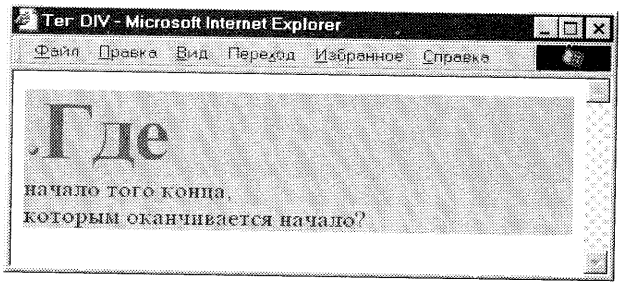
Эти теги играют особую роль для CSS. Они позволяют выделять в документе отдельные области, задавая для них специфические свойства. Все, что нужно сделать для этого, — это поместить теги, принадлежащие конструируемой области, внутрь **<DIV>...</DIV>** или **<SPAN>...</SPAN>**.

Разница между **DIV** и **SPAN** только в одном: после блока **DIV** браузер переходит на новую строку, а после блока **SPAN** — нет. Использование тега **SPAN** позволяет задавать стилевые свойства даже отдельным словам и буквам.

Посмотрите примеры с иллюстрациями использования этих тегов.

Пример 1

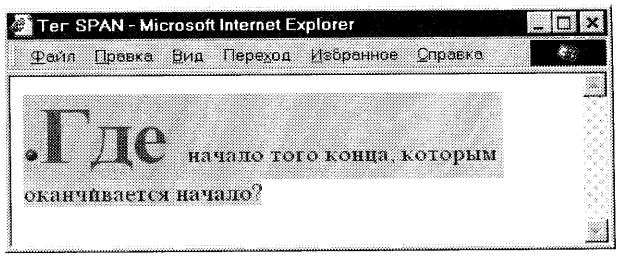
```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Тег DIV</TITLE>
  <STYLE type="text/css">
  <!--
   .area1
   {
    color:red; font-weight:bolder;
    font-size:40pt; background:aqua;
   }
   .area2
   {
    color:maroon; background:#CFB597;
   }
   .area3
   {
    color:blue; background:#C0C0C0;
   }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <DIV class=area1><IMG src=./pic/ball.gif>Где</DIV>
   <DIV class=area2>начало того конца,</DIV>
   <DIV class=area3>которым оканчивается начало?</DIV>
 </BODY>
</HTML>
```



Пример 2

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Тег SPAN</TITLE>
  <STYLE type="text/css">
```

```
  <!--
   .area1
   {
    color:red; font-weight:bolder;
    font-size:40pt; background:aqua;
   }
   .area2
   {
    color:maroon; background:#CFB597; padding:6pt;
   }
   .area3
   {
    color:blue; background:#C0C0C0; padding:6pt;
   }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <SPAN  class=area1><IMG src=./pic/ball.gif>Где </SPAN
   ><SPAN class=area2>начало того конца, </SPAN
   ><SPAN class=area3>которым оканчивается начало?</SPAN>
 </BODY>
</HTML>
```

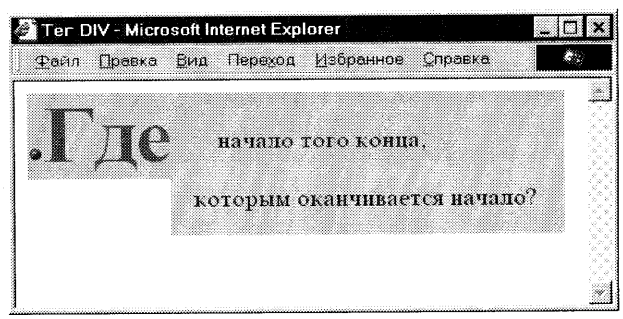


Абсолютное позиционирование

При помощи CSS можно отображать элементы на экране, используя реальные координаты, отсчитываемые от левого верхнего угла окна браузера. Такую возможность предоставляет стилевое свойство `position` со значением `absolute`. Сами координаты задаются при помощи свойств `left` (горизонтальная координата) и `top` (вертикальная координата).

Применение этих свойств иллюстрирует следующий пример.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type" content="text/html;
        charset=windows-1251">
  <TITLE>Тег DIV</TITLE>
  <STYLE type="text/css">
  <!--
   .area1 {
    position:absolute; top:10; left:10;
    color:red; font-weight:bolder;
    font-size:40pt; background:aqua
   }
   .area2 {
    position:absolute; top:20; left:150;
    color:maroon; background:#CFB597;
    padding:12pt;
   }
   .area3 {
    position:absolute; top:70; left:130;
    color:blue; background:#C0C0C0;
    padding:12pt;
   }
  -->
  </STYLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <DIV class=area1><IMG src=./pic/ball.gif>Где</DIV>
   <DIV class=area2>начало того конца,</DIV>
   <DIV class=area3>которым оканчивается начало?</DIV>
 </BODY>
</HTML>
```



Как видите, браузер разместил три области, заданные тегами **DIV**, в указанные координаты. При этом области перекрывают друг друга. Ближе к пользователю получается та область, которая следует в HTML-коде последней. Если переставить порядок следования тегов **DIV** в программе, то и порядок наложения областей друг на друга также изменится.

Однако CSS предоставляет программисту и другой, более гибкий, инструмент для управления порядком наложения элементов. Это — z-index.

Z-index

Это стилевое свойство позволяет указывать, в каком слое (на каком уровне) находится элемент на экране. Номер основного уровня (на который выводятся обычные элементы без стилевых указаний) равен нулю. Следовательно, элементы с отрицательным z-index расположены ниже (дальше), с положительным — выше (ближе) основного экранного слоя.

Если элементы имеют одинаковый z-index, то они расположены в одном слое. В противном случае "ближе" к нам расположен слой, имеющий больший z-index.

Пример

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
   <TITLE>Z-index</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H2>Z-index</H2>
  <HR>
  <DIV style="position:absolute; top:50; left:140;
              height:130; width:100; background:red;
              font-size:60; color:white; z-index:3">
    Это ближе</DIV>
  <DIV style="position:absolute; top:50; left:360;
              height:30; width:100; background:blue;
              font-size:30; color:white; z-index:1">
    Это дальше</DIV>
  <DIV style="position:absolute; top:80; left:270;
              width:125; z-index:2">
    <IMG src=pic/explorer.gif width=125 height=82 alt ="">
  </DIV>
 </BODY>
</HTML>
```

Каскадирование

Web-программист благодаря CSS получает в свои руки гибкий инструмент для определения своих собственных стилей представления информации на экране компьютера. Однако важно не запутаться в том, как созданные стили будут взаимодействовать друг с другом, с атрибутами тегов (они тоже определяют стиль!) и со стилями самого браузера (теми стилями, которыми браузер показывает документ по умолчанию).

Как было рассказано в предыдущем уроке, программист может использовать в HTML-документе три типа стилей:

— **Встроенный** (*inline*). Стиль, описанный внутри тега при помощи атрибута **style**. Контролирует представление отдельного тега.

— **Внедренный** (*embedded*). Стиль, описанный в заголовке HTML-файла при помощи тегов **<STYLE>...</STYLE>**. Контролирует представление отдельного HTML-документа.

— **Связанный** (*linked*). Стиль, описанный в отдельном CSS-файле. Контролирует представление многих HTML-документов. Для ссылки на стилевой файл в головной части HTML-документа записывают тег **<LINK>**.

В одном документе могут применяться все описанные выше три стилевых механизма. Добавим к этому стили, задаваемые обычными атрибутами тегов, и стиль "по умолчанию" самого браузера. Как это все взаимодействует, какому стилю отдает предпочтение браузер при построении документа на экране компьютера?

Правила, которым должен следовать "послушный" браузер, называются **каскадными**. Это означает, что для браузера самыми главными являются встроенные стили, затем, по убыванию старшинства, внедренный и связанный. Как следует из опытов предыдущего урока, внедренные и связанные стили совершенно равнозначны для

браузера. Он всегда следует последнему по порядку указанию.

Самым младшим в стилевой иерархии оказывается стиль "по умолчанию".

Его браузер использует тогда, когда нет никаких стилевых указаний. В понятие "каскадирование" входит и механизм наследственности, о котором говорилось ранее. Это означает, что, если у потомка нет своего стиля, к нему применяется стиль родителя. Что касается стилей, задаваемых обычными атрибутами тегов, то они работают по общему каскадному правилу (см. рисунок).

<FONT color="red"> <P style="color:blue"> Это будет показано синим. </P> </FONT>	Это будет показано синим.
<P style="color:blue"> Это будет показано синим, а <FONT color="red"> это будет показано красным. </FONT> </P>	Это будет показано синим, а это будет показано красным.

Задания

При подготовке заданий, представленных ниже, использовались задачи к лекциям Е.П. Лилитко "Программирование для Internet". При выполнении упражнений рекомендуется использовать справочник свойств CSS, который содержат эта и электронная книги (с испытателями).

1. Создайте страницу, в которой все абзацы выровнены по левому и правому краям и имеют красную строку в один сантиметр.

2. Для текстов заголовков и абзацев используйте рубленый шрифт. Сделайте новый стиль `p.def`. Абзацы этого стиля должны выравниваться по левому и правому краям, иметь отступ слева 2 см и справа 1 см. Расположите на странице обычные абзацы вместе с абзацами `p.def`.

3. Определите стили для написания старой и новой цен товара. Старая цена — серого цвета, перечеркнутая. Новая цена — красного цвета, на 50% более крупного кегля, чем остальной текст. Напишите список товаров со старыми и новыми ценами.

4. Определите два стиля. В первом стиле:

— Буквы должны печататься коричневым по светло-серому фону.

— Расстояние между содержимым и рамкой элемента должно составлять 0,5 см.

— Текст выравнивается по левому и правому краям.

Во втором стиле:

— Фон бирюзовый.

— Расстояние между содержимым и рамкой элемента должно составлять 0,5 см.

— Поля слева и справа от элемента по 1 см.

— Рубленый шрифт.

Подготовьте документ с двумя разделами. Первый раздел определяется первым стилем, второй — вторым. Второй раздел должен быть вложен в первый, чтобы было видно наследование. Какие стилевые указания наследуются во втором разделе, а какие нет?

5. Определите стиль `.nb` таким образом, чтобы элемент заключался в рамку (бордюр), занимал (по ширине) половину окна браузера (независимо от его размера), был расположен у левого края, а остальные элементы страницы обтекали бы этот элемент справа.

6. Подготовьте стиль для абзаца, у которого слева и справа проводится вертикальная черта (на всю высоту абзаца).

7. Подготовьте стиль абзаца "подложка" (`.ground`) и стиль "надпись" (`.poster`). Используя созданные стили, создайте страницу с надписью поверх подложки. Подложка пишется очень крупными буквами мягкого светло-серого цвета. Надпись пишется коричневыми буквами обычного размера.

8. Используя стили, сделайте страницу, в которой текст выводится в две колонки (как в газете). Таблицы при этом не используйте.

9. Постройте на экране две области с линейками прокрутки и поместите в них информационные элементы.

10. Используя свойство `z-index`, постройте на экране несколько перекрывающих друг друга областей.

УРОК 4. ПОЗИЦИОНИРОВАНИЕ, Z-INDEX

Позиционирование

Позиционирование — это управление координатами размещения элемента в окне браузера. CSS предлагает для позиционирования свойство `position`.

У этого свойства могут быть три значения: `absolute` (абсолютное позиционирование), `relative` (относительное позиционирование) и `static` (статическое позиционирование). Значение `static` размещает элемент на странице так, как он располагался бы без всякого позиционирования, поэтому использование этого значения не дает ничего нового. Алгоритмы работы значений `absolute` и `relative` будут рассмотрены ниже, но сначала одно важное напоминание об иерархической структуре кода HTML.

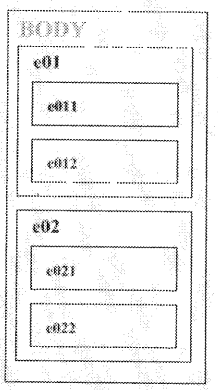
Иерархия кода страницы

Иерархия — это структура, в которой содержимое "упаковано" по пакетикам, вложенным друг в друга наподобие матрешек.

Все элементы страницы расположены внутри элемента **BODY** (блока **<BODY>...</BODY>**). Таким образом, элемент **BODY** является родителем всех других элементов страницы. Пусть, например, страница имеет следующий код:

```
<BODY id=e0>
  <DIV id=e01>
    <DIV id=e011>
      ...
    </DIV>
    <DIV id=e012>
      ...
    </DIV>
  </DIV>
  <DIV id=e02>
    <DIV id=e021>
      ...
    </DIV>
    <DIV id=e022>
      ...
    </DIV>
  </DIV>
</BODY>
```

Элемент BODY имеет здесь два прямых потомка e01 и e02. В свою очередь, эти элементы тоже имеют по два потомка e011, e012 и e021, e022 соответственно. Эту иерархическую схему можно изобразить так:



Позиционирование элементов выполняется с учетом иерархии кода. Вот почему так важно отчетливо представлять HTML-структуру документа.

Опытные программисты всегда записывают код лесенкой (как в приведенном примере), и у них возникает гораздо меньше проблем при программировании, тестировании и отладке своих страниц.

Абсолютное позиционирование

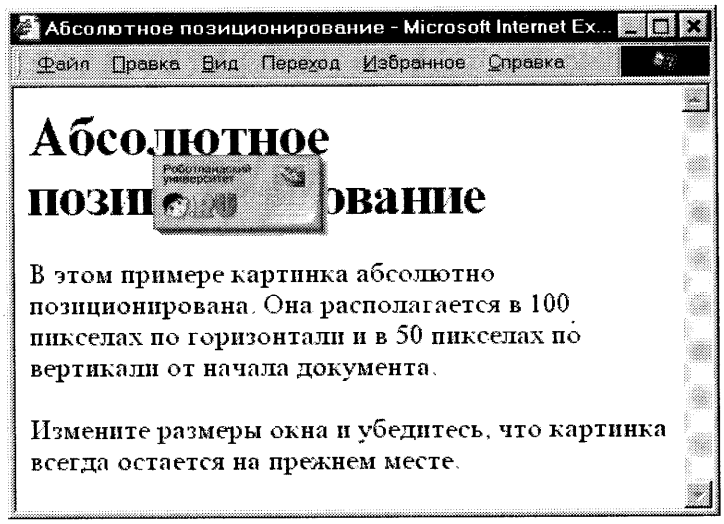
Абсолютное позиционирование задается стилевым указанием `position:absolute`. При этом начало координат элемента находится в верхнем левом углу прямого предка, при условии, что он также позиционирован (абсолютно или относительно). Если родитель не позиционирован, то берется его родитель. Если все предки не имеют указаний `position`, то в качестве точки отсчета принимается левый верхний угол экранного образа тега **BODY**, то есть левый верхний угол документа.

Горизонтальная и вертикальная координаты задаются свойствами `left` и `top` соответственно.

Ниже показан пример, в котором картинка позиционируется в точку (100, 50). Началом координат браузер считает начало документа.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Абсолютное позиционирование</TITLE>
 </HEAD>
```

```
<BODY bgcolor=white text=black>
  <H1>Абсолютное позиционирование</H1>
  <P>
  В этом примере картинка абсолютно позиционирована. Она располагается в 100 пикселях по
  горизонтали и в 50 пикселях по вертикали от начала документа.
  Измените размеры окна и убедитесь, что картинка всегда остается на прежнем месте.
  <IMG  src=./pic/let.gif width=126 height=60 border=0
        alt="Роботландский университет"
        style="position:absolute;
               left:100px; top:50px;">
 </BODY>
</HTML>
```



В следующем примере абсолютно позиционированы две картинки. В коде для каждой из них указаны координаты (100, 50), но для одной началом координат является начало документа, а для другой — начало таблицы.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html;
                 charset=windows-1251">
  <TITLE>Абсолютное позиционирование
  </TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Абсолютное позиционирование</H1>
  <P>
  Обе картинки абсолютно позиционированы в точку (100, 50).
  Начало координат для одной картинки совпадает с началом документа, для второй — с началом
  таблицы.
  <TABLE border=1 cellspacing=0 cellpadding=10
         style="position:absolute; left:50px; top:300px;">
    <TR><TD>
      Таблица абсолютно позиционирована.
      <IMG src=./pic/let.gif width=126 height=60 border=0
           alt="Роботландский университет"
           style="position:absolute; left:100px; top:50px;">
    </TD></TR>
  </TABLE>

  <IMG src=./pic/let.gif width=126
       height=60 border=0
       alt="Роботландский университет"
       style="position:absolute;
              left:100px; top:50px;">
 </BODY>
</HTML>
```

Приведенные примеры наглядно показывают, что абсолютно позиционированные элементы выпадают из процесса обычного последовательного форматирования. Браузер не берет в расчет порядок следования кодов, а учитывает только вложенность для определения начала координат. Элементы выводятся в указанном месте, перекрывая при этом другие элементы страницы.

Порядок следования кодов абсолютно позиционируемых элементов становится важным, если они начинают перекрывать друг друга: "выше" оказывается тот элемент, код которого идет позже.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Абсолютное позиционирование</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Абсолютное позиционирование</H1>
  <P>
   В этом примере обе картинки абсолютно позиционированы.
   Так как код "Незнайки" идет раньше кода "Профессора", то вторая картинка перекрывает
   первую.
  <IMG  src=./pic/prav01.gif width=140 height=150 border=0
        alt="Незнайка"
        style="position:absolute; left:100px; top:150px;">

  <IMG  src=./pic/prav31.gif width=140 height=150 border=0
        alt="Профессор"
        style="position:absolute; left:200px;
               top:250px;">
 </BODY>
</HTML>
```

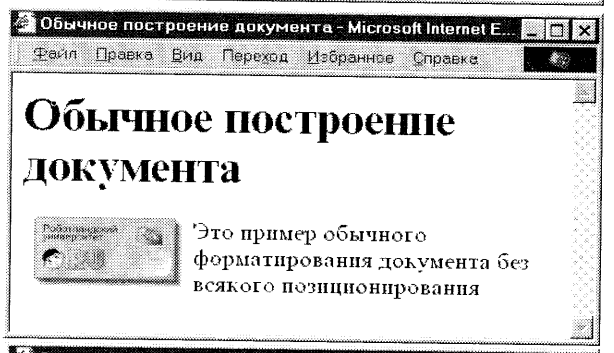
Относительное позиционирование

Относительное позиционирование задается стилевым указанием `position:relative`. Такой элемент принимает участие в обычном последовательном форматировании документа. За начало координат принимается та точка, куда элемент был бы размещен без позиционирования.

