МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ УЗБЕКИСТАН
ТАШКЕНТСКИЙ ИНСТИТУТ ТЕКСТИЛЬНОЙ И ЛЕГКОЙ
ПРОМЫШЛЕННОСТИ

**Кафедра: «Информатики»**

# *ЛАБОРАТОРНАЯ РАБОТА*

Публикация WEB документов в зарубежных сетях

Выполнил: Умархажаев Х. Н
Группа 1р-09
Проверила: Яфарова Г. М.

**Ташкент-2010**

# ПУБЛИКАЦИЯ WEB ДОКУМЕНТОВ В ЗАРУБЕЖНЫХ СЕТЯХ

## 1. Модель объектов javascript - объекты navigator'a

Идея JavaScript очень проста. Все операции, которые можно исполнять в программе на JavaScript, описывают действия над хорошо известными и понятными объектами, которыми являются элементы рабочей области программы Netscape Navigator и контейнеры языка HTML. Собственно объектная ориентированность JavaScript на этом и кончается. Есть только объекты с набором свойств и набор функций над объектами. Последние называются методами. Кроме методов существуют и другие функции, которые больше похожи на функции из традиционных языков программирования и позволяют работать со стандартными математическими типами или управлять процессом выполнения программы. Еще в JavaScript есть события - аналог программных прерываний. Эти события также ориентированы на работу в World Wide Web, например, загрузка страницы в рабочую область Navigator'a или выбор гипертекстовой ссылки. Используя события, автор гипертекстовой страницы и программы ее отображающей может организовать просмотр динамических объектов, например, бегущая строка, или управление многооконным интерфейсом.

## 2. Методы объектов и свойства объектов. управление потоком вычислений

Каждый из этих классов имеет функции управления объектами класса - методы. Самыми главными их этих методов являются те, которые позволяют переназначать значения объектов. Делается это обычно по операции присваивания. Вообще, все типы операторов, которые поддерживаются обычными языками программирования, реализованы JavaScript (+,-,*, /, %, >>,<<, +=, -=, ...). При этом оператор сложения "+" при работе со строками означает конкатенацию последних, т.е. добавление в конец строки новую строку:

```
s = "string1"+"string2"
```

Кроме операций с числами и описаний стандартных классов в JavaScript есть команды управления потоком вычислений:

- *break* - принудительный выход из цикла;
- while(i &lt 6)
  if(i==3) break;

*continue* - переход в конец цикла;

- while(i &lt 6)
  if(i==3) continue;
- *for* - цикл;
- for(i=0;i<9;i++)

*for* - цикл свойств объекта (переменных определенных в классе);

- for(i in obj)
- str = obj[i]
- *if..else* - условный оператор;
- if(i>0)
- ...
- else
- ...
- *wile* - условный цикл;
- wile(j==k)
- j++;
- k--;

*var* - оператор объявления переменной.

var kuku = "kuku"

Тип переменной определяется по присвоенному ей значению.

Перечисленные здесь операторы не представляют полного перечня операторов JavaScript, но их вполне достаточно для выполнения практических занятий.

## 3. События, массивы n графика

Важным элементом языка являются события. Программист использует события для выполнения определенных частей программного кода скрипта. Один из наиболее часто используемых приемов - исполнение определенных действий в момент загрузки страницы в Navigator.

Не будем перечислять все события, но упомянем о наиболее часто используемых:

- *onLoad* - выполнение скрипта или функции при загрузке;
- *onChange* - порождается при изменении значения элемента формы;
- *onClick* - порождается при выборе объекта (button, checkbox и т.п.);
- *onSelect* - порождается при выборе текстового объекта (text, textarea);
- *onSubmit* - при нажатии на кнопку Submit;
- *onUnload* - при переходе к другой странице.

Появление Netscape Navigator 3.0 и новой версии JavaScript 1.1 заставляет продолжить обзор возможностей управления сценариями просмотра Website, который был опубликован в предыдущем выпуске "Открытых Систем Сегодня" (CW N 46, 1996). В новой версии языка были введены: возможность взаимодействия JavaScript и Java, определение установленных plug-ins, определены новые типы объектов (Area, Function, Image) и ряд других особенностей, которые по мнению разработчиков должны повысить мощь программирования на JavaScript.

Первый тип новых объектов, которые мы рассмотрим, являются массивы. Тип "Array" введен в JavaScript 1.1 для возможности манипулирования самыми разными объектами, которые отображаются Navigator'ом. Это - список всех гипертекстовых ссылок данной страницы Website, список всех картинок на данной странице, список всех applet'ов данной страницы, список всех элементов формы и т.п. Пользователь может создать и свой собственный массив, используя конструктор Array(). Делается это следующим образом:

```
new_array = new Array()
new_array5 = new Array(5)
```

```
colors = new Array ("red","white","blue")
```

Размерность массива может динамически изменяться. Можно сначала определить массив, а потом присвоить одному из его элементов значение. Как только это значение будет присвоено, изменится и размерность массива:

```
colors = new Array()
colors[5] = "red"
```

В данном случае массив будет состоять из 6 элементов, т.к. первым элементом массива считается элемент с индексом 0. Для массивов определены три метода: join, reverse, sort. Join объединяет элементы массива в строку символов, в качестве аргумента в этом методе задается разделитель:

```
colors = new Array("red","white","blue")
string = acolors.join("+")
```

В результате выполнения присваивания значения строке символов string мы получим следующую строку:

```
string = "red + white + blue"
```

Другой метод, reverse, изменяет порядок элементов массива на обратный, а метод sort отсортировывает их в порядке возрастания. У массивов есть два свойства: length и prototype. Length определяет число элементов массива. Если нужно выполнить некоторую рутинную операцию над всеми элементами массива, то можно воспользоваться циклом типа:

```
color = new Array("red","white","blue")
n = 0
while(n != colors.length)
{.... операторы тела цикла ...}
```

Свойство prototype позволяет добавить свойства к объектам массива. Однако наиболее часто, в программе на JavaScript используются встроенные массивы, главным образом графические образы (Images) и гипертекстовые ссылки (Links).

До Navigator 3.0 в JavaScript использовались только встроенные объекты типа Image. В новой версии языка появился конструктор для этого типа объектов:

```
new_image = new Image()
new_image = new Image (width,height)
```

Часто для целей создания мультипликации создают массивы графических объектов, которые потом прокручивают один за другим:

```
img_array = new Array()
img_array[0] = new Image(50,100)
img_array[1] = new Image(50,100)
....
img_array[99] = new Image(50,100)
```

У объекта Image существует 10 свойств, из которых, пожалуй, самым важным является src. Так, для присваивания конкретных картинок элементам массива img_array следует воспользоваться следующей последовательностью команд:

```
img_array[0].src = "image1.gif"
img_array[1].src = "image2.gif"
...
img_array[99].src = "image100.gif"
```

В данном случае можно было воспользоваться и циклом для присвоения имен, так как они могут быть составлены из констант и значения индексной переменной. В новой версии языка объект типа Image можно поименовать в HTML-теге IMG. После этого можно обращаться к нему по имени. При этом следует учитывать, что если Image используется внутри формы, то он является свойством этой формы. Это значит, что для следующего графического объекта должны быть использованы разные составные имена:

```
<img name=car src=car.gif>
<--- Встроенный в документ объект
document.car.src = "car1.gif"
```

```
 <form name=kuku>
<img name=car src=car.gif>
 <-- Встроенный в форму документ.
</form>
document.kuku.car.src = "car1.gif"
```

Однако, наиболее часто в примерах использования скриптов можно встретить обращение к Image по индексу в массиве всех графических объектов данной страницы. Если наш объект, например, является вторым Image на странице, то будь он внутри формы или за ее пределами, к нему всегда можно обратиться по индексу:

```
document.images[1].src = "car1.gif"
```

Расширяя наш пример с массивом Image построим теперь документ, в котором будет встроена мультипликация, определенная нашим массивом:

*Пример :*

```
<HTML>
<HEAD>
<SCRIPT>
function multi_pulti()
img_array = new Array()
img_array[0] = new Image(50,100)
....
img_array[99] = new Image(50,100)
img_array[0].src = "image1.gif"
...
img_array[99].src = "image100.gif"
n=0
while(n==0)
document.images[0].src = img_array[0].src
...
</SCRIPT>
```

```
</head>
<body onLoad="multi_pulti()">
<img src=image1.gif>
</body>
</html>
```

Довольно часто используют не мультипликацию, а выбор картинки через OPTION, другой новый объект JavaScript. При этом можно через поле формы SELECT менять не только саму картинку, но и гипертекстовую ссылку, которая может быть связана с этой картинкой. На гипертекстовую ссылку также можно ссылаться по индексу:

```
document.links[index].href = kuku.html
```

Данный прием оправдан и с точки зрения интерфейса навигатора. При использовании такого сорта массивов ссылок не требуется их длительное перечисление и листание страницы в рабочей области навигатора - можно просто выбрать ссылку из "выпадающего" меню. Другой способ для сокращения числа нажатий на клавиши - использование событий. В том же объекте OPTION можно использовать событие onChange, что делает необязательным нажатие кнопок типа submit. В этом случае достаточно будет просто выбрать альтернативу и перейти к новой странице сразу после выбора.

## СТЕКИ ГИПЕРТЕКСТОВЫХ ССЫЛОК

Не обошли своим внимание авторы JavaScript и стеки гипертекстовых ссылок. В язык теперь введен новый тип объектов типа Area. Area - это элемент контейнера MAP, который определяет client-site imagemap. Собственно, главное достоинство такого объекта состоит в том, что гипертекстовые ссылки, которые определены в AREA, стали доступны для переопределения. Они появляются в массиве обычных ссылок страницы, и можно как получить значение URL, так и переопределить его. К объекту AREA нельзя обратиться по имени. Можно использовать только индекс массива гипертекстовых ссылок документа.