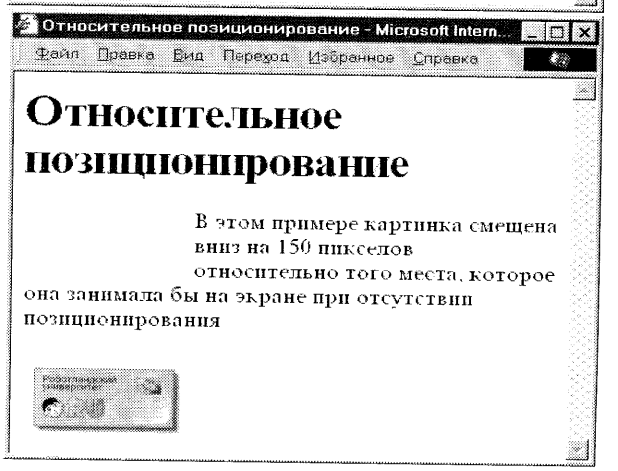
Алгоритм относительного позиционирования можно представить следующим образом. Сначала браузер размещает элемент на странице, выполняя обычное форматирование, а затем по указаниям `left` и `top` смещает его на заданные значения.

В приведенном ниже примере документ форматируется обычным образом.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
       content="text/html; charset=windows-1251">
  <TITLE>Обычное построение документа</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
  <H1>Обычное построение документа</H1>
  <P>
   <IMG src=./pic/let.gif width=126 height=60
        border=0 alt="Университет" align=left
        hspace=10>
   Это пример обычного форматирования документа без
   всякого позиционирования.
 </BODY>
</HTML>
```



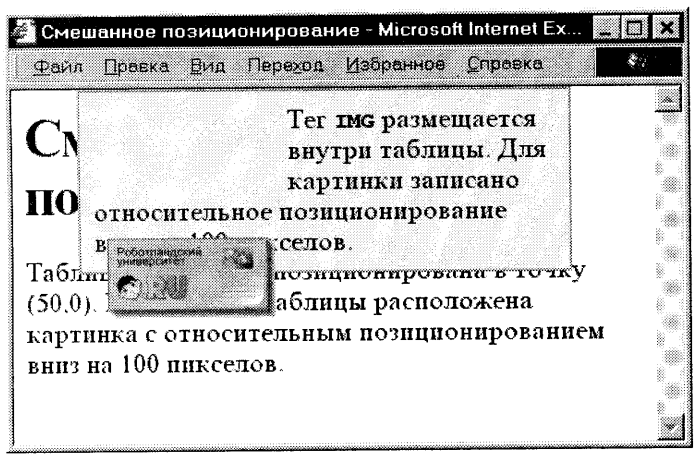
В следующем примере на картинку наложено относительное позиционирование. Если изменить размер окна, то можно увидеть, что картинка перемещается по экрану вслед за точкой отсчета — началом абзаца, которому принадлежит тег **IMG** в коде документа. Начало абзаца меняет свое положение тогда, когда предшествующий заголовок записывается в одну или две экранные строки.

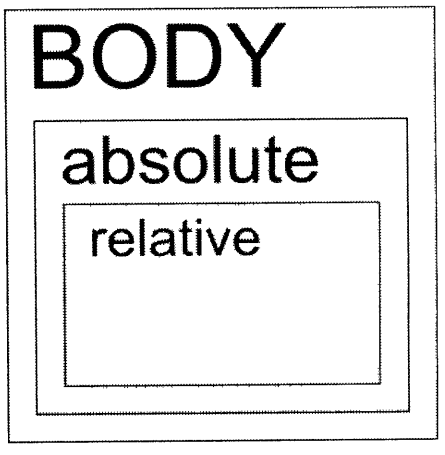


Смешанное позиционирование

В первом примере код относительно позиционированной картинки расположен внутри кода абсолютно позиционированной таблицы. Таблица смещена от начала документа вправо на 50 пикселей, и это положение остается фиксированным при изменении размеров окна. Картинка смещена вниз на 100 пикселей относительно ее "нормального" положения внутри таблицы. При этом в таблице текст расположен так, как будто картинка находится на своем обычном месте.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Смешанное позиционирование</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <H1>Смешанное позиционирование</H1>
   <P>
   Таблица абсолютно позиционирована в точку (50, 0).
   Внутри кода таблицы расположена картинка с относительным позиционированием вниз на
   100 пикселей.
   <P>
   <TABLE border=1 cellspacing=0 cellpadding=10
          width=80% bgcolor=#EEE5DB
          style="position:absolute; left:50px; top:0px">
     <TR><TD>
       <P>
        <IMG src=./pic/let.gif width=126 height=60 border=0
             alt="Университет" align=left hspace=10
             style="position:relative; left:0px; top:100px;">
        Тег <TT><B>IMG</B></TT> размещается внутри таблицы.
        Для картинки записано относительное позиционирование
        вниз на 100 пикселей.<BR clear=left>
     </TD></TR>
   </TABLE>
 </BODY>
</HTML>
```

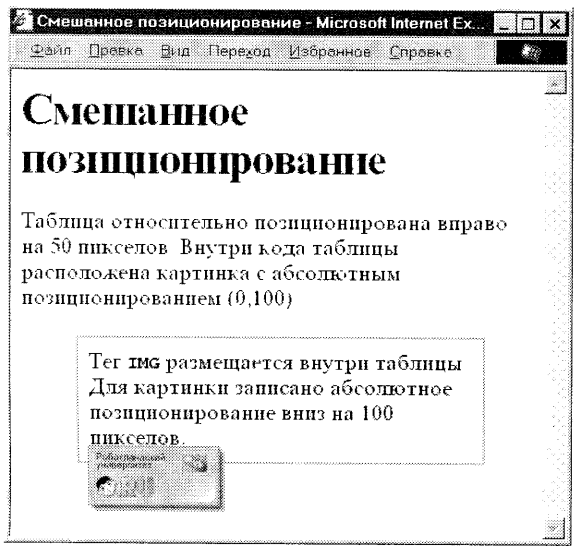


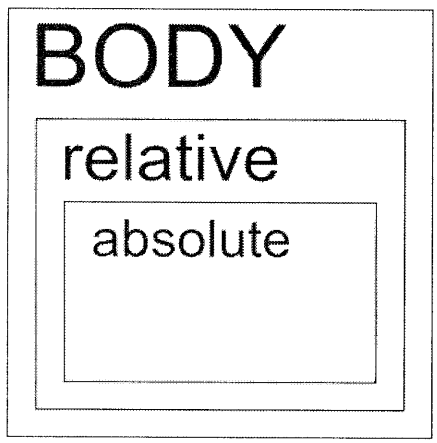


Во втором примере код абсолютно позиционированной картинки расположен внутри кода относительно позиционированной таблицы. Таблица смещена относительно своего "нормального" положения вправо на 50 пикселей. Начало координат для картинки связано с левым верхним углом таблицы. Картинка смещается вниз относительно этого положения на 100 пикселей. При этом текст в таблице форматируется без учета тега **IMG**.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Смешанное позиционирование</TITLE>
 </HEAD>
```

```
<BODY bgcolor=white text=black>
  <H1>Смешанное позиционирование</H1>
  <P>
  Таблица относительно позиционирована вправо на 50 пикселей.
  Внутри кода таблицы расположена картинка с абсолютным
  позиционированием (0,100).
  <P>
  <TABLE border=1 cellspacing=0 cellpadding=10
         width=80% bgcolor=#EEE5DB
         style="position:relative; left:50px; top:0px">
    <TR><TD>
      <P>
       <IMG src=./pic/let.gif width=126 height=60 border=0
            alt="Университет" align=left hspace=10
            style="position:absolute; left:0px; top:100px;">
       Тег <TT><B>IMG</B></TT> размещается внутри таблицы.
       Для картинки записано абсолютное позиционирование
       вниз на 100 пикселей.<BR clear=left>
    </TD></TR>
  </TABLE>
 </BODY>
</HTML>
```

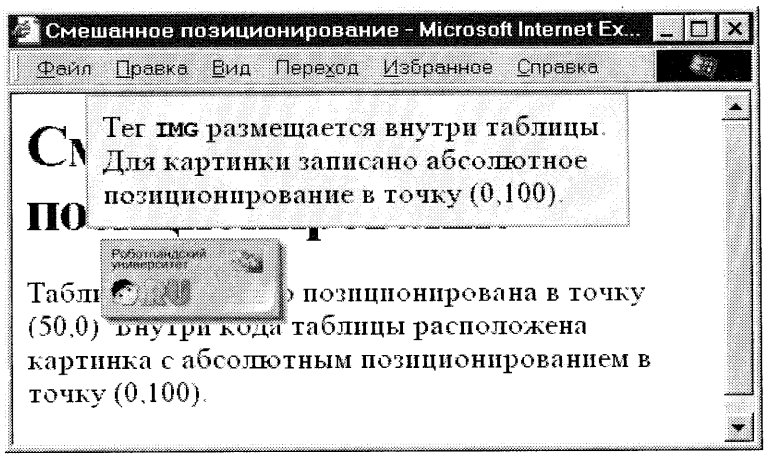




В третьем примере код абсолютно позиционированной картинки расположен внутри кода абсолютно позиционированной таблицы. Картинка не меняет своего положения на экране при изменении размера окна, потому что координаты таблицы остаются неизменными.

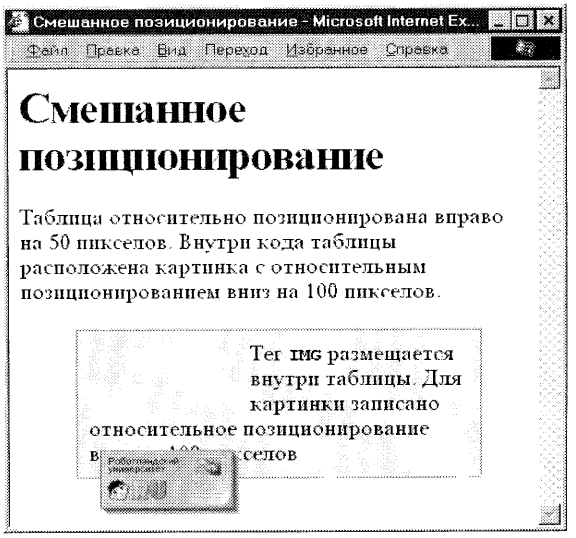
```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Смешанное позиционирование</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <H1>Смешанное позиционирование</H1>
   <P>
   Таблица абсолютно позиционирована в точку (50,0).
   Внутри кода таблицы расположена картинка с абсолютным
   позиционированием в точку (0,100).
   <P>
   <TABLE border=1 cellspacing=0 cellpadding=10
          width=80% bgcolor=#EEE5DB
          style="position:absolute; left:50px; top:0px">
```

```
      <TR><TD>
        <P>
         <IMG src=./pic/let.gif width=126 height=60 border=0
              alt="Университет" align=left hspace=10
              style="position:absolute; left:0px; top:100px;">
         Тег <TT><B>IMG</B></TT> размещается внутри таблицы.
         Для картинки записано абсолютное позиционирование
         в точку (0,100).<BR clear=left>
      </TD></TR>
    </TABLE>
  </BODY>
</HTML>
```

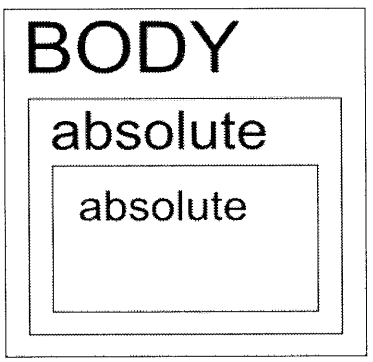


В четвертом примере код относительно позиционированной картинки расположен внутри кода относительно позиционированной таблицы. Когда положение таблицы меняется на экране из-за изменения размеров окна, меняется и положение картинки: ведь смещается начало координат, ниже которого на 100 пикселей должна быть расположена картинка.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Смешанное позиционирование</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
   <H1>Смешанное позиционирование</H1>
   <P>
   Таблица относительно позиционирована вправо
   на 50 пикселей.
   Внутри кода таблицы расположена картинка
   с относительным позиционированием вниз
   на 100 пикселей.
   <P>
   <TABLE border=1 cellspacing=0 cellpadding=10
          width=80% bgcolor=#EEE5DB
          style="position:relative; left:50px; top:0px">
     <TR><TD>
       <P>
        <IMG src=./pic/let.gif width=126 height=60 border=0
             alt="Университет" align=left hspace=10
             style="position:relative; left:0px; top:100px;">
        Тег <TT><B>IMG</B></TT> размещается внутри
        таблицы.
        Для картинки записано относительное
        позиционирование вниз
        на 100 пикселей.<BR clear=left>
     </TD></TR>
   </TABLE>
  </BODY>
</HTML>
```



Приведенные выше примеры наглядно показывают, что элемент позиционируется вместе со своими потомками, независимо от того, как позиционируются последние (или не позиционируются вовсе).

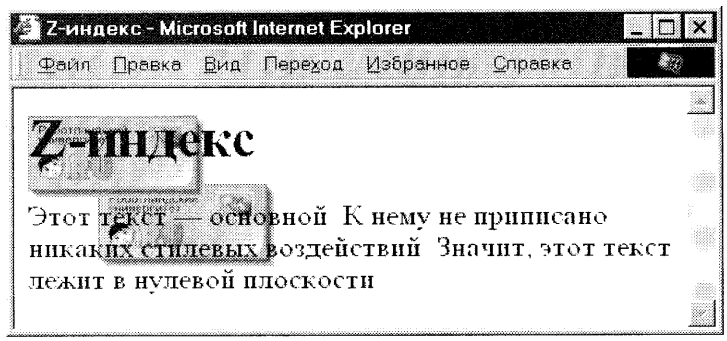


Z-индекс

Стилевое свойство z-index позволяет отойти от плоского представления документа на экране. Его значением является целое число, номер плоскости, в которой будет размещаться элемент (вместе со своими потомками). Основной текст имеет нулевой уровень (z-index:0).

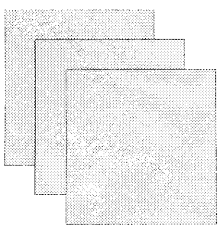
Положительный z-индекс размещает элементы над основным текстом, отрицательный — под ним. Из двух плоскостей размещения та расположена выше, у которой z-индекс больше.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Z-индекс</TITLE>
 </HEAD>
 <BODY bgcolor=white text=black>
 <H1>Z-индекс</H1>
 <P>
 Этот текст &#151; основной. К нему не приписано никаких стилевых воздействий.
 Значит, этот текст лежит в нулевой плоскости.
  <IMG src=./pic/let.gif width=126 height=60 border=0
       alt="Университет" align=left hspace=10
       style="position:absolute; left:0px; top:20px; z-index=-1;">
  <IMG src=./pic/let.gif width=126 height=60 border=0
       alt="Университет" align=left hspace=10
       style="position:absolute; left:50px; top:70px; z-index=-2;">
 </BODY>
</HTML>
```

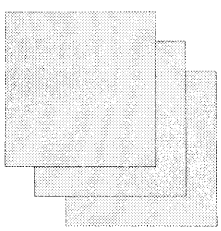


Если у элементов один уровень (задан явно или не задан вовсе), то тот из них будет располагаться выше, чей HTML-код идет позже.

Задания

1. 

Создайте страницу, в которой одна и та же картинка выводится несколько раз со смещением вниз и вправо так, чтобы каждая следующая копия была выше предыдущей.

2. 

Создайте страницу, в которой одна и та же картинка выводится несколько раз со смещением вниз и вправо так, чтобы каждая следующая копия была ниже предыдущей.

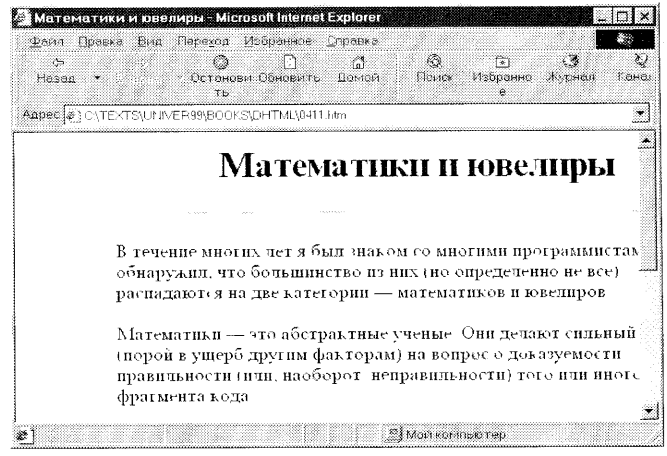
3. Иван пошел в лес
за белыми грибами.

Стилевое свойство `overflow` используется для определения того, что случится, если наполнение элемента выйдет за пределы заданной области. Среди предписываемых стандартом значений нормально работает только `scroll` — содержимое прокручивается. Напишите страничку, которая демонстрирует работу свойства `overflow`.

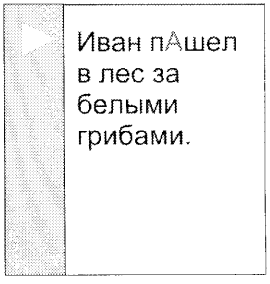
4. Иван Мячиков решил задать для текстового блока на своей странице широкое левое поле (см. страницу "Математики и ювелиры"). Он поместил блок внутрь конструкции

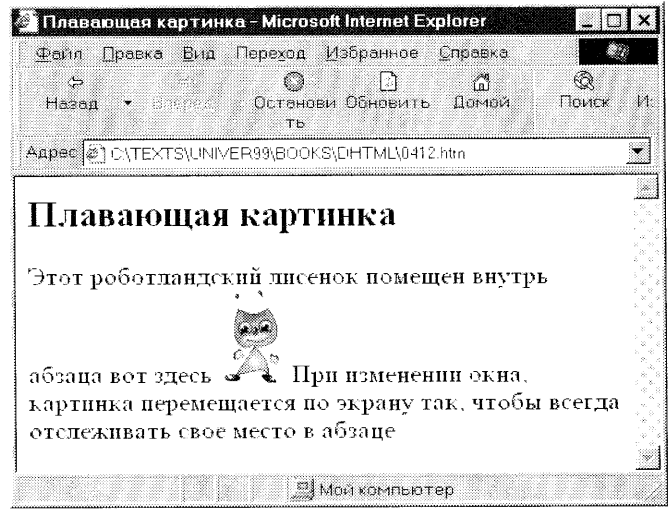
```
<DIV style="position:relative; left:100;">
  ...
</DIV>
```

Поле получилось, но текст перестал форматироваться по правой границе окна. Помогите Мячикову исправить положение.



5.





Если у абсолютно позиционированного элемента опущена одна или две координаты, то:

— Если не задано свойство `left`, то левый край элемента будет располагаться справа за элементом-родителем.

— Если не задано свойство `top`, то верхний край элемента будет располагаться ниже элемента-родителя. При этом если родитель позиционирован относительно, то элемент будет располагаться по вертикали там, где он располагался бы, если бы вообще не был позиционирован.

Используя описанные особенности, создайте страничку с левым полем, в котором бы располагался маркер, указывающий на ошибочное слово в строке. При этом, если из-за изменения размера окна ошибочное слово переместится в другую строку, маркер должен автоматически следовать за ним.

6. Если картинку поместить в абзац, то она ведет себя как символ, перемещаясь по экрану за своим местом (при изменении размера окна браузера).

Создайте страничку, в которой целый блок (с несколькими картинками, текстом, другими элементами) ведет себя подобным образом.

7. DHTML

Используя позиционирование, создайте "объемную" надпись из обычного текста (без картинок). Эта надпись должна быть привязана к своему месту в абзаце, подобно обычному слову.

УРОК 5. CSS + JAVASCRIPT

Принцип программного управления

Если для тега HTML применен стиль CSS, то это отражается на объектной модели документа. Браузер предусматривает для соответствующего объекта свойство **style**. Это свойство само является объектом и содержит информацию о заданных стилях. Эту информацию можно читать и менять, а значит, менять внешнее представление (или положение) элемента на экране. В этом и состоит простая суть программного управления стилями в HTML-документе.

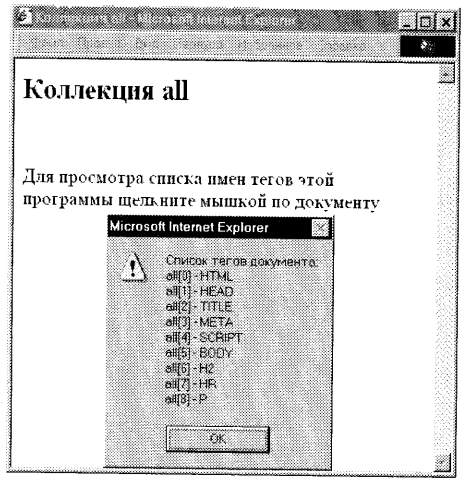
Доступ к элементам стиля

Возникает законный вопрос: как добраться до свойства **style** объекта, построенного браузером для тега документа? Один из способов — использовать коллекцию `all` объекта `document`.

Коллекция — это массив объектов. Коллекция `all` содержит объекты, построенные для всех тегов документа. Так, `document.all[0]` — это объект, построенный для первого тега HTML-документа, `document.all[1]` — для второго и так далее. Каждый объект коллекции `all` имеет свойство `tagName` — имя тега, которому соответствует объект.

Расположенный ниже HTML-документ показывает в окошке `alert` свои теги:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Коллекция all</TITLE>
  <SCRIPT language=JavaScript>
  <!--
   function showRU()
   {
     var str="Список тегов документа:\n";
     for(var i=0; i<document.all.length; i++)
       str += "all["+i+"] - "+document.all[i].tagName+"\n";
     alert(str);
   }
  //-->
  </SCRIPT>
 </HEAD>
 <BODY bgcolor=white text=black onclick="showRU()">
  <H2>Коллекция all</H2>
  <HR>
  <P>
   Для просмотра списка имен тегов этой программы
   щелкните мышкой по документу.
 </BODY>
</HTML>
```



Обращаться по индексу к элементу коллекции `all` неудобно. Во-первых, этот индекс не хочется высчитывать вручную, во-вторых, он меняется, если мы помещаем в документ новые теги. Последняя неприятность делает использование числового индекса совершенно невозможным. К счастью, в качестве индекса можно использовать имя или идентификатор тега. Имя мы использовали и раньше, оно вводится при помощи атрибута **name**. Идентификатор, играющий практически ту же роль, что и имя, задается атрибутом **id**.

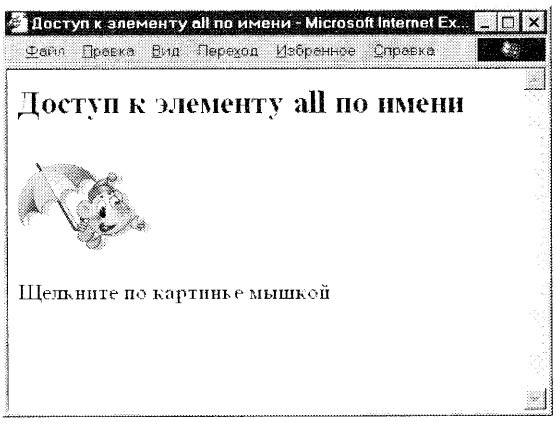
Пусть, например, в HTML-коде задан тег:

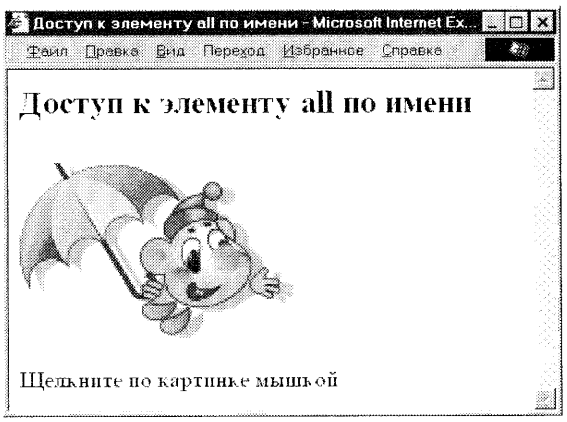
```
<IMG id=pic src=pic/explorer.gif width=125
     height=82 alt ="" onclick="showRU()">
```

Добраться до соответствующего ему объекта можно, заменив индекс коллекции `all` на имя, заданное `id`-атрибутом: `document.all["pic"]`. Теперь в функции `showRU()` можно менять свойства этого объекта, например, его размер. Именно так и сделано в следующем примере.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Доступ к элементу all по имени</TITLE>
```

```
<SCRIPT language=JavaScript>
<!--
 var size=true;
 function showRU()
 {
   if (size)
   {
    document.all["pic"].width  *= 2;
    document.all["pic"].height *= 2;
   }
   else
   {
    document.all["pic"].width  /= 2;
    document.all["pic"].height /= 2;
   }
   size = !size;
 }
//-->
</SCRIPT>
</HEAD>
```






```
<BODY bgcolor=white text=black>
 <H2>Доступ к элементу all по имени</H2>
 <HR>
   <IMG id=pic src=pic/explorer.gif width=125
        height=82 alt="" onclick="showRU()">
 <P>Щелкните по картинке мышкой.
</BODY>
</HTML>
```

Замечание

Если в приведенном выше примере заменить доступ к объекту **IMG**, записанный как `document.all["pic"]`, просто на `pic`, то все будет прекрасно работать! То есть функция `showRU()` может иметь и такой вид:

```
function showRU()
{
  if (size)
  {
   pic.width  *= 2;
   pic.height *= 2;
  }
  else
  {
   pic.width  /= 2;
   pic.height /= 2;
  }
  size = !size;
}
```

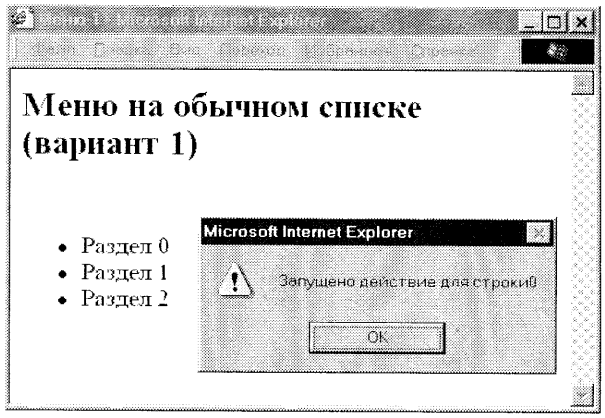
Однако такое обращение к объектам документа работает не для всех тегов.

Нас интересует свойство **style**. Добраться до него можно по следующей схеме: `document.all["имя"].style`.

Построение динамического меню

Посмотрите на следующий HTML-код:

```
<UL type=disc>
 <LI onclick="go(0)">Раздел 0
 <LI onclick="go(1)">Раздел 1
 <LI onclick="go(2)">Раздел 2
</UL>
```



Ничего особенного! Обычный список с обработкой события **onclick** на каждом из трех строк списка. Функция `go` может выполнять какие-то действия, например, загружать на экран новые страницы. Ограничим пока работу функции `go` простым `alert`-сообщением.

Меню работает, но выглядит не очень симпатично. Хочется, чтобы при "наезде" курсором на строки они инвертировали свой цвет. Давайте применим CSS! Будем отслеживать события **onmouseover** (курсор мыши появился над элементом) и **onmouseout** (курсор мыши ушел с элемента) и соответственно менять стиль этого элемента. Список теперь будет иметь вид:

```
<UL type=disc>
 <LI id=item0 onclick="go(0)"
     onmouseover="over(0)" onmouseout="out(0)">Раздел 0
 <LI id=item1 onclick="go(1)"
     onmouseover="over(1)" onmouseout="out(1)">Раздел 1
 <LI id=item2 onclick="go(2)"
     onmouseover="over(2)" onmouseout="out(2)">Раздел 2
</UL>
```

Функция `over` должна менять стиль строки списка таким образом, чтобы он был написан белым по черному, а функция `out` должна обеспечить обычный черный текст по белому фону.

Добраться до нужных свойств первой строки списка достаточно просто:

`document.all["item0"].style.color` — цвет текста
`document.all["item0"].style.background` — цвет фона

Функции `over` и `out` можно записать так:

```
function over(i)
{
 eval('document.all["item'+i+'"].style.color="white"');
 eval('document.all["item'+i+'"].style.background="black"');
}
function out(i)
{
 eval('document.all["item'+i+'"].style.color="black"');
 eval('document.all["item'+i+'"].style.background="white"');
}
```

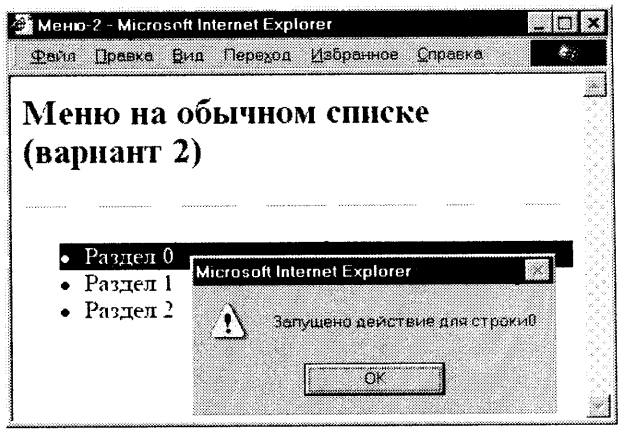
Как видите, для вычислений вызывается стандартная функция `eval`. Эта функция выполняет аргумент-строку как JavaScript-программу. Вот в эту строку мы и помещаем правильный идентификатор тега, который зависит от переменной `i`. Для `i` = 0 получится `"item0"`, для `i` = 1 — `"item1"`, для `i` = 2 — `"item2"`, то есть как раз то, что нам надо.

Определим для тега **LI** начальный стиль:

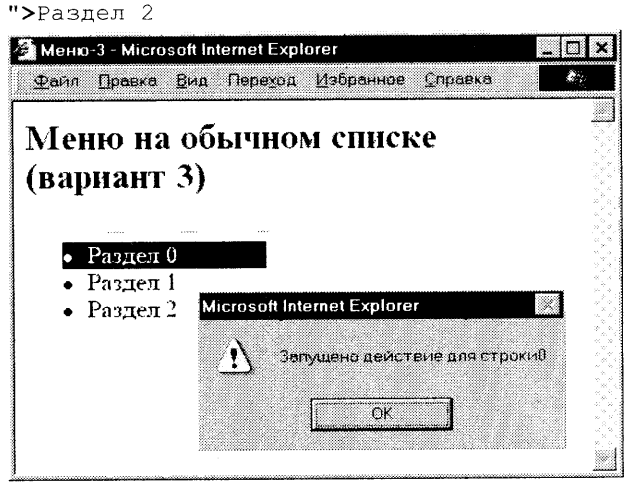
```
LI { color:black; background:white; }
```

И теперь можно посмотреть, что у нас получилось.

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Меню-2</TITLE>
  <STYLE type="text/css">
  <!--
   LI
```




```
  {
   color:black;
   background:white;
  }
 -->
 </STYLE>
 <SCRIPT language=JavaScript>
 <!--
  // Запуск действия, соответствующего
      i-й строке меню
   function go(i)
   {
     alert("Запущено действие для строки"+i);
   }
  // Курсор сместился над i-й строкой меню
  function over(i)
  {
   eval('document.all["item' + i + '"].style.color="white"');
   eval('document.all["item' + i + '"].style.background="black"');
  }
  // Курсор ушел с i-й строки меню
  function out(i)
  {
   eval('document.all["item' + i + '"].style.color="black"');
   eval('document.all["item' + i + '"].style.background="white"');
  }
 //-->
 </SCRIPT>
</HEAD>
<BODY bgcolor=white text=black onblur="window.close()">
  <H2>Меню на обычном списке (вариант 2)</H2>
  <HR>
  <UL type="disc">
    <LI id=item0 onclick="go(0)"
        onmouseover="over(0)" onmouseout="out(0)">Раздел 0
    <LI id=item1 onclick="go(1)"
        onmouseover="over(1)" onmouseout="out(1)">Раздел 1
    <LI id=item2 onclick="go(2)"
        onmouseover="over(2)" onmouseout="out(2)">Раздел 2
  </UL>
 </BODY>
</HTML>
```



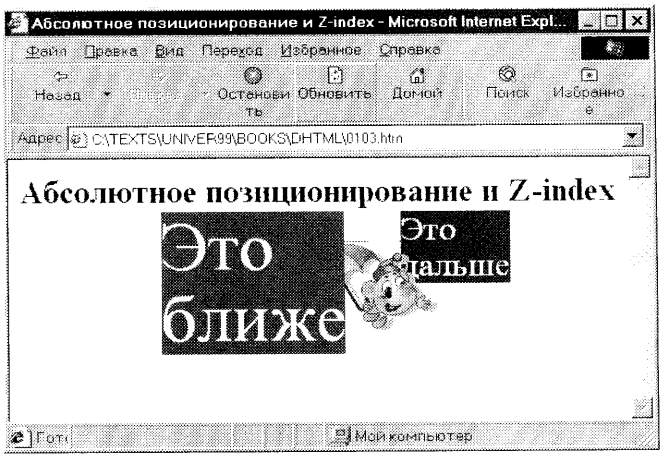
Меню работает! Но почему-то подсветка занимает всю ширину строки, а это не очень красиво. Так получается потому, что мы не задали в стиле ширину строк, а значит, она наследуется от родителя — тега **BODY** и тем самым равна ширине окна браузера. Попытка задать ширину в описании стиля для тега **LI** терпит неудачу. Дело в том, что стилевое свойство `width` поддерживается только тегами **DIV** и **SPAN**. Придется разместить наш список внутри тега **DIV**:

```
<DIV style="width:200px">
 <UL type=disc>
  <LI id=item0 onclick="go(0)"
      onmouseover="over(0)" onmouseout="out(0)">Раздел 0
  <LI id=item1 onclick="go(1)"
      onmouseover="over(1)" onmouseout="out(1)">Раздел 1
  <LI id=item2 onclick="go(2)"
      onmouseover="over(2)" onmouseout="out(2)">Раздел 2
 </UL>
</DIV>
```

И последний штрих: укажем в стиле для тега **LI** форму курсора такую, как на ссылке:

```
LI { color:black; background:white; cursor:hand}
```

Движение на экране



В следующем документе картинка начинает двигаться по экрану сразу после загрузки документа.

Это происходит потому, что в теге **BODY** размещен атрибут `onload`. После полной загрузки документа выполнится JavaScript-выражение, записанное в качестве значения этого атрибута: `"tim=setInterval('Move()',20)"`

Функция `setInterval('Move()',20)` создаст таймер, который будет запускать функцию `Move()` каждые 20 миллисекунд.

Функция `Move()` увеличивает горизонтальную координату картинки на два пикселя, если эта координата меньше 500. В противном случае функция останавливает таймер:

```
function Move()
{
  var obj = document.all["move"].style;
  if (obj.posLeft < 500) obj.posLeft += 2;
  else clearInterval(tim);
}
```

Замечание

Заметьте, что вместо стилевого свойства `left` используется свойство `posLeft`. Эти свойства совершенно идентичны, но первое хранится с окончанием `"px"` (например, `"200px"`) и тем самым неудобно для использования в арифметических вычислениях, второе — обычное число.

Вот полный код документа:

```
<HTML>
 <HEAD>
   <META http-equiv="Content-Type"
         content="text/html; charset=windows-1251">
   <TITLE>Абсолютное позиционирование и Z-index</TITLE>
   <SCRIPT language=JavaScript>
   <!--
     var tim; // таймер
     // Шаг перемещения по экрану
     function Move()
     {
      var obj = document.all["move"].style;
      if (obj.posLeft < 500) obj.posLeft += 2;
      else clearInterval(tim);
     }
   //-->
   </SCRIPT>
 </HEAD>
 <BODY bgcolor=white text=black
       onload="tim=setInterval('Move()',20)">
   <H2>Абсолютное позиционирование и Z-index</H2>
   <HR>
   <DIV style="position:absolute; top:50; left:140;
               height:130; width:100; background:red;
               font-size:60; color:white; z-index:3">
      Это ближе
   </DIV>
   <DIV style="position:absolute; top: 50; left:360;
                      height: 30; width:100;  background: blue;
                      font-size:30; color:white; z-index:1">
      Это дальше
   </DIV>
```

```
    <DIV id=move style="position:absolute; top:80; left:0;
         width:125; z-index:2">
      <IMG src=pic/explorer.gif width=125 height=82 alt="">
    </DIV>
  </BODY>
</HTML>
```

Задания

1. Создайте страничку, на которой по основному тексту "проплывает" небольшой фрагмент другого текста. Алгоритм движения: текст начинает двигаться слева направо; через некоторое время текст "скачком" возвращается в исходное положение слева и все повторяется.

2. Создайте страничку, на которой по основному тексту "проплывает" небольшой фрагмент другого текста. Алгоритм движения: текст начинает двигаться слева направо; через некоторое время текст меняет направление движения, доходит до исходного положения и все повторяется.

3. Создайте страничку, на которой по основному тексту "проплывает" небольшой фрагмент другого текста. Алгоритм движения: текст должен двигаться по синусоиде вправо и "скачком" возвращаться в исходное положение.

4. Создайте страничку, на которой по основному тексту "проплывает" небольшой фрагмент другого текста. Алгоритм движения: текст должен двигаться по синусоиде вправо, затем менять направление движения; в исходном положении все повторяется заново.

5. Создайте страничку, на которой по основному тексту "плавает" небольшой фрагмент другого текста. Управление движением — при помощи экранных кнопок.

6. Создайте страничку, на которой по основному тексту "плавает" небольшой фрагмент другого текста. Управление движением — при помощи кнопок клавиатуры.

7. Создайте страничку, на которой по основному тексту можно перетаскивать мышью картинку.

8. Создайте страничку, на которой кнопку "Нажми меня" преследует курсор мыши.

9. Создайте страничку, на которой кнопка "Догони меня" убегает от курсора мыши.

УРОК 6. УПРАВЛЕНИЕ СОДЕРЖИМЫМ СТРАНИЦЫ

Пользуясь свойствами innerText, outerText, innerHTML, outerHTML, можно динамически, то есть уже после построения страницы браузером, менять отдельные элементы на экране. Этими четырьмя свойствами обладают большинство видимых элементов.

Свойство innerText

Свойство имеет своим значением весь текст, содержащийся в элементе, включая содержимое всех наследников, без тегов HTML. Присвоение свойству нового текстового значения приводит к замене содержимого элемента на этот текст. При этом вся старая теговая структура внутри элемента пропадает, а новые теги, если они есть, воспринимаются как обычный текст.

Пусть элемент — это следующий абзац: Это **важное** слово

Его код имеет вид:

```
<P id=prim>Это <B>важное</B> слово.</P>
```

Тогда `alert(document.all["prim"].innerText)` выдаст значение "Это важное слово".

Получается, что свойство innerText "не видит" теговую разметку внутри элемента.

После присваивания:

```
document.all["prim"].innerText="Это <I>важное</I> слово.";
```

абзац станет таким: Это <I>важное</I> слово

Таким образом, теговая разметка в новом содержимом рассматривается как обычный текст.

Свойство outerText

Это свойство при чтении дает тот же результат, что и innerText. При записи нового значения оно работает аналогично, но заменяет не только внутренность элемента, но и теги, задающие элемент.

Пусть элементом является полужирный текст внутри абзаца: Это **важное** слово

Код этого фрагмента:

```
<P>Это <B id=prim>важное</B> слово.</P>
```

Команда `alert(document.all["prim"].outerText)` выдаст значение "важное".

После присваивания:

```
document.all["prim"].outerText="не важное";
```

абзац станет таким: Это не важное слово.

Выделение пропало. Как видите, свойство outerText позволяет заменять содержимое элемента вместе с задающими его тегами.

Если команда была бы такой

```
document.all["prim"].innerText="не важное";
```

выделение сохранилось бы: Это **не важное** слово

Свойство innerHTML

Это свойство имеет значением весь код, содержащийся внутри элемента.

Пусть элемент — это следующий абзац: Это **важное** слово.

Его код имеет вид:

```
<P id=prim>Это <B>важное</B> слово.</P>
```

Тогда `alert(document.all["prim"].innerHTML)` выдаст значение "Это <B>важное</B> слово".

После присваивания:

```
document.all["prim"].innerHTML="Это <I>важное</I> слово.";
```

абзац станет таким: Это *важное* слово.

Как видите, теги, записанные в новом значении, корректно работают.

Свойство outerHTML

Отличие от innerHTML только в том, что в расчет дополнительно берутся начальный и конечный теги, задающие элемент.

Пусть элементом является полужирный текст внутри абзаца: Это **важное** слово

Код этого фрагмента:

```
<P>Это <B id=prim>важное</B> слово.</P>
```

Команда `alert(document.all["prim"].outerHTML)` выдаст значение "<B id=prim>важное</B>".
После присваивания:

```
document.all["prim"].outerHTML="<I id=prim>не важное</I>";
```

абзац станет таким: Это *не важное* слово

Крестики-нолики

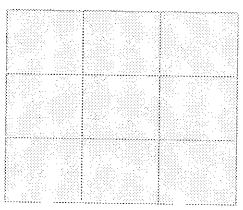
Посмотрим, как можно применить описанные свойства при проектировании игры "Крестики-нолики". Игровое поле построим при помощи таблицы. Для пустых клеток зададим стиль:

```
.empty /* Стиль пустой клетки */
{
  background-color:#EEE5DB;
  color:#EEE5DB;
  width:50; height:60;
  text-align:center;
}
```

Для клеток, в которых будут записаны ходы, опишем стиль:

```
.busy  /* Стиль занятой клетки */
{
  background-color:#D1B284;
  width:50; height:60;
  text-align:center;
}
```

Первоначально таблица будет иметь вид:



Вот ее код:

```
<TABLE border=1 cellspacing=0 cellpadding=10>
  <TR>
    <TD class=empty id=game00> </TD>
    <TD class=empty id=game01> </TD>
    <TD class=empty id=game02> </TD>
  </TR>
  <TR>
    <TD class=empty id=game10> </TD>
    <TD class=empty id=game11> </TD>
    <TD class=empty id=game12> </TD>
  </TR>
  <TR>
    <TD class=empty id=game20> </TD>
    <TD class=empty id=game21> </TD>
    <TD class=empty id=game22> </TD>
  </TR>
</TABLE>
```

Построить таблицу можно и программным путем при помощи такой функции:

```
var nameId = "game"; // Основа идентификатора клетки таблицы
// Построение игровой таблицы
function ShowRU()
{
 var str="<TABLE border=1 cellspacing=0 cellpadding=10>";
 for (var i = 0; i < 3; i++)
 {
  str += "<TR>";
  for (var j = 0; j < 3; j++)
    str += "<TD class=empty id="+nameId+i+j+"> </TD>";
  str += "</TR>";
 }
 str += "</TABLE>";
 document.write(str);
}
```

Щелчок мыши будем обрабатывать такой функцией:

```
var step = "0"; // Переключатель хода

// Обработка щелчка мыши
function ClickRU()
{
 var obj = event.srcElement;    // Тег, на котором щелкнули
 if (obj.className == "empty") // Щелчок по пустой клетке
 {
   obj.className="busy";        // Изменим стиль клетки
   step = step == "X" ? "0":"X";
   obj.innerText = step;        // Запишем ход игрока
 }
}
```

На рисунке показано, что из этого получилось.

Полный код этого примера:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type"
        content="text/html; charset=windows-1251">
  <TITLE>Крестики-нолики</TITLE>
  <STYLE type="text/css">
  <!--
    .empty /* Стиль пустой клетки */
    {
      background-color:#EEE5DB;
      color:#EEE5DB;
      width:50; height:60;
      text-align:center;
    }
    .busy  /* Стиль занятой клетки */
    {
      background-color:#D1B284;
      width:50; height:60;
      text-align:center;
    }
   -->
   </STYLE>
   <SCRIPT language=JavaScript>
   <!--
    var nameId = "game"; // Основа идентификатора клетки таблицы
    var step   = "0";    // Переключатель хода
    // Построение игровой таблицы
    function ShowRU()
    {
      var str="<TABLE border=1 cellspacing=0 cellpadding=10>";
      for (var i = 0; i < 3; i++)
```

```
    {
      str += "<TR>";
      for (var j = 0; j < 3; j++)
        str += "<TD class=empty id="+nameId+i+j+"> </TD>";
      str += "</TR>";
    }
    str += "</TABLE>";
    document.write(str);
  }
  // Обработка щелчка мыши
  function ClickRU()
  {
    var obj = event.srcElement;      // Тег, на котором щелкнули
    if (obj.className == "empty")    // Если щелчок по пустой клетке
    {
      obj.className="busy";          // Изменим стиль клетки
      step = step == "X" ? "0":"X";
      obj.innerText = step;          // Запишем ход игрока
    }
  }
  // Очистка игровой таблицы
  function ClearRU()
  {
    for (var i = 0; i < 3; i++)
      for (var j = 0; j < 3; j++)
      {
        var obj;
        eval("obj=document.all['"+nameId+i+j+"']");
        obj.className="empty";
        obj.innerText="n";
      }
    step = "0";
  }
 //-->
 </SCRIPT>
</HEAD>
<BODY onclick="ClickRU()" bgcolor=white text=black>
 <H2>Крестики-нолики</H2>
 <SCRIPT language=JavaScript>
 <!--
   ShowRU(); // Нарисуем игровую таблицу
 //-->
 </SCRIPT>
 <FORM>
  <INPUT type=button value="Очистка" onclick="ClearRU()">
 </FORM>
</BODY>
</HTML>
```

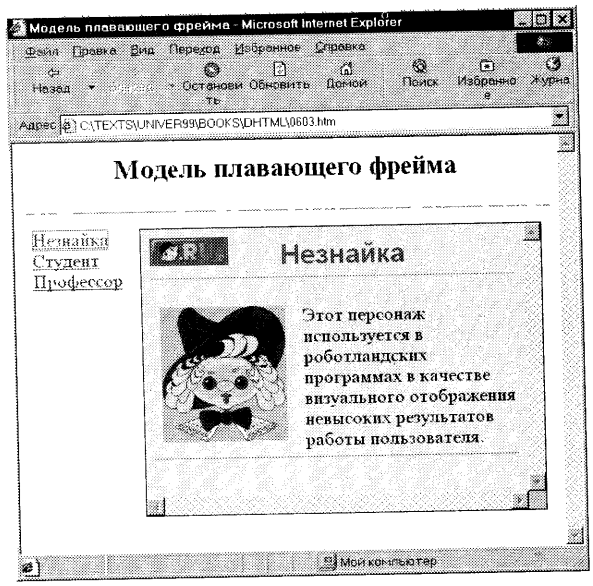
Чтобы не загромождать учебный код, программа не проверяет окончание игры, но такую проверку легко добавить, и в ней пригодятся неиспользуемые пока идентификаторы клеток таблицы.

Игра будет выглядеть более симпатично, если внутри таблицы расставлять не текстовые, а графические пометки.

Щелчок мыши теперь будет обрабатываться такой функцией:

```
// Код для вывода картинок в табличные клетки
var PicX="<IMG src=./pic/gx.gif width=29 height=31
                                border=0 alt=''>";
var Pic0="<IMG src=./pic/g0.gif width=29 height=31
                                border=0 alt=''>";
var step="0"; // Переключатель хода
```

```
// Обработка щелчка мыши
function ClickRU()
{
 var obj = event.srcElement;  // Тег, на котором щелкнули
 if (obj.className == "empty")// Щелчок по пустой клетке
 {
   obj.className = "busy";    // Изменить стиль клетки
   if (step == "X")
   {                          // Шаг игрока "0"
     step="0";
     obj.innerHTML = Pic0;
   }
   else
   {                          // Шаг игрока "X"
     step="X";
     obj.innerHTML = PicX;
   }
 }
}
```



Модель плавающего фрейма

Рассмотрим еще один пример динамического изменения содержимого гипертекстовой страницы.

Это приложение имеет следующую структуру:

```
<HTML>
 <HEAD>
  ...
 </HEAD>
 <BODY bgcolor=white text=black
       link=blue alink=red vlink=purple
       onload="setRU(0)">
  <H2 align=center>Модель плавающего фрейма</H2>
  <HR>
  <TABLE border=0 cellspacing=0 cellpadding=10
         width=100%>
    <TR>
      <TD valign=top align=left>
        <A href="javascript:void(0)"
           onclick="setRU(0)">Незнайка</A><BR>
        <A href="javascript:void(0)"
           onclick="setRU(1)">Студент</A><BR>
        <A href="javascript:void(0)"
           onclick="setRU(2)">Профессор</A><BR>
      </TD>
      <TD valign=top align=right>
        <SPAN id=frame
              style="position:relative;
                     width:450; height:320;
                     overflow:scroll;
                     text-align:left;
                     padding:10;
                     border: 1px solid black;
                     background:#EEE5DB;">
          nbsp;
        </SPAN>
      </TD>
    </TR>
  </TABLE>
  <DIV id=frame0 style="display:none">
    ...
  </DIV>
  <DIV id=frame1 style="display:none">
    ...
```

```
  </DIV>
  <DIV id=frame2 style="display:none">
    ...
  </DIV>
 </BODY>
</HTML>
```

В первой ячейке таблицы располагается гипертекстовое меню, а во второй — модель (пустого) фрейма. Загрузка гипертекстового содержимого в область-фрейм выполняется при помощи функции `setRU`:

```
// Базовое имя фрейма и областей загрузки
var nameFrame = "frame";
// Загрузка содержимого области n во фрейм
function setRU(n)
{
  eval("document.all['"+nameFrame+
       "'].innerHTML=document.all['"+
       nameFrame+n+"'].innerHTML");
}
```

При `n` = 0 команда внутри функции равнозначна такой:

```
document.all['frame'].innerHTML = document.all['frame0'].innerHTML;
```

То есть во фрейм (тег **SPAN** внутри таблицы) загружается содержимое области с идентификатором `frame0`. Именно такая команда срабатывает после загрузки документа (**onload**="`setRU(0);`" в теге **BODY**).

Гипертекстовые ссылки:

```
<A href="javascript:void(0)" onclick="setRU(0)">Незнайка</A><BR>
<A href="javascript:void(0)" onclick="setRU(1)">Студент</A><BR>
<A href="javascript:void(0)" onclick="setRU(2)">Профессор</A><BR>
```

устроены так, что вместо обычного перехода записан фиктивный `javascript:void(0)`, а щелчок обрабатывается функцией `setRU` с соответствующим параметром. Обращение `setRU(0)` загружает во "фрейм" область `frame0` (про Незнайку), обращение `setRU(1)` — область `frame1` (про Студента) и обращение `setRU(2)` — область `frame2` (про Профессора).

Для того чтобы области `frame0`, `frame1` и `frame2` не были видны на экране, к блокам, которые их содержат, применен стиль `display:none` (скрытие элемента).

Задания

1. Доработайте приложение "Крестики-нолики". Напишите краткую инструкцию по игре, разработайте подходящий дизайн, а главное, напишите функцию для проверки окончания игры.

Как проверить, например, не стоит ли (в текстовом варианте игры) символ "X" в первой ячейке таблицы? Доступ к ячейке запишется так: `document.all[game00]`, а проверка на "X" будет иметь вид:

```
document.all[game00].innerText == "X".
```

2. Постройте на основе "Модели плавающего фрейма" собственное приложение.

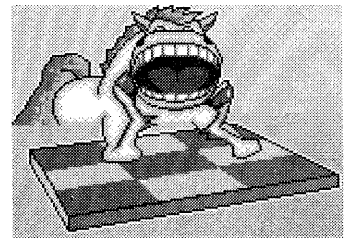
3. Разработайте приложение — версию игры "15". Игра происходит в таблице (3 × 5). В клетках таблицы расположены 14 фишек с числами 1, 2, ..., 14. В одной ячейке может располагаться только одна фишка. Фишку можно перемещать в соседнюю клетку, если она пуста. Игра заканчивается, когда фишки занимают свои места в порядке возрастания чисел на них:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	

Предусмотрите автоматическую генерацию начального состояния игры, контроль числа ходов, проверку на окончание игры. Фишка должна перемещаться в соседнюю пустую клетку щелчком мыши.

Приложение будет более привлекательным, если фишки выполнить в графике.

РАЗДЕЛ 2. "КУХНЯ" СИДОРОВА

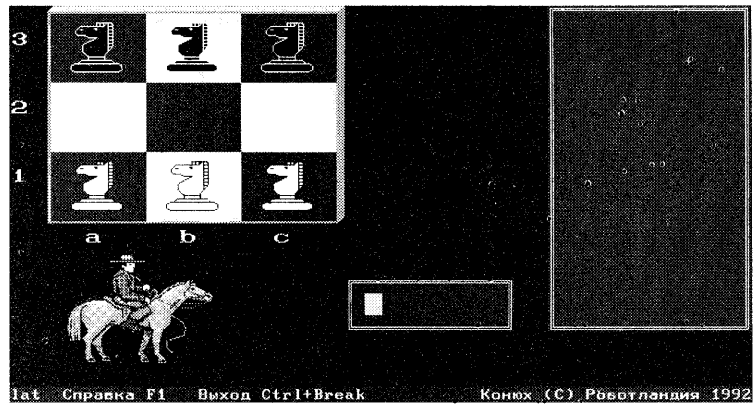


УРОК 7. КОНЮХ (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

Описание задачи

В роботландском курсе информатики для младших школьников в разделе "Алгоритмические этюды" есть программа Конюх. Назначение программ этой серии — закрепление на практике понятий "исполнитель", "среда исполнителя", "система команд", "алгоритм и программа", "аварийные сообщения исполнителя".

Программа Конюх предлагает обучаемому решить две задачи на маленькой шахматной доске из 9 клеток. В начальный момент черные кони расположены на верхней горизонтали, а белые — на нижней. Требуется переставить фигуры местами так, чтобы белые кони заняли верхний ряд, а черные — нижний. При этом кони перемещаются обычным шахматным прыжком по букве "Г". В первой задаче на поле 6 коней (она проще), во второй — 4 (она сложнее).



На рисунке показано, как выглядит роботландская программа перед решением задачи с шестью конями.

На экране расположены три области:

1. Среда исполнителя (фрагмент шахматной доски из 9 клеток с фигурками коней).
2. Поле для ввода команд исполнителю (в нем виден текстовый курсор).
3. Поле, в котором отображается протокол работы с исполнителем (программа).

Команда исполнителю формируется по правилам записи шахматных ходов. Например, ход белого коня из клетки `a1` в клетку `c2` задается командой: `a1—c2`.

Сидоров получил задание разработать Конюха в гипертекстовом виде для нового роботландского интерактивного учебника "Азы информатики".

Понятно, что работу он начал с составления технического задания.

Техническое задание

Требуется создать программу (экранную среду) для решения задач Конюха.

Предполагается, что пользоваться программой будут школьники 1—5-х классов в рамках начального курса информатики (раздел "Алгоритмические этюды"). Круг пользователей и учебная цель проектируемой программы накладывают на нее следующие требования:

— Программа должна быть визуально привлекательной для младшего школьника.

— Программа должна работать в среде браузера IE, начиная с четвертой версии, и отображаться без линеек прокрутки в окне 640 × 480.

Экран программы должен содержать:

1. Среду исполнителя (шахматная доска и фигуры коней).
2. Поле ввода команды с группой экранных кнопок для:
   — Ввода команды.
   — Откатки.
   — Накатки.
   — Сброса среды (откатки в начало).
3. Поле протокола (для демонстрации программы работы исполнителя).
4. Блок экранных кнопок для:
   — Выполнения настроек программы:
     • число коней (6 или 4; 6 по умолчанию);
     • режим решения (визуальный или командный; командный по умолчанию).
   — Показа условия задачи.
   — Показа пользовательских инструкций.
   — Выхода из программы.

Среда должна позволять:

— Решать задачу в командном режиме при помощи записи текстовой команды в поле ввода.

— Решать задачу в визуальном режиме путем перетаскивания коней по экрану в нужную клетку поля.

— Предоставлять возможность выполнять полную откатку и накатку выполненных изменений среды исполнителя во время решения задачи.

— Выводить на экран диагностику при неверных действиях пользователя: "Не понимаю" — задана неверная команда; "Не могу" — команду выполнить невозможно.

Среда должна иметь контекстно-зависимые подсказки.

Среда должна информировать пользователя об окончании решения задачи.

Наряду с интерфейсом мыши (экранные кнопки) должны работать "горячие" клавиатурные клавиши.

Построение технического задания Сидоров завершил рисунком, на котором изобразил макет внешнего вида программы (см. рисунок).

Экран Конюха содержит:

— Название программы, которое будет сопровождаться рисунком исполнителя.

— Блок кнопок для:

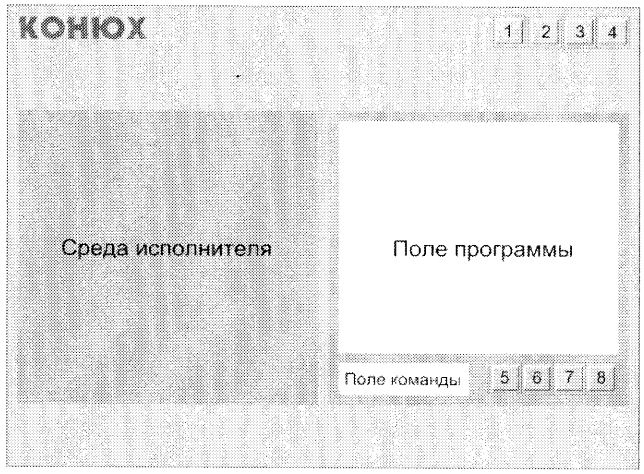
1. Установки режимов работы программы.
2. Показа условия задачи.
3. Показа пользовательских инструкций.
4. Выхода из программы.

— Среду исполнителя.

— Панель управления исполнителем, содержащую:

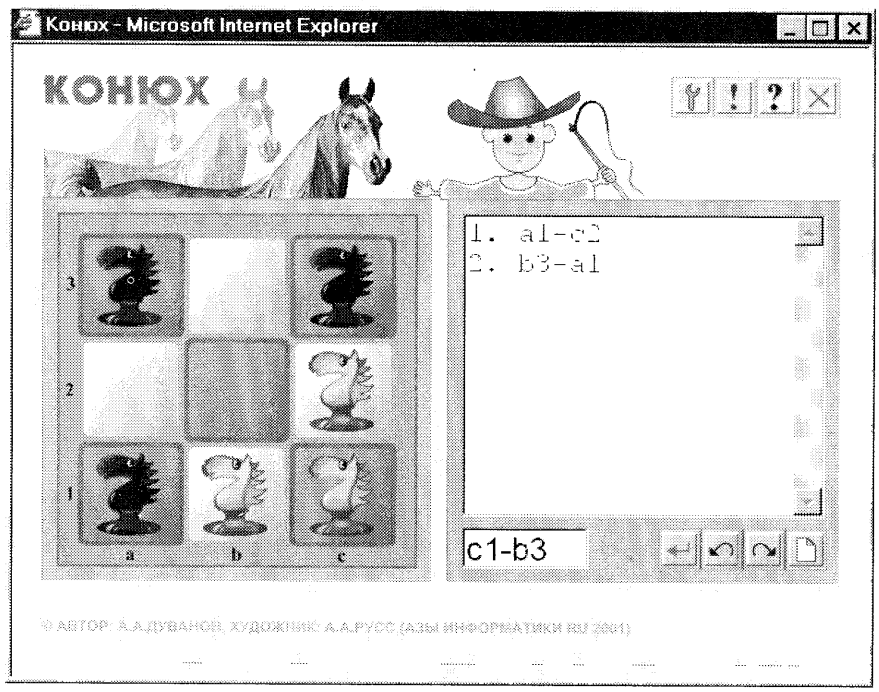
- Поле ввода команды.
- Поле программы.
- Кнопки управления:

5. Ввод команды.
6. Откатка.
7. Накатка.
8. Сброс (откатка в начало).



Готовый продукт

Прошло около недели, и Сидоров принес в Роботландию готовый продукт.



УРОК 8. КОНЮХ (СТИЛЕВЫЕ РЕШЕНИЯ)

Программа Конюх будет вызываться со страниц учебника "Азы информатики" для работы в отдельном окне. Положим размеры этого окна 610 × 460 — это (с небольшим запасом) обеспечит показ программы без линеек прокруток на мониторах с разрешением 640 × 480.

Ниже представлен пример вызова Конюха при помощи стандартной кнопки **<INPUT type**=button**>**:

```
<FORM>
 <INPUT type=button value=Конюх
  onclick="window.open('./horse/horse.htm','',
  'width=610,height=460,menubar=0,resizable=0,scrollbars=0'
                        )">
</FORM>
```

Конюх

Используя CSS-позиционирование элементов, построим жесткий экранный каркас программы, который не будет форматироваться браузером. Все экранные элементы поместим внутрь "выпуклого" прямоугольника, который зададим следующим стилевым определением:

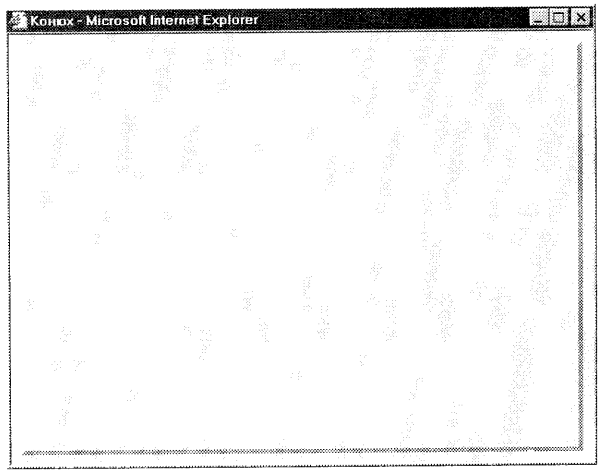
```
.program /* Стиль основной области */
{
  position:absolute;
  left:10px; top:10px;
  width:590px; height:440px;

  /* "Выпуклая" граница */
  border-style:        solid;
  border-width:        1px;
  border-left-color:   white;
  border-top-color:    white;
  border-right-color:  gray;
  border-bottom-color: gray;
}
```

Размеры прямоугольника — 590 × 440, что вместе с отступами по краям в 10 пикселей как раз дает размер окна 610 × 460. В HTML-файле Конюха запишем такой код:

```
<BODY bgcolor=#EEE5DB text=black>
 <DIV class=program id=program title="Исполнитель Конюх">
 </DIV>
</BODY>
```

На рисунке показано, как это будет выглядеть на странице браузера.



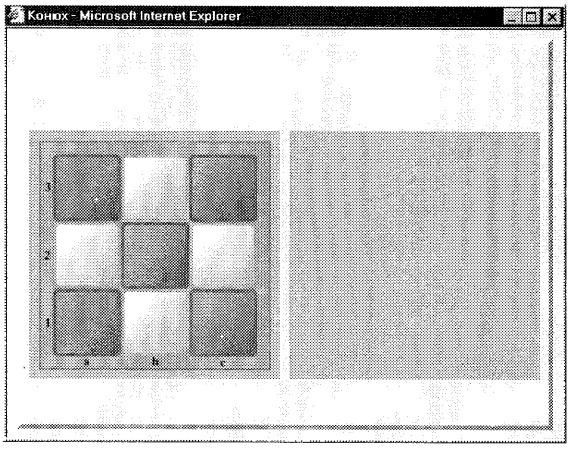
Теперь зададим стили для среды исполнителя и панели управления:

```
.sreda /* Стиль среды исполнителя */
{
  position:absolute;
  left: 10px; top:100px;
  width:280px; height:280px;
  padding: 10px;
  /* "Вдавленная" граница */
  background-color:    #D1B284;
  border-style:        solid;
  border-width:        1px;
  border-left-color:   gray;
  border-top-color:    gray;
  border-right-color:  white;
  border-bottom-color: white;
}
```

```
.control /* Стиль панели управления */
{
  position:absolute;
  left: 300px; top:100px;
  width:280px; height:280px;
  padding: 10px;
  /* "Вдавленная" граница */
  background-color:    #D1B284;
  border-style:        solid;
  border-width:        1px;
  border-left-color:   gray;
  border-top-color:    gray;
  border-right-color:  white;
  border-bottom-color: white;
}
```

Эти две области будут вложены в основную область program, поэтому отсчет координат в них выполняется от левого верхнего угла program:

```
<BODY bgcolor=#EEE5DB text=black>
 <DIV class=program id=program title="Исполнитель Конюх">
   <SPAN class=sreda id=sreda title="Среда исполнителя"
     ><IMG src=./pic/horsef.gif width=258 height=258
           hspace=0 vspace=0 border=0
           alt="Среда исполнителя"
           title="Среда исполнителя"
   ></SPAN>
   <SPAN class=control title="Панель управления">
   </SPAN>
 </DIV>
</BODY>
```



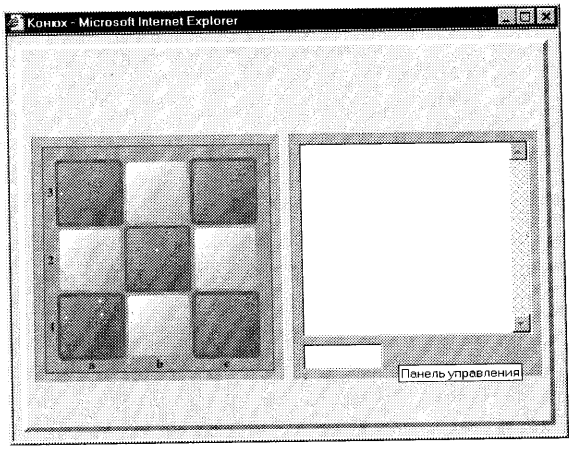
Зададим стили для поля программы (протокола) и поля ввода команд:

```
.list /* Стиль протокола */
{
  position:absolute;
  left: 10px; top:10px;
  width:258px; height:220px;
  font-size:12pt;
}
.com /* Стиль поля ввода */
{
  position:absolute;
  left: 10px; top:238px;
  width:90px; height:30px;
  font-size:14pt;
  font-family:"Arial Cyr",Geneva,Helvetica,sans-serif;
}
```

Поле протокола и поле ввода будут размещаться в области control, поэтому координаты полей будут отсчитываться от левого верхнего угла области control:

```
<BODY bgcolor=#EEE5DB text=black>
 <DIV class=program id=program title="Исполнитель Конюх">
   <SPAN class=sreda id=sreda title="Среда исполнителя"
     ><IMG src=./pic/horsef.gif width=258 height=258
           hspace=0 vspace=0 border=0
           alt="Среда исполнителя"
           title="Среда исполнителя"
   ></SPAN>
   <SPAN class=control title="Панель управления">
    <FORM name=forma>
      <TEXTAREA class=list name=list cols=20 rows=10
                title="Программа" readonly>
      </TEXTAREA>
```

```
      <INPUT class=com name=com type=text size=7
             maxlength=5 title="Поле ввода команды">
    </FORM>
   </SPAN>
  </DIV>
 </BODY>
```



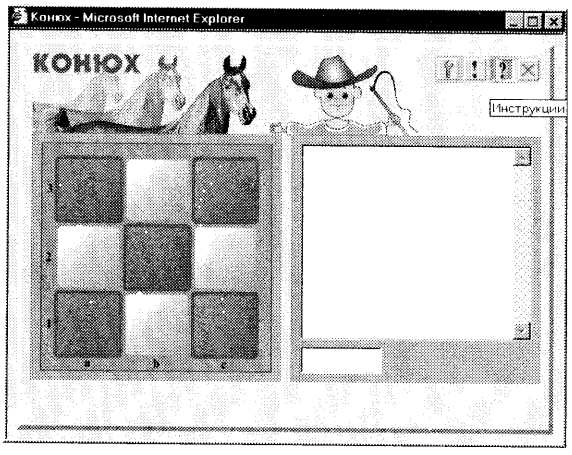
Введем область programUp — верхнюю часть экранного образа программы. В нее вложим картинку с изображением исполнителя и область programKey с кнопками 1—4. Новые стилевые определения выглядят так:

```
/* Стиль верхней части программной области */
.programUp
{
  position:absolute;
  left: 10px; top:10px;
}
/* Стиль блока с кнопками в верхней части */
.programKey
{
  position:absolute;
  left: 449px;
}
```

"Чувствительность" экранных кнопок будем имитировать подменой изображения при перемещении курсора на кнопку и уходе с нее (события mouseover и mouseout):

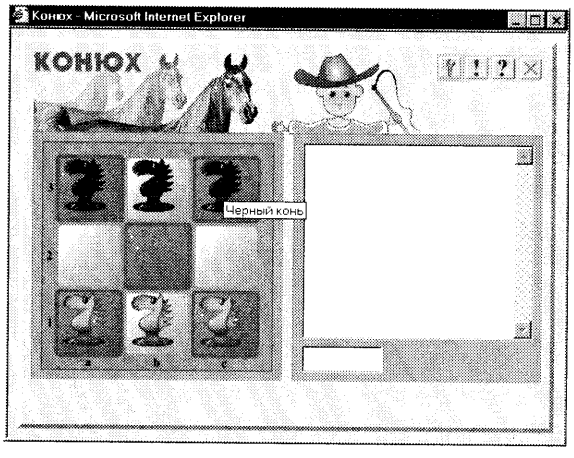
```
<BODY bgcolor=#EEE5DB text=black>
 <DIV class=program id=program title="Исполнитель Конюх">
  <SPAN class=programUp>
    <IMG src=./pic/horser.gif width=430 height=91
      border=0 hspace=0 vspace=0 alt="Исполнитель Конюх"
      title="Исполнитель Конюх">
    <SPAN class=programKey>
     <NOBR><IMG src=./pic/key1.gif width=30 height=30
       border=0 hspace=0 vspace=0
       alt="Настройка программы"
       title="Настройка программы"
       onmouseover="this.src='./pic/key1_.gif'"
       onmouseout="this.src='./pic/key1.gif'"
     ><IMG src=./pic/key2.gif width=30 height=30
       border=0 hspace=0 vspace=0
       alt="Постановка задачи" title="Постановка задачи"
       onmouseover="this.src='./pic/key2_.gif'"
       onmouseout="this.src='./pic/key2.gif'"
     ><IMG src=./pic/key3.gif width=30 height=30
       border=0 hspace=0 vspace=0
       alt="Инструкции" title="Инструкции"
       onmouseover="this.src='./pic/key3_.gif'"
       onmouseout="this.src='./pic/key3.gif'"
     ><IMG src=./pic/key4.gif width=30 height=30
       border=0 hspace=0 vspace=0
       alt="Выход" title="Выход"
       onmouseover="this.src='./pic/key4_.gif'"
       onmouseout="this.src='./pic/key4.gif'"
     ></NOBR>
    </SPAN>
   <SPAN class=sreda id=sreda title="Среда исполнителя"
     ><IMG src=./pic/horsef.gif width=258 height=258
        hspace=0 vspace=0 border=0
        alt="Среда исполнителя" title="Среда исполнителя"
   ></SPAN>
   <SPAN class=control title="Панель управления">
    <FORM name=forma>
      <TEXTAREA class=list name=list cols=20 rows=10
                title="Программа" readonly>
      </TEXTAREA>
```

```
      <INPUT class=com name=com type=text size=7
             maxlength=5 title="Поле ввода команды">
    </FORM>
   </SPAN>
  </DIV>
 </BODY>
```



В среде исполнителя разместим фигурки коней. Было бы логично поместить соответствующие HTML-коды внутрь блока sreda, но приходится располагать их "снаружи". Дело в том, что браузер IE-5 интерпретирует координаты элемента при его перетаскивании по экрану (а коней хочется перемещать мышкой!) совсем не так, как браузер IE-4, когда элемент позиционируется не относительно блока **BODY**. Координаты коней отсчитываются от левого верхнего угла рабочего поля окна (элемента **BODY**).

```
<IMG style="position:absolute;left:60px;top:144px;"
     src=./pic/horseb.gif
     id=horse00 width=47 height=61 border=0
     alt="Черный конь" title="Черный конь">
<IMG style="position:absolute;left:136px;top:144px;"
     src=./pic/horseb.gif
     id=horse10 width=47 height=61 border=0
     alt="Черный конь" title="Черный конь">
<IMG style="position:absolute;left:212px;top:144px;"
     src=./pic/horseb.gif
     id=horse20 width=47 height=61 border=0
     alt="Черный конь" title="Черный конь">
<IMG style="position:absolute;left:60px;top:296px;"
     src=./pic/horsew.gif
     id=horse02 width=47 height=61 border=0
     alt="Белый конь" title="Белый конь">
<IMG style="position:absolute;left:136px;top:296px;"
     src=./pic/horsew.gif
     alt="Белый конь" title="Белый конь"
     id=horse12 width=47 height=61 border=0
     alt="Белый конь" title="Белый конь">
<IMG style="position:absolute;left:212px;top:296px;"
     src=./pic/horsew.gif
     id=horse22 width=47 height=61 border=0
     alt="Белый конь" title="Белый конь">
```



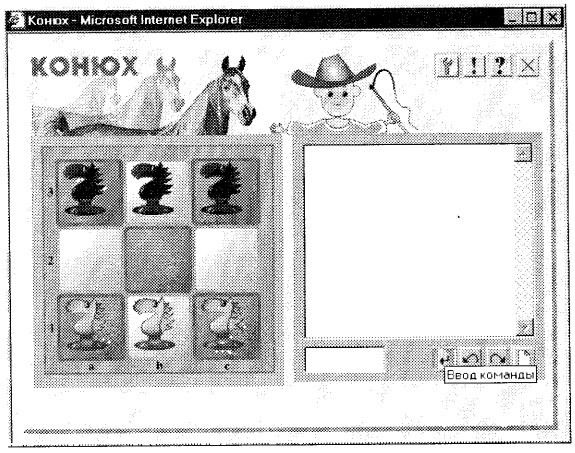
Теперь осталось совсем немного. Определим стиль блока с кнопками на панели управления:

```
/* Стиль кнопок на панели управления */
.controlKey
{
  position:absolute;
  left:150px; top:238px;
  width:160px;
}
```

И разместим в этом блоке сами кнопки:

```
<BODY bgcolor=#EEE5DB text=black>
 <DIV class=program id=program title="Исполнитель Конюх">
   <SPAN class=programUp>
     ...
     <SPAN class=programKey>
       ...
     </SPAN>
   </SPAN>
   <SPAN class=sreda id=sreda title="Среда исполнителя"
     ...
   </SPAN>
```

```
  <SPAN class=control title="Панель управления">
   <FORM name=forma>
     ...
   </FORM>
   <SPAN class=controlKey>
     <NOBR><IMG src=./pic/key5.gif width=30 height=30
          border=0 hspace=0 vspace=0 alt="Ввод команды"
          title="Ввод команды"
          onmouseover="this.src='./pic/key5_.gif'"
          onmouseout="this.src='./pic/key5.gif'"
     ><IMG src=./pic/key6.gif width=30 height=30
          border=0 hspace=0 vspace=0 alt="Откатка"
          title="Откатка"
          onmouseover="this.src='./pic/key6_.gif'"
          onmouseout="this.src='./pic/key6.gif'"
     ><IMG src=./pic/key7.gif width=30 height=30
          border=0 hspace=0 vspace=0 alt="Накатка"
          title="Накатка"
          onmouseover="this.src='./pic/key7_.gif'"
          onmouseout="this.src='./pic/key7.gif'"
     ><IMG src=./pic/key8.gif width=30 height=30
          border=0 hspace=0 vspace=0 alt="Сброс"
          title="Сброс"
          onmouseover="this.src='./pic/key8_.gif'"
          onmouseout="this.src='./pic/key8.gif'"
     ></NOBR>
    </SPAN>
  </SPAN>
 </DIV>
</BODY>
```

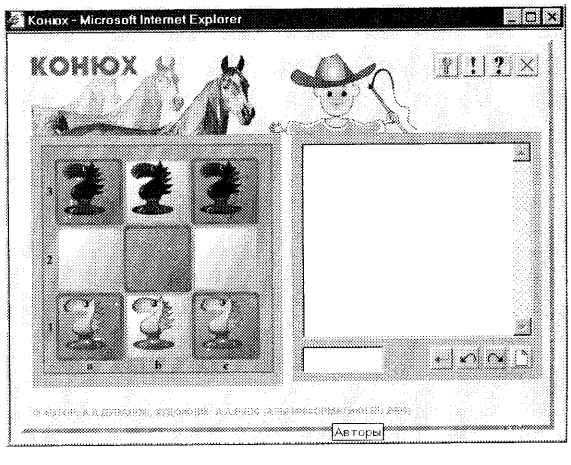


Последний штрих: добавим картинку-строку с авторской подписью. Стиль новой области определим так:

```
/* Стиль нижней части программной области */
.programDown
{
  position:absolute;
  left: 10px; top:400px;
}
```

Картинку расположим внутри этой области:

```
<BODY bgcolor=#EEE5DB text=black>
 <DIV class=program id=program title="Исполнитель Конюх">
   <SPAN class=programUp>
     ...
     <SPAN class=programKey>
       ...
     </SPAN>
   </SPAN>
   <SPAN class=sreda id=sreda title="Среда исполнителя">
     ...
   </SPAN>
   <SPAN class=control title="Панель управления">
     ...
   </SPAN>
   <SPAN class=programDown>
     <IMG src=./pic/autor.gif width=421 height=11
          border=0 alt="Авторы" title="Авторы">
   </SPAN>
 </DIV>
</BODY>
```



Экранный образ программы практически построен, но это просто изображение, которое никак не реагирует ни на мышиные, ни на клавиатурные воздействия. Изображение превратится в настоящую программу тогда, когда будут написаны коды скриптов, обрабатывающие мышиные и клавиатурные события. Некоторые поучительные фрагменты этих кодов будут рассмотрены в следующем уроке.

УРОК 9. КОНЮХ (СКРИПТЫ)

Перетаскивание элементов по экрану

Перетаскивание экранного элемента в графических системах пользователь привык выполнять так:

— Нажать кнопку мыши на элементе (захват элемента, событие **mousedown**).

— Не отпуская нажатую кнопку, перемещать курсор мыши, а вместе с ним и элемент по экрану (перемещение элемента, событие **mousemove**).

— Отпустить кнопку мыши в нужном месте экрана (освобождение элемента, событие **mouseup**).

Если нужно перемещать по экрану, например, картинку **IMG**, то вызов собственных обработчиков событий можно вмонтировать в этот тег:

```
<IMG ...
     onmousedown = "..."
     onmousemove = "..."
     onmouseup   = "...">
```

Когда перемещаемых элементов много, удобнее вызов обработчиков помещать не в тегах, а прямо в объекте `document`:

```
// Назначение обработчиков для событий,
// связанных с перетаскиванием
document.onmousedown = mouseDownEvent;
document.onmousemove = mouseMoveEvent;
document.onmouseup   = mouseUpEvent;
// Обработка захвата элемента
// --------------------------
function mouseDownEvent()
{
  ...
}
// Обработка перемещения элемента
// ------------------------------
function mouseMoveEvent()
{
  ...
}
// Обработка освобождения элемента
// -------------------------------
function mouseUpEvent()
{
  ...
}
```

Свойства `onmousedown`, `onmousemove`, `onmouseup` объекта `document` содержат ссылки на функции, которые обрабатывают соответствующие события.

Команды:

```
document.onmousedown = mouseDownEvent;
document.onmousemove = mouseMoveEvent;
document.onmouseup   = mouseUpEvent;
```

записывают в эти свойства ссылки на функции `mouseDownEvent()`, `mouseMoveEvent()`, `mouseUpEvent()`, которые предстоит запрограммировать.

Пусть, например, нам нужно перемещать по экрану любую картинку (тег **IMG**). Когда пользователь "захватывает" элемент, управление передается в функцию `mouseDownEvent()`. Нужно проверить, что элемент действительно является картинкой, и запомнить ссылку на объект, построенный браузером для этого элемента:

```
// Указатель на перетаскиваемый элемент или null
var imgDragging = null;
// Обработка захвата элемента
// --------------------------
function mouseDownEvent()
{
  // Это картинка?
  if (event.srcElement.tagName == "IMG")
  {
    // Запомним указатель на объект
    imgDragging = event.srcElement;
```

```
    }
    // Перетаскивание запрещено
    else imgDragging = null;
  }
```

На самом деле в момент захвата элемента нужно сделать еще две вещи: запомнить координаты курсора мыши относительно левого верхнего угла перемещаемого элемента и "приподнять" элемент над основным слоем (чтобы он перемещался "поверх" всех других элементов экрана):

```
// Указатель на перетаскиваемый элемент или null
var imgDragging = null;
// Положение курсора относительно элемента
// в момент начала перетаскивания
var dx;
var dy;
// Обработка захвата элемента
// --------------------------
function mouseDownEvent()
{
  // Это картинка?
  if (event.srcElement.tagName == "IMG")
  {
    // Запомним указатель на объект
    imgDragging = event.srcElement;
    // Вычислим смещения курсора мыши
    // относительно начала перетаскиваемого элемента
    dx = event.offsetX;
    dy = event.offsetY;
    // Передвинем элемент в верхний слой
    imgDragging.style.zIndex = 1;
  }
  // Перетаскивание запрещено
  else imgDragging = null;
}
```

Вот теперь все подготовлено к перемещению элемента по экрану! Обработка события `mousemove` может быть такой:

```
// Обработка перемещения элемента
// ------------------------------
function mouseMoveEvent()
{
  if (imgDragging != null)
  {
    // Вычислим новые координаты элемента
    imgDragging.style.pixelLeft = event.x - dx;
    imgDragging.style.pixelTop  = event.y - dy;
    // Остановим прохождение события
    event.cancelBubble = true;
    event.returnValue = false;
  }
}
```

Последние две команды запрещают браузеру обрабатывать событие `mousemove` стандартным образом (стандартная обработка помешает работе наших кодов). Теперь осталось написать обработчик события `mouseup` (пользователь отпустил кнопку мыши). Код этого обработчика крайне прост: в нем устанавливается флаг запрета дальнейшего перемещения, и элемент возвращается в основной экранный слой.

```
// Обработка освобождения элемента
// -------------------------------
function mouseUpEvent()
{
  if (imgDragging != null)
  {
    // Вернем элемент в основной слой
    imgDragging.style.zIndex = 0;
    // Сбросим признак перемещения
    imgDragging = null;
  }
}
```

Выше были приведены лишь основные функции, которые используются при реализации Конюха. Полностью код Конюха можно скопировать с адреса **ftp://ftp.botik.ru/rented/robot/univer/horse.zip**.

СПРАВОЧНИК

Единицы измерения

Типографские значения

pt — пункты (72 пункта на дюйм)
pc — пики (1 пика равна 12 пунктам)

Экранные значения

px — пиксели — элемент экрана, размер которого зависит от установок монитора и видеокарты.

Абсолютные значения длины

in — дюймы
cm — сантиметры
mm — миллиметры

Шрифт

Первая колонка таблицы содержит название свойства CSS, а также (выделено) название соответствующего свойства объекта `style`.

Пусть, например, в HTML-коде задан такой элемент:

```
<DIV id=block style="font-size:small">
  Огромный шар, надутый паром,<BR>
  Поднялся в воздух он недаром,<BR>
  Наш коротышка хоть не птица,<BR>
  Летать он все-таки годится.<BR>
  И все доступно уж, эхма!<BR>
  Теперь для нашего ума!<BR>
</DIV>
```

Если в процессе интерактивного взаимодействия страницы с пользователем необходимо изменить размер шрифта элемента `block`, то это достигается следующими строками на JavaScript:

```
var block=document.all["block"].style;
block.fontSize = "large";
```

Во второй колонке таблицы приводятся возможные значения свойства. При этом запись в угловых скобках означает целое семейство значений, а запись без угловых скобок задает допустимое ключевое слово, которое можно записывать в качестве значения свойства.

Свойство	Описание
`font-family` **fontFamily**	Значения: название семейства шрифта (например, "New York") или родовое название (`serif`, `sans-serif`, `monospace`, `cursive`, `fantasy`). По умолчанию: устанавливается браузером. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет. Можно установить несколько возможных значений в порядке предпочтения (на случай, когда у пользователя не окажется нужного шрифта). Значения-указания разделяются запятыми: `font-family:"Arial Cyr",Geneva,Helvetica,sans-serif.`
`font-size` **fontSize**	Значения: <*абсолютный*> — `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`; <*относительный*> — `larger`, `smaller`; <*длина*> — в любых допустимых единицах измерения; <*процент*> — значения представлены в процентах от родительского размера шрифта.

По умолчанию: `medium`.
Поддерживается: всеми элементами.
Наследование: да.
Процентная запись: да.

`font-style`
`fontStyle`

Значения:
`normal`, `italic`, `oblique`.
По умолчанию: `normal`.
Поддерживается: всеми элементами.
Наследование: да.
Процентная запись: нет.

`font-variant`
`fontVariant`

Значения:
`normal`, `small-caps`.
По умолчанию: `normal`.
Поддерживается: всеми элементами.
Наследование: да.
Процентная запись: нет.

`font-weight`
`fontWeight`

Значения:
`normal`, `bold`, `bolder`, `lighter` или число от 100 до 900.
По умолчанию: `normal`.
Поддерживается: всеми элементами.
Наследование: да.
Процентная запись: нет.
Если используется числовое значение, то оно должно быть кратным 100.
Значение 400 — это то же, что `normal`, 700 — то же, что и `bold`.

`font`
`font`

Значения:
`<font-style> <font-variant> <font-weigth>`
`<font-size>/<font-heigth> <font-family>`.
По умолчанию: не определено.
Поддерживается: всеми элементами.
Наследование: да.
Процентная запись: только для `<font-size>` и `<font-heigth>`.
Это свойство позволяет установить сразу несколько характеристик шрифта.
Разделитель значений — пробел. Порядок значений важен, но любое значение может быть опущено, кроме `<font-size>` и `<font-family>`:
`font: bold 12pt/14pt sans-serif`.

Цвет и фон

Свойство	Описание

`color`
`color`

Значения:
название цвета или его номер.
По умолчанию: зависит от браузера.
Поддерживается: всеми элементами.
Наследование: да.
Процентная запись: нет.
Цвет может быть задан названием (например, `blue`) или номером по шкале RGB в шестнадцатеричной системе, например, `#ffffff`.

`background`
`background`

Значения:
`transparent` — отсутствие фона;
<цвет> — название цвета или его номер;
<ссылка> — картинка с заданным адресом.
Адрес заключается в скобки и идет непосредственно за ключевым словом `url`:

BODY {background: url(fon1.gif)}. Можно использовать и цвет, и картинку. Картинка будет лежать поверх цветного фона;
<*повтор*> — repeat (по умолчанию), repeat-x, repeat-y, no-repeat;
<*скролл*> — fixed, scroll.
Определяет, будет ли фоновая картинка оставаться на месте или прокручиваться вместе со страницей;
<*позиция*> — определяет расположение картинки на странице.
Значения могут быть процентными, абсолютными или ключевыми словами: top, middle, bottom, left, center, right;
По умолчанию: transparent.
Поддерживается: всеми элементами.
Наследование: нет.
Процентная запись: да, относительно размера самого элемента.
Порядок следования указаний не важен, разделитель — пробел:
background: #C0C0C0 right no-repeat url(../pic/explorer.gif).

background-attachment
backgroundAttachment

Значения:
fixed, scroll.
По умолчанию: scroll.
Поддерживается: всеми элементами.
Наследование: нет.
Процентная запись: нет.
Определяет, будет ли фоновая картинка занимать фиксированное положение или прокручиваться вместе с содержимым.

background-color
backgroundColor

Значения:
transparent, <*цвет*>.
По умолчанию: transparent.
Поддерживается: всеми элементами.
Наследование: нет.
Процентная запись: нет.
Определяет цвет фона элемента. Цвет может быть задан названием (например, blue) или номером по шкале RGB в шестнадцатеричной системе, например, #ffffff.

background-image
backgroundImage

Значения:
none, <*ссылка*>.
По умолчанию: none.
Поддерживается: всеми элементами.
Наследование: нет.
Процентная запись: нет.
Определяет фоновую картинку.
Синтаксис записи ссылки: BODY {background-image: url(fon1.gif)}.

background-position
backgroundPosition

Значения:
<*позиция*>, top, center, bottom, left, right.
По умолчанию: top, left.
Поддерживается: всеми элементами.
Наследование: нет.
Процентная запись: нет.
Определяет начальное положение фоновой картинки (или фоновой заливки) с помощью двух значений (сначала по горизонтали, потом по вертикали) или при помощи двух ключевых слов.

background-repeat
backgroundRepeat

Значения:
repeat, repeat-x, repeat-y, no-repeat.
По умолчанию: repeat.
Поддерживается: всеми элементами.
Наследование: нет.
Процентная запись: нет.
Определяет, повторяется ли фоновая картинка.
Если используются repeat-x или repeat-y, то картинка повторяется лишь вдоль одного направления.

Текст

Свойство	Описание
`letter-spacing` **`letterSpacing`**	Значения: `normal`, *<расстояние>*. По умолчанию: `normal`. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет. Определяет добавочное расстояние между буквами.
`line-height` **`lineHeight`**	Значения: `normal`, *<число>*, *<расстояние>*, *<процент>*. По умолчанию: зависит от браузера. Поддерживается: всеми элементами. Наследование: да. Процентная запись: да, по отношению к размеру шрифта элемента. Определяет высоту текущей строки. Значение *<число>* интерпретируется как размер шрифта текущего элемента, умноженный на соответствующее число (например, 1,2). Если используется *<расстояние>*, должны применяться единицы измерения. Процентное отношение используется в сравнении с текущим размером шрифта и должно быть больше 100%.
`text-align` **`textAlign`**	Значения: `left`, `right`, `center`, `justify`. По умолчанию: зависит от браузера. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет. Определяет, как текст будет выровнен относительно содержащего его элемента.
`text-decoration` **`textDecoration`**	Значения: `none`, `overline`, `underline`, `line-through`. По умолчанию: `none`. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Линия вдоль строки (над, под, через).
`text-indent` **`textIndent`**	Значения: *<величина>*, *<процент>*. По умолчанию: 0. Поддерживается: всеми элементами. Наследование: да. Процентная запись: да, относительно ширины родительского элемента. Устанавливает величину отступа первой строки элемента (красной строки) в единицах измерения или как процент от ширины родительского элемента.
`text-transform` **`textTransform`**	Значения: `capitalize` — делает заглавной первую букву каждого слова; `uppercase` — делает все буквы в словах заглавными; `lowercase` — делает все буквы в словах строчными. По умолчанию: `none`. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет.

vertical-align **verticalAlign**	Значения: baseline — выравнивание по образцу родительского элемента; sub — выравнивает элемент под базовой линией; super — выравнивает элемент над базовой линией; top — выравнивает верх элемента по верху самого высокого элемента текущей строки; text-top — выравнивает элемент по верху текста, набранного шрифтом родительского элемента; middle — устанавливает вертикальную среднюю линию элемента на основе родительского элемента плюс половина строки последнего; bottom — выравнивает по низу самого низкого элемента текущей строки; text-bottom — выравнивает элемент по низу текста, набранного шрифтом родительского элемента. По умолчанию: baseline. Поддерживается: встроенными элементами. Наследование: нет. Процентная запись: да, по отношению к высоте строки.
list-style-image **listStyleImage**	Значения: none, *<ссылка>*. По умолчанию: none. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет. Определяет адрес картинки, используемой в качестве маркера для каждого элемента списка.
list-style-type **listStyleType**	Значения: none, circle, disc, square, decimal, lower-alpha, upper-alpha, lower-roman, upper-roman. По умолчанию: disc. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет. Определяет вид маркера списка.
list-style-position **listStylePosition**	Значения: outside, inside. По умолчанию: outside. Поддерживается: всеми элементами. Наследование: да. Процентная запись: нет. Указывает, внутри или вне "тела" списка должен находиться маркер.

Поля и рамки

Свойство	Описание
border-color border-top-color border-bottom-color border-right-color border-left-color **borderColor** **borderTopColor** **borderBottomColor** **borderRightColor** **borderLeftColor**	Значения: *<цвет>*. По умолчанию: *<нет цвета>*. Поддерживается: блоками элементов и замещающими элементами. Наследование: нет. Процентная запись: нет. Определяет цвет четырех сторон рамки. Свойство работает, если задано свойство border-style.

`border-style`
`border-top-style`
`border-bottom-style`
`border-right-style`
`border-left-style`
`borderStyle`
`borderTopStyle`
`borderBottomStyle`
`borderRightStyle`
`borderLeftStyle`
	Значения:
	`none, solid, double, groove, ridge, inset, outset.`
	По умолчанию: `none`.
	Определяет стиль четырех сторон рамки.
	Поддерживается: блоками элементов и замещающими элементами.

`border-width`
`border-top-width`
`border-bottom-width`
`border-right-width`
`border-left-width`
`borderWidth`
`borderTopWidth`
`borderBottomWidth`
`borderRightWidth`
`borderLeftWidth`
	Значения:
	`thin, medium, thick, <`*размер*`>`.
	По умолчанию: `medium`.
	Поддерживается: блоками элементов и замещающими элементами.
	Наследование: нет.
	Процентная запись: нет.
	Определяет ширину четырех сторон рамки.

`border`
`border-top`
`border-bottom`
`border-right`
`border-left`
`border`
`borderTop`
`borderBottom`
`borderRight`
`borderLeft`
	Значения:
	`<`*ширина*`> <`*стиль*`> <`*цвет*`>`.
	По умолчанию: `medium none <`*нет цвета*`>`.
	Поддерживается: блоками элементов и замещающими элементами.
	Наследование: нет.
	Процентная запись: нет.
	Определяет свойства четырех сторон рамки.
	Объединяет все другие свойства рамки.

`margin`
`margin-top`
`margin-bottom`
`margin-right`
`margin-left`
`margin`
`marginTop`
`marginBottom`
`marginRight`
`marginLeft`
	Значения:
	`auto, <`*ширина*`>, <`*процент*`>`.
	По умолчанию: 0.
	Поддерживается: блоками элементов и замещающими элементами.
	Наследование: нет.
	Процентная запись: да, относительно ширины родительского элемента.
	Определяет размеры отступов данного элемента.

`padding`
`padding-top`
`padding-bottom`
`padding-right`
`padding-left`
`padding`
`paddingTop`
`paddingBottom`
`paddingRight`
`paddingLeft`
	Значения:
	`<`*ширина*`>, <`*процент*`>`.
	По умолчанию: 0.
	Поддерживается: блоками элементов и замещающими элементами.
	Наследование: нет.
	Процентная запись: да, относительно ширины родительского элемента.
	Определяет расстояние между содержимым и рамкой элемента.

Вид

Свойство	Описание
display **display**	Значения: "", none. По умолчанию: "". Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Это свойство определяет, будет ли показан элемент. Если элемент не виден (none), то место на странице под него не отводится.
float **styleFloat**	Значения: none, left, right. По умолчанию: none. Поддерживается: **DIV**, **SPAN** и замещаемыми элементами. Наследование: нет. Процентная запись: нет. Определяет обтекание элемента другими элементами (под, слева или справа).
clear **clear**	Значения: none, both, left, right. По умолчанию: none. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Указывает, что следующие элементы должны быть показаны ниже элемента, выровненного по левому или правому краю. По умолчанию текст "обтекает" такие элементы.
overflow **overflow**	Значения: "", scroll. По умолчанию: "". Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Значение scroll указывает, что если содержимое элемента выйдет за его границы, то оно будет прокручиваться.
visibility **visibility**	Значения: "", visible, hidden, inherit. По умолчанию: inherit. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Позволяет элементу быть видимым или невидимым на странице. Невидимые (hidden) элементы занимают то же место и так же влияют на расположение других элементов, как и видимые (visible), но становятся прозрачными. Значение inherit означает, что элемент виден, если его родительский элемент является видимым.
width **width** **pixelWidth** **posWidth**	Значения: auto, <*ширина*>, <*процент*>. По умолчанию: auto. Поддерживается: **DIV**, **SPAN** и замещаемыми элементами. Наследование: нет. Процентная запись: да, относительно ширины родительского элемента. Устанавливает ширину элемента. Значение возвращается как строка, включающая тип единицы измерения (px, % и т.д.). Для получения значения в виде числа используется свойство posWidth объекта style.

height **height** **pixelHeight** **posHeight**	Значения: auto, <*высота*>. По умолчанию: auto. Поддерживается: **DIV**, **SPAN** и замещаемыми элементами. Наследование: нет. Процентная запись: нет. Устанавливает высоту элемента. Значение возвращается как строка, включающая тип единицы измерения (px, cm и т.д.). Для получения значения в виде числа используется свойство posHeight объекта style.

Позиционирование, z-index

Свойство	Описание
positon **positon**	Значения: absolute, relative, static. По умолчанию: static. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Свойство position объекта style доступно только для чтения.
left **left** **pixelLeft** **posLeft**	Значения: auto, <*координата*>, <*процент*>. По умолчанию: auto. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: да, относительно ширины родительского элемента. Устанавливает горизонтальную координату элемента. Значение возвращается как строка, включающая тип единицы измерения (px, % и т.д.). Для получения значения в виде числа используется свойство posLeft объекта style.
top **top** **pixelTop** **posTop**	Значения: auto, <*координата*>, <*процент*>. По умолчанию: auto. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: да, относительно ширины родительского элемента. Устанавливает вертикальную координату элемента. Значение возвращается как строка, включающая тип единицы измерения (px, % и т.д.). Для получения значения в виде числа используется свойство posTop объекта style.
z-index **zIndex**	Значения: <*число*>. По умолчанию: в зависимости от контекста в исходном тексте HTML. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Указывает, в каком порядке элементы будут перекрывать друг друга. Элементы с более высоким z-индексом будут располагаться сверху элементов с более низким. Положительные значения ставят элемент перед обычным текстом, отрицательные — ниже его.

Курсор

Свойство	Описание
cursor **cursor**	Значения: auto, crosshair, default, hand, move, e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, w-resize, text, wait, help. По умолчанию: auto. Поддерживается: всеми элементами. Наследование: нет. Процентная запись: нет. Определяет вид курсора мыши.

Один из августовских номеров "Информатики" (№ 30) будет посвящен памяти Геннадия Анатольевича Звенигородского. Он был одним из тех, кто начинал развивать школьную информатику в нашей стране. Анонсируя этот номер, мы предоставляем слово коллеге Г.А. Звенигородского по работе в новосибирском Академгородке, автору множества замечательных материалов, опубликованных в "Информатике", **Юрию Абрамовичу ПЕРВИНУ**.

От Робика до Роботландии

Система "Школьница", созданная Г.А. Звенигородским и его учениками 20 лет тому назад в новосибирском Академгородке, и программно-методическая система "Роботландия", рожденная в Переславле-Залесском, не схожи ни по замыслу, ни по реализации. Это неудивительно: ведь никто из "роботландцев", включая тех, кто перебрался из Новосибирска в Переславль, никогда не оперировал в теоретических аргументациях введенными Г.А. Звенигородским терминами "экстравертивных" и "интровертивных" языковых средств, предназначенных для учебных целей; никто в "Роботландии" не программировал на лучшем детище практической языкотворческой деятельности Звенигородского – Рапире. И тем не менее сейчас, когда Г.А. Звенигородский отмечал бы свой 50-летний юбилей, волей-неволей возникают ассоциации, и хочется в том следе, который оставила на небосклоне отечественной школьной информатики яркая метеорная полоса творчества Г.А. Звенигородского, проследить начала пути "от Робика до Роботландии".

1. Концепция исполнителей

Главная встреча в жизни Г.А. Звенигородского — это, бесспорно, знакомство с академиком А.П. Ершовым, знакомство, переросшее в активное плодотворное творчество и сотрудничество. А.П. Ершов высоко ценил Звенигородского. Менее чем через год после смерти Г.А. Звенигородского Андрей Петрович выпустил (собрал, отредактировал и написал заглавную статью) книгу-сборник, которую он назвал "Проблемы школьной информатики", отдавая дань памяти не только младшему коллеге и другу, но и человеку, непосредственно участвовавшему вместе с Ершовым в создании самого имени новой прикладной науки — "школьная информатика" ([1]). В своей статье этого сборника А.П. Ершов отмечает, что значительный "пласт работ Звенигородского составляют учебно-методические материалы, которые, по моему мнению, весьма интересны для исследователя как образец конкретного педагогического материала, не обремененного излишними рассуждениями, но содержащего немало конкретных находок, из которых вырастали общие концепции и установки".

К числу важнейших педагогических открытий Звенигородского принадлежит, несомненно, понятие исполнителя как фундаментального методического инструмента в первичном освоении основ информатики. В созданной им школе программирования, где он учил младших школьников, он уверенно, убежденно ввел понятие исполнителя и детально разработал методику уроков ([2]). Могу с полной ответственностью назвать это педагогическим открытием: хотя Звенигородский начал рассказывать детям об исполнителях позднее великого Сеймура Пейперта, но совершенно независимо от него: я познакомил Звенигородского с работами С.Пейперта только в 1974 году ([10]). Он сразу высоко оценил значимость пейпертовских идей, а ведь к этому времени у Звенигородского была уже целая коллекция разнообразных, очень простых исполнителей (что с дидактических позиций — многообразия учебных сред как одного из принципов дидактики — опережало идею единственной (хотя и "многоликой") Черепашки. Правда, в отличие от реально работающей программной модели Черепашки Пейперта (а в некоторых реализациях Лого — даже аппаратных реализаций такого робота-исполнителя), роботы Звенигородского оставались умозрительными. Тем более значим педагогический, методический результат Звенигородского, который по-настоящему сумел увлечь малышей не только рассказами о многочисленных придуманных им роботах, не только предложить ученикам обширную коллекцию содержательных задач-упражнений, но и добиться усвоения важнейших, фундаментальных понятий, с которыми можно было смело входить в информатику, — предписание (команда), система предписаний, условие, список допустимых условий, исполнитель, программа, основные законы программирования, язык управления роботом. Исполнителей Звенигородского не перечислишь — это и робот-повар, умело оперирующий с кастрюлями, и робот-артиллерист, стреляющий из старинной пушки, и робот-водолаз, перезаряжающий подводные фотокамеры на автоматических глубоководных станциях, и робот-механик, заменяющий детали на шагающем экскаваторе, и робот-океанограф, отлавливающий редких рыб для научной коллекции... Когда не хватало имен и обозначений для все новых и новых роботов, в ход шли имена и прозвища — Робот Миши Киберовского, Робот Светы Ошибочкиной и, конечно, Робот Гены Крокодилова (зеленый Гена-крокодил стал своеобразной эмблемой Районной школы юных программистов Академгородка).

Заметьте два существенных обстоятельства. Первое из них: вся эта кипучая практическо-педагогическая и теоретическо-методическая работа началась и настойчиво велась задолго до появления персональных компьютеров. Зато, когда они оказались доступными (А.П. Ершов смог добиться, чтобы первые зарубежные персоналки — Ямахи, а затем и первые отечественные персональные компьютеры Агаты пришли прежде всего в новосибирский Академгородок, в группу школьной информатики академического Вычислительного центра), Г.А. Звенигородский со свойственной ему творческой страстью набросился на программные разработки учебных роботов и заразил этой страстью своих юных учеников-сибиряков. Часть исполнителей перекочевали на экраны Агатов со страниц методических брошюрок Звенигородского, некоторые создавались по ходу проектирования и разработки учебного языка управления исполнителями — Робик ([3]). Каждый из таких программно реализованных роботов-исполнителей был ориентирован на решение конкретных, частных педагогических задач. Так, робот с простым названием "Том Сойер" занимался только тем, что формировал у детей понятие цикла: именно в цикле — доска за доской — Том Сойер красил забор (и надо сказать, что на уроках Звенигородского дети работали с этим роботом не менее заинтересованно, чем друзья прототипа Тома Сойера красили забор у Марка Твена). Главной задачей известного робота Дежурика было формирование навыков строгого следования допустимых серий команд...

Звенигородский был первым, кто у нас в Советском Союзе ввел исполнителей в школьную информатику. Только затем этим путем прошли все известные в нашей стране группы разработчиков учебного программного обеспечения — группа А.Г. Кушниренко в Москве (Робот, Чертежник...), группа екатеринбуржцев (Ру и Робби, Кенгуренок) и, конечно, целиком построенная на исполнителях Роботландия из Переславля-Залесского.

Прежде чем рассказывать о судьбах идей Звенигородского в Роботландии, уместно вспомнить первую встречу А.А. Дуванова и Ю.А. Первина, которая состоялась на "нейтральной" территории — в Свердловске в дни проходившей там конференции по школьной информатике. Тогда уже вышел на просторы школьных компьютерных экранов робот-исполнитель Муравей. Место этого робота в раннем обучении информатике дальневосточная группа А.А. Дуванова отметила сразу. Но она же нашла и "слабое место" Муравья. При всей "старательности" этого неутомимого исполнителя он не умел понимать и выполнять условные команды. Чтобы исправить это положение, А.А. Дуванов задумал усовершенствовать Муравья. Для этого, ясно, исполнителю были нужны более длинные усы (как иначе можно "ощупать" перевернутый кубик на клетчатом поле!). Так родился исполнитель, которого не было у Звенигородского и который (из-за длинных своих усов) был назван Тараканом. Трудная жизнь сложилась у Таракана: на "методическом" пути этого (по существу, встреченного на ура) исполнителя оказалось много противников (особенно в сфере руководящих административных работников образования), по разным соображениям не принявших некрасивые ассоциации имени этого робота. Выручила поездка разработчиков на Кубу (1987 г., когда А.А. Дуванов уже был ведущим роботландцем): с кубинской выставки советских учебных программных средств Таракан вернулся с испанским именем Кукарача (это простой перевод слова "таракан" на испанский язык) и сразу обрел многочисленных друзей и поклонников в нашей стране как среди детей, так и среди школьных учителей.

Второе обстоятельство, о котором следует упомянуть в рассказе о пионерских педагогических начинаниях Г.А. Звенигородского: разработку своих роботов-исполнителей Звенигородский начинал и вел в ту пору, когда в стране вовсю бушевали дискуссии языкотворцев. (Сам Г.А. Звенигородский языкотворчества не только не чуждался, но и сам был активнейшим языкотворцем, хорошо понимавшим сложность, актуальность и ответственность этого вида деятельности: и Робик, и Рапира — это языки, созданные и обоснованные им.) Спорили о том, каким должен быть язык для обучения школьников программированию. Приводились доводы в пользу Бейсика (мол, самый простой и самый "дешевый" язык с трансляторами, встраивавшимися во все выпускавшиеся тогда персональные ЭВМ); говорили о "гибкости" Алгола, который близок по своим управляющим структурам к свойствам человеческого мышления и в силу этого качества заслуживавший, как казалось, места в школьном курсе; обсуждались учебные перспективы Кобола, который в то время считался самым распространенным языком в мире, активно выступали сторонники Смолл Тока, Паскаля и т.п. Забывалось при этом, что многоязычие в мире программирования объективно именно потому, что для каждой предметной области создается адекватный ей язык. Но ведь предметная область, которую можно было бы определить как "методика раннего обучения информатике" (или, если отдавать дань исторической правде — обучения программированию), никак не похожа ни на экономические расчеты, где имеет право "царствовать" Кобол, ни на расчеты ядерной физики, где традиционно и обоснованно господствует Фортран, ни на задачи о списках, где удобен Лисп. Одна из научных заслуг Г.А. Звенигородского — это выработка взвешенных требований к учебным и учебно-производственным языкам и их место в классификации высокоуровневых языковых систем программирования ([4]).

Имели место подобные языкотворческие дискуссии и в Роботландии. Главным претендентом на роль основного учебного языка казался Лого (здесь взвешивались и высокие объективные оценки языка, и субъективное личное обаяние С.Пейперта, неоднократно бывавшего в Переславле и даже присутствовавшего на уроках четвероклассников по Лого). И все же упоминавшийся выше дидактический принцип многообразия учебных сред показался разработчикам настолько весомым, что даже Лого (будучи хотя и весьма разви-

той, но единственной системой в учебном процессе) в глазах авторов проектируемой системы вынужден был отступить. В качестве главного инструмента проектируемого комплекса была выбрана система программных исполнителей, каждый из которых замышлялся для формирования того или иного из множества умений и навыков, в совокупности составлявших сформулированный еще А.П. Ершовым ([8]) операционный стиль мышления. Влияние идей Г.А. Звенигородского здесь ощущалось несомненно. Роботландия реализована как методически обоснованная система программных исполнителей так, что место каждого из них объективно определено требованиями методики раннего обучения информатике.

Разумеется, речь шла лишь об идейной близости, а не о близости реализационных установок. Действительно, Роботландия создавалась позднее, чем система "Школьница": в это время наряду с алгоритмическим направлением в школьной информатике существенное влияние приобрело и технологическое направление. Роботландия — это система раннего обучения *информатике*, тогда как "Школьница" — это инструмент обучения *программированию*. В силу этого наряду с широким набором исполнителей Роботландия обрела многочисленные технологические инструменты-модули — текстовые, графические, музыкальные редакторы, адаптированные базы данных. Последняя (разрабатываемая сейчас) версия Роботландии настолько пронизана влиянием коммуникационных технологий, что это нашло отражение даже в ее названии — Роботландия.RU.

2. Методически обусловленная цепочка расширяющихся программных средств

Еще одна идея, связывающая (правда, несколько опосредованно) систему "Школьница" и последовавшую через десятилетие после нее программно-методическую систему Роботландия, — идея, которую можно назвать методически обусловленной совокупностью языковых высокоуровневых учебных систем программирования. Эта идея тоже была рождена в группе школьной информатики Вычислительного центра СО АН СССР и была доведена Г.А. Звенигородским и его учениками до уровня реализации, прошедшей как практическую проверку в учебном процессе многих школ, так и официальную государственную приемку.

Из отмеченной выше объективности многоязычия в мире программирования можно было бы сделать "конструктивный" вывод: давайте создадим язык, полностью удовлетворяющий требованиям предметной области (в нашем случае — методике раннего обучения информатике), и социально-педагогическая задача обучения поколения молодежи будет решена. Увы, в действительности положение сложнее. И дело даже не в лежащем на поверхности контрдоводе: если в школьном курсе изучаться будет некий "учебный" язык, то в производственной практике, в жизни, знания выпускника школы не будут востребованы. Дело в удивительном парадоксе — программист, хотя и является представителем профессии, расположенной на передовом фронте научно-технического прогресса, оказывается консерватором в силу "универсальности" любого из языков программирования: будучи уверенным, что любую производственную задачу программист может запрограммировать и выполнить на машине, владея одним языком программирования, он (программист) не стимулирован к изучению (вообще говоря, весьма трудоемкому) другого языка.

Здесь и рождается идея методической цепочки (системы) двух (или, быть может, нескольких) учебных языков. В этом случае учащийся изучает сначала некоторый очень простой язык (типа языков управления исполнителями). Затем методические трудности, возникающие на некотором этапе учебного процесса, приводят к необходимости освоения иных программных средств, иного языка, в которых встретившиеся трудности могут быть обойдены более естественным путем. Ясно при этом, что те самые "трудности", о которых идет речь (и так это на самом деле и происходит), программируются разработчиками системы априорно с целью стимулировать переход к освоению нового языка. Именно поэтому язык начального обучения Робик и наследующая его Рапира — учебно-производственный язык — увязываются в единую методическую цепочку.

В "Школьнице" переход "Робик — Рапира" задуман следующим образом. Ученики достаточно комфортно чувствуют себя, пока им приходится управлять простыми исполнителями. Даже управляющие структуры — циклы и ветвления — воспринимаются и осваиваются сравнительно легко. Но вот наступает этап освоения естественного, казалось бы, механизма любого языка программирования — вычисление арифметических выражений и присваивание значений именованным переменным. И тут ученик "спотыкается" о громоздкость конструкции изучавшегося до этой поры языка. Конечно, и в Робике можно запрограммировать вычисление арифметического выражения, записывая его, например, так:

СЛОЖИТЬ ЗНАЧЕНИЕ А И ЗНАЧЕНИЕ Б
И ПРИСВОИТЬ ИМЕНИ В (1)

Написание такой команды (здесь она записана грамматически верно, по правилам Робика) становится утомительным для школьника очень скоро, буквально после второй подобной строки. Поначалу учитель пытается "утешить" школьника тем, что ряд ключевых слов в этой конструкции факультативен, и, следовательно, освоив семантику использованных здесь ключевых слов, ученик может свести запись (1) к более компактной, например, такой:

СЛОЖИТЬ А И Б ПРИСВОИТЬ В

Но необходимость записывать даже такие фразы для мало-мальски длинных выражений приводит учащегося в уныние. И тогда учитель делает "революционный" шаг: он показывает, что можно сделать это на другом языке гораздо проще — в Рапире строка (1) записывается намного более выразительно:

$$А + Б \rightarrow В \qquad (2)$$

При достигнутом к этому моменту понимании сущности механизма присваивания команда (2) не просто понятна, а увлекает детей своей простотой и естественностью. Вот он — стимул перехода к другому языку!

Дальше происходят удивительные вещи. Ученики заинтересованно изучают (в течение учебного года) развитый учебно-производственный язык Рапира. А в конце года этим ученикам предстоит производственная практика в одном из вычислительных центров разнообразных научных институтов (напоминаю: речь идет об Академгородке). И тогда выясняется, что в зависимости от места производственной практики школьнику предстоит освоить новый, ранее незнакомый "производственный" язык. Но ребята, уже понявшие в ходе уроков, что переход от языка к языку — это нормальное явление, изучают новое средство (подчас самостоятельно) за один или два вечера.

Многие скептики пытались объяснять очевидные успехи Г.А. Звенигородского не значимостью полученных им педагогических результатов, а исключительно его индивидуальными педагогическими способностями: мол, дальше реализации курса с участием самого Звенигородского система нежизнеспособна. Поэтому столь важно, что методическая цепочка последовательно изучаемых языков со стимулированием перехода от одного к другому оказалась не только практически эффективной в конкретной практике новосибирской школы юных программистов, но и показала общность дидактических категорий в новой школьной дисциплине: так же, как многие другие, годами изучаемые в школе предметы (история, естествознание, ...) "работают" по принципу дидактической спирали, поднимаясь из года в год, от одного слоя содержания образования к другому, более насыщенному, так и методика обучения программированию в непрерывном школьном образовании, оказывается, должна переходить от простых механизмов, понятий, идей к более сложным вместе с ростом потенциала общекультурных знаний учащегося и изменением его психолого-педагогических качеств.

В программно-методической системе раннего обучения информатике — Роботландии — не возникала проблема многоязычия. В ней вообще программирование сдвигалось в самый конец курса и ограничивалось пропедевтическим языком управления Кукарача (превосходящим Робик, но уступающим по своим возможностям Рапире). Тем не менее принцип методической последовательности постепенно усложняющихся дидактических инструментов оказывается эффективным не только в цельном курсе (как у Звенигородского — в курсе программирования), но и в любой достаточно самостоятельной сложной теме курса. Именно таким образом построена в Роботландии, например, важная и в практическом, и в гносеологическом плане тема текстовой обработки информации. От простейшего строкового редактора (Правилка), в котором дети осваивают навыки клавиатурного набора и умения эффективной корректировки ошибок, учащиеся переходят к ограниченным по размерам рабочего поля прикладным и игровым задачам, закрепляющим простейшие навыки текстового редактирования (Привет, Блокнот) и только после этого начинают осваивать простой адаптированный учебный текстовый редактор Микрон.

Некоторые "экспериментаторы" от информатики хвастаются тем, что их дети во втором и третьем классе ловко оперируют Word'ом. Согласен, добиться этого можно. Но нужно ли? Такие эксперименты не вызывают доверия, поскольку ребенок этого возраста не подготовлен к восприятию того обилия инструментальных механизмов (бесчисленные стрелки, кнопки, пиктограммы, специальные приемы, ...), которыми Word способен ошеломить даже взрослого пользователя.

Осваивая Микрон (в котором впервые школьники знакомятся с такими фундаментальными понятиями, как принципы хранения информации, файл, его имя, каталог, операции информационного обмена), дети приходят к типовой задаче контекстного поиска. Отсюда рукой подать до не менее актуальной задачи текстового редактирования — контекстной замены. Конечно, она реализуема и в Микроне. Но каких это стоит усилий! Это не недосмотр разработчиков, а специально задуманный стимулятор перехода от учебного текстового редактора (Микрон) к редактору учебно-производственному, но пока еще моноширинному (МикроМир). С помощью этого редактора оказывается легко решаемой не только задача контекстной замены, но и многие другие интересные и дидактически важные задачи (групповые операции над текстами, механизмы обработки таблиц, встроенные вычислительные операции, многооконная обработка текстовой информации, макросы, настройки и проч.). Казалось бы, учебно-производственный редактор способен покрыть мыслимые потребности учащихся. Однако теперь, когда они знакомы с фундаментом этой прикладной темы, учащиеся готовы к освоению новых, профессиональных проблем и инструментов — совмещение текста и графики в Word'е, создание и использование гипертекстов и средств разметки текстов (HTML). Именно реализованный Звенигородским впервые в практике проектирования учебного программного обеспечения принцип методически обусловленной цепочки расширяющихся программных средств, перенесенный в Роботландию, позволил системно решить проблему инструментария в теме текстового редактирования.

Примеры применения этого принципа не ограничиваются, конечно, только текстовым редактированием.

3. Практическая педагогическая деятельность со школьниками

Г.А. Звенигородскому довелось испытать себя в разных формах педагогического труда — и общеобразовательная школа, и внеклассная работа в Районной школе юных программистов ([2]), и заочная школа программирования на страницах общесоюзного юношеского журнала "Квант" ([9]), и предпрофессиональное обучение программированию в межшкольном комбинате ([7]). Каждая из этих педагогических реализаций Звени-

городского заслуживает отдельного обсуждения и исследования. В смысле поднятой здесь темы — продолжения сибирских традиций в деятельности Роботландии — представляются интересными и важными проводившиеся в течение десяти лет всесоюзные летние школы под Новосибирском. Эти традиции зародились с 1976 года, с первого приезда в Академгородок группы харьковских юных программистов, возглавляемых их молодым учителем Г.А. Звенигородским. Уже со следующего года к новосибирцам и харьковчанам присоединилась группа школьников из Барнаула. После этого ежегодные встречи переросли во всесоюзные летние школы юных программистов ([5], [6]). Об их научно-методической значимости и педагогическом опыте было написано немало. Остались эти школы и в памяти участников. Не менее важно, однако, и то, что эта форма внешкольной работы, задуманная и реализованная с непосредственным участием Г.А. Звенигородского, оказалась востребованной. Переславская Роботландия переняла эстафету летних школ, организовав в Международном детском компьютерном лагере на берегу Плещеева озера ежегодные июньские встречи молодых талантливых и увлеченных детей. Хотя переславские летние школы не являются параллельным переносом сибирского опыта во времени и в пространстве, роботландцы никогда не отрицали наследие этого опыта. Позднее, когда несколько изменились условия работы в Переславле, методология летних школ была продолжена энтузиастами школьной информатики в Красноярске (Л.Б. Чубаров), Самаре (А.Н. Никитин) и даже за рубежом — в Словакии и Болгарии (Ю.М. Горвиц).

Роботландия осознанно пошла на проведение летних школ, несмотря на огромную дополнительную работу, прежде всего потому, что, подобно новосибирским школам, лагерь на берегу Плещеева озера стал научным полигоном для проверки педагогических, программных и технических гипотез информатизации учебного процесса. И если успех и продуктивность этого увлекательного вида педагогической деятельности ныне общепризнан, то важно помнить, что истоки этого успеха восходят к тем методическим находкам, которые были найдены и развиты вместе с Г.А. Звенигородским на берегах Обского моря.

Литература

1. *Ершов А.П.* О работах Г.А. Звенигородского по школьной информатике. Заглавная редакторская статья в сборнике "Проблемы школьной информатики". Новосибирск: ВЦ СО АН СССР, 1986.

2. *Звенигородский Г.А.* Первые шаги в программировании. "Наука" (Библиотечка "Квант"). М., 1985.

3. *Звенигородский Г.А., Налимов Е.В.* Язык Робик и его реализация в системе "Школьница". В сб. "Проблемы школьной информатики". Новосибирск: ВЦ СО АН СССР, 1986.

4. *Звенигородский Г.А.* Сравнительный анализ языков программирования, используемых в школьном учебном процессе. В сб. "Проблемы школьной информатики". Новосибирск: ВЦ СО АН СССР, 1986.

5. *Виленкина Л.Н., Звенигородский Г.А.* Четвертая летняя школа юных программистов. Квант № 12, 1979.

6. *Звенигородский Г.А., Первин Ю.А., Сосинский А.Б.* Пятая Всесоюзная летняя школа юных програмистов. Квант № 12, 1980.

7. *Звенигородский Г.А., Первин Ю.А.* Профориентационное обучение программированию в условиях школьного кабинета информатики (из опыта работы). Препринт ВЦ СО АН СССР № 6456, выпуск 10. Новосибирск, 1986.

8. *Ершов А.П., Звенигородский Г.А., Первин Ю.А.* Школьная информатика (концепция, состояние, перспективы). Препринт ВЦ СО АН СССР № 152. Новосибирск, 1979. (Ретроспективная публикация: *Первин Ю.А.* Школьная информатика. ИНФО № 1, 2000.)

9. *Юнерман Н.А.* Итоги трехлетнего цикла Всесоюзной заочной школы программирования. Материалы Объединенного семинара ВЦ СО АН СССР, НГУ и Новосибирского отделения Всероссийского педагогического общества "ЭВМ и учебный процесс", ВЦ СО АН СССР. Новосибирск, 1982.

10. *Первин Ю.А.* Три вехи в череде воспоминаний. Информатика № 30/2002.

Номер лота	Название	Цена
10-01-0	С.Л. Соловейчик. Пушкинские проповеди	60 руб.
10-02-0	С.Л. Соловейчик. Последняя книга	75 руб.
10-05-0	С.Л. Соловейчик. Педагогика для всех	90 руб.
10-06-0	М.А. Балабан. Школа-парк. Как построить школу без классов и уроков	65 руб.
▼10-03-0	Школа сотрудничества (Сборник статей по педагогике сотрудничества)	46 руб.
27-01-9	Я иду на урок физики. 7 класс (в 3-х книгах)	138 руб.
27-02-0	Я иду на урок астрономии. Звездное небо. 11 класс	46 руб.
29-01-0	Я иду на урок химии. 8–11 классы	46 руб.
29-02-0	Я иду на урок химии. Летопись важнейших открытий. XVII–XIX веков	46 руб.
21-01-0	Я иду на урок математики. 5 класс	46 руб.
21-01-7	Я иду на урок математики. Тесты. 5 класс	25 руб.
21-02-0	Я иду на урок математики. 6 класс	46 руб.
21-03-0	Я иду на урок математики. Алгебра. 7 класс	46 руб.
17-01-0	Я иду на урок информатики. Задачи по программированию. 7–11 классы	46 руб.
24-01-0	Я иду на урок русского языка. Диктанты. 5–9 классы	46 руб.
24-02-0	Шапиро Н.А. Русский язык в упражнениях. 5–7 классы	37 руб.
12-01-0	Я иду на урок биологии. Зоология. Беспозвоночные	46 руб.
12-02-0	Я иду на урок биологии. Зоология. Рыбы и земноводные	46 руб.
12-03-0	Я иду на урок биологии. Зоология. Пресмыкающиеся	46 руб.
12-04-0	Я иду на урок биологии. Человек и его здоровье	46 руб.
19-02-0	Я иду на урок истории. Древнейшая и древняя история	46 руб.
19-04-0	Я иду на урок истории. Материалы по истории средних веков	46 руб.
▼19-03-0	Л.А. Кацва. Россия в 1917–1918 годах. 10–11 классы	46 руб.
19-05-1	Занимательная история России. С древнейших времен до середины XVI века	46 руб.
19-05-2	Занимательная история России. Середина XVI века — конец XVII века	46 руб.
19-05-3	Занимательная история России. 1700–1762 годы	46 руб.
14-01-0	Я иду на урок географии. Физическая география материков и океанов	46 руб.
14-02-0	Я иду на урок географии. История географических открытий	46 руб.
14-03-0	Я иду на урок географии. Физическая география России	46 руб.
20-02-0	Я иду на урок литературы. 5 класс	46 руб.
20-03-0	Я иду на урок литературы. 9 класс	46 руб.
20-01-0	Я иду на урок литературы. 10 класс	46 руб.
20-04-0	Я иду на урок литературы. Готовимся к экзаменационному сочинению. 11 класс	46 руб.
20-05-0	Я иду на урок литературы. Современная русская литература. 1970—1990-е годы	46 руб.

▼ Тираж ограничен.

Указанные цены действительны до 1 октября 2002 года

ЧТОБЫ ПРИОБРЕСТИ ЗАИНТЕРЕСОВАВШИЕ ВАС КНИГИ НАЛОЖЕННЫМ ПЛАТЕЖОМ:

1. Укажите в купоне ваши фамилию, имя, отчество, индекс и почтовый адрес.
2. Выберите нужные вам книги из списка (не более 5 названий в один купон) и укажите соответствующий номер лота и количество экземпляров.
3. Вырежьте и вышлите купон в конверте с пометкой «Книга — почтой» по адресу: 150000, г. Ярославль, ул. П. Морозова, д. 5, а/я «Первое сентября».

Телефоны для справок: (0852) 45-07-77, (095) 249-28-77.

КУПОН *Книга – почтой*

Издательский дом **Первое сентября**

ЗАПОЛНЯЕТСЯ ПЕЧАТНЫМИ БУКВАМИ!

ФАМИЛИЯ

ИМЯ

ОТЧЕСТВО

ИНДЕКС АДРЕС

НОМЕР ЛОТА — — КОЛИЧЕСТВО ЭКЗЕМПЛЯРОВ

НОМЕР ЛОТА — — КОЛИЧЕСТВО ЭКЗЕМПЛЯРОВ

НОМЕР ЛОТА — — КОЛИЧЕСТВО ЭКЗЕМПЛЯРОВ

НОМЕР ЛОТА — — КОЛИЧЕСТВО ЭКЗЕМПЛЯРОВ

НОМЕР ЛОТА — — КОЛИЧЕСТВО ЭКЗЕМПЛЯРОВ

00-1-02

ОБРАЗЕЦ: 0123456789

Указанная в купоне цена включает в себя стоимость доставки наземным транспортом от издательства до вашего почтового отделения (при авиаперевозках сумма наложенного платежа увеличивается на авиатариф). Рассылка производится только на территории РФ. Если полученная бандероль содержит неполный заказ, это означает, что заказ разбит на две или более бандероли.

"ЖАРКОЕ ЛЕТО-2002"

Жаркое лето
1111010010

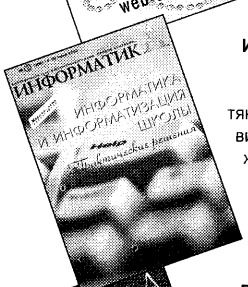
Уникальный комплект замечательных материалов к новому учебному году

Д.М. Златопольский.
Задачник по электронным таблицам (Excel)

Вопросы, связанные с обработкой информации с помощью электронных таблиц, занимают важное место в школьном курсе информатики. Но специализированного школьного задачника по электронным таблицам нет. Вернее, не было, а теперь есть. Его первая часть будет опубликована в летних номерах нашей газеты.

А.И. Терентьев.
Организация школьного web-сайта

Как сделать школьный web-сайт "с нуля"? Как связать компьютеры в сеть, какое программное обеспечение выбрать и как его установить? Как, наконец, заставить все это "хозяйство" работать? В одном из летних номеров мы познакомим наших читателей с ответами на эти (и не только на эти!) вопросы.



А.И. Сенокосов.
Информатика и информатизация школы. Практические решения

Допустим, вы сделали школьный web-сайт. Протянули провода, настроили локальную сеть, установили программное обеспечение. Все работает. А что же дальше? Как организовать информационное наполнение сайта? Как "встроить" web-сайт в школьную жизнь? Все, кто решал эти вопросы, знают, что они-то и являются настоящими **вопросами**.

Д.М. Златопольский.
Внеклассная работа по информатике. Избранные задания

Летом мы предложим подписчикам целый номер с заданиями, которые можно использовать на викторинах, конкурсах и других внеклассных мероприятиях по информатике. В него войдут множество новых заданий, а также лучшие из опубликованных ранее материалов популярной рубрики нашей газеты.

О Г.А. Звенигородском.
Редактор-составитель — Н.А. Юнерман

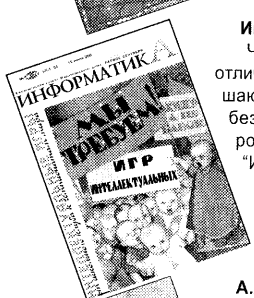
История становления школьной информатики не может быть написана без страниц, посвященных исследованиям и разработкам Г.А. Звенигородского. И сегодня остается актуальным воплощенное в них единство тематических и педагогических сторон работы с учащимися, живой практики и теоретического осмысления материала. 9 августа 2002 г. Г.А. Звенигородскому исполнилось бы 50 лет.

А.А. Дуванов.
DHTML-конструирование

"Бумажная" версия электронного учебника продолжает роботландский курс гипертекстового конструирования (ранее были опубликованы "HTML-конструирование" и "JavaScript-конструирование"). Новый учебник посвящен созданию динамических интерактивных приложений. В нем изложены основы CSS (каскадные таблицы стилей) и показаны способы управления содержимым страницы при помощи воздействий на гипертекстовую модель документа.

Л.О. Сергеев.
Уроки по теме "Базы данных"

В этом тематическом выпуске мы предложим вниманию читателей цикл уроков по теме "Базы данных". В качестве основного инструмента для изучения этой темы автор предлагает использовать язык SQL. Теоретический материал подкрепляется большим количеством разноуровневых заданий.



Интеллектуальные игры

Что такое спортивное "Что? Где? Когда?" и чем оно отличается от телевизионного? Какие головоломки решают на чемпионатах мира? Во что можно поиграть без компьютера? Это и многое другое, а также вопросы, вопросы, вопросы... в специальном выпуске "Интеллектуальные игры".

А.А. Дуванов.
Азы информатики. Книга 3 — "Пишем на компьютере"

"Бумажная" версия третьей книги нового интерактивного курса для малышей "Азы информатики" (первая книга — "Знакомство с компьютером" — была опубликована в № 1, 2, публикация второй — "В мире информации" продолжается в текущих номерах). Заглавная тема третьей книги (современная обработка текстов) нагружена "анатомией" трех китов информатики: хранение, передача и обработка информации.

Дорогие коллеги! "Информатика" распространяется только по подписке. Подписаться на нашу газету можно по каталогу "Роспечати", индекс подписки для индивидуальных подписчиков – 32291.

Гл. редактор
С.Л. Островский
Зам. гл. редактора
А.И. Сенокосов
Редакция:
Е.В. Андреева
Н.Л. Беленькая
Л.Н. Картвелишвили
Н.П. Медведева
Дизайн и верстка:
Н.И. Пронская
Корректоры:
Е.Л. Володина,
С.М. Подберезина

Адрес редакции и издателя:
121165, Киевская, 24
тел. 249-48-96
Отдел рекламы
тел. 249-98-70

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта изданий 32744

Тел.: (095)249-31-38, 249-33-86. Факс (095)249-31-84

Учредитель: ООО "Чистые пруды"

Зарегистрировано в Министерстве РФ по делам печати. ПИ № 77-7230 от 12.04.2001.
Отпечатано в ОИД "Медиа-Пресса",
125993, ГСП-3, Москва, А-40, ул. "Правды", 24.
Тираж 6100 экз.
Срок подписания в печать по графику 19.06.2002.
Номер подписан 19.06.2002.
Заказ № 14154
Цена свободная

Internet: inf@1september.ru
WWW: http://www.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ», ГЛАВНЫЙ РЕДАКТОР — А.СОЛОВЕЙЧИК

Газеты ИЗДАТЕЛЬСКОГО ДОМА: **Первое сентября** — гл. ред. Е.Бирюкова, **Английский язык** — гл. ред. А.Громушкина, **Библиотека в школе** — гл. ред. О.Громова, **Биология** — гл. ред. Н.Иванова, **Воскресная школа** — гл. ред. монах Киприан (Ященко), **География** — гл. ред. О.Коротова, **Дошкольное образование** — гл. ред. М.Аромштам, **Здоровье детей** — гл. ред. А.Лекманов, **Информатика** — гл. ред. С.Островский, **Искусство** — гл. ред. Н.Исмаилова, **История** — гл. ред. А.Головатенко, **Литература** — гл. ред. Г.Красухин, **Математика** — гл. ред. И.Соловейчик, **Начальная школа** — гл. ред. М.Соловейчик, **Немецкий язык** — гл. ред. М.Бузоева, **Русский язык** — гл. ред. Л.Гончар, **Спорт в школе** — гл. ред. Н.Школьникова, **Управление школой** — гл. ред. А.Адамский, **Физика** — гл. ред. Н.Козлова, **Французский язык** — гл. ред. Г.Чесновицкая, **Химия** — гл. ред. О.Блохина, **Чудесная газета** — гл. ред. М.Аромштам, **Школьный психолог** — гл. ред. М.Сартан.