В контексте стека гипертекстовых ссылок интересно рассмотреть еще одну возможность JavaScript, связанную с переходом по гипертекстовой ссылке вообще. В обычном случае параметр HREF контейнера A должен иметь какое-нибудь значение. Если, например, по событию onClick необходимо открыть новое окно и в старом сохранить отображенный документ, то его URL следует указывать в качестве значения HREF. В противном случае, в старое окно будет загружена пустая страница, если HREF=" ". В новой версии JavaScript введена функция void. Точнее тип void, который означает отсутствие какого-либо значения. Если нам необходимо выполнить некоторые действия при выборе гипертекстовой ссылки, но при этом не перегружать текущие страницы, то в параметре HREF можно указать конструкцию:

```
<A HREF="javascript:void(0)">kuku</A>
```

Таким приемом часто пользуются при программировании событий, связанных с проходом манипулятора мыши через поле гипертекстовой ссылки.

### 1. Фреймы и окна

При работе с фреймами и окнами в предыдущих версиях JavaScript постоянно приходилось отслеживать последовательность открытия окон и фреймов, для того, чтобы аккуратно их потом закрывать. На некоторых неточностях работы с окнами были основаны так называемые mail-bombs. Суть этих "подарков" заключалась в том, что если пользователь по почте принимает документ, который состоит только из одной команды:

window.close()

то система не спрашивая пользователя закрывала текущее окно, а в этот момент таким окном является окно электронной почты. Теперь, перед тем как что-либо закрыть, система будет спрашивать разрешения. Правда, опять не всегда. Если в момент получения команды на закрытие окна на экране только одно окно Navgator, то система его закроет без каких-либо комментариев.

Однако, работа с окнами и фреймами на этом не исчерпывается. Во-первых, в систему введено новое свойство opener, которое определено для текущего окна или фрейма, и методы blur и focus распространены на работу с окнами. Свойство opener определяет окно документа, который вызвал открытие окна текущего документа. Свойство определено для любого окна и фрейма. Необходимо выполнить некоторые функции по отношению к окну, открывшему данное окно, и можно использовать выражение типа:

window.opener.[method]

Например, если требуется закрыть окно-предшественник, то можно просто выполнить метод close:

window.opener.close()

Точно таким же способом можно изменить содержание этого окна при помощи методов write или writeln. Можно менять и другие свойства объектов в окне предшественнике. Следующий пример взят из дополнений к спецификации JavaScript:

window.opener.document.bgColor='cyan'

В данном случае для окна предшественника определен светло голубой цвет в качестве цвета фона. Но самым замечательным является то, что предшественника можно менять. Это значит, что автор получает возможность открывать и закрывать окна не строго иерархической последовательности, а в произвольном порядке. Управление многооконным интерфейсом в этом случае становится более гибким:

window.opener= new_window

window.opener = null

Первый пример переназначает для текущего окна окно-предшественник, в то время как второй вообще защищает предшественника от каких-либо действий. Все, что было сказано об окнах, распространяется и на фреймы, которые являются просто частным случаем окна. При работе с фреймами часто фрейм может быть порожден путем разбиения другого фрейма на части. Если при этом потребуется обратиться к окну-предшественнику фрейма-предшественника, то свойство opener в этом случае незаменимо. Разработчики языка следуют за пожеланиями авторов Websit'ов и практикой применения JavaScrip, которая на начальной стадии разработки языка не была столь очевидной. Кроме обращения к различным свойствам окон и фреймов разработчики расширили действие методов blur и focus с фреймов до окон. Теперь не только фрейм, но и окно может быть сделано текущем с использованием метода focus или, наоборот, переведено в фон при помощи метода blur. В ряде случаев, при порождении нескольких страниц, например, обращение к этим функциям бывает довольно полезным.

**2. Наследование кода скриптов различными страницами**

Отсутствие какого-либо наследования между различными страницами Website заставляло разработчиков перетаскивать из одной страницы в другую довольно большое количество часто используемых функций и переменных. Разговоры о том, что было бы неплохо получить возможность доступа к глобальным ресурсам или возможность определять такие глобальные ресурсы, ведутся с самого момента появления JavaScript. К сожалению, стройного логичного механизма передачи параметров, функций и переменных от одного окна или фрейма другому нет и в JavaScript 1.1. Однако, продвижение в этом направлении есть.

У контейнера SCRIPT появился атрибут SRC. Это дает возможность авторам строить своеобразную библиотеку функций, к которым можно обращаться из любой страницы, в которой будет ссылка на такую библиотеку. При этом вовсе необязательно размещать саму библиотеку на том же сервере, где размещены и гипертекстовые страницы Website. Можно использовать и

чужие функции, написанные кем-либо из ветеранов программирования на JavaScript на другом конце Земли. В атрибуте SRC используется обычный URL. Внутри файла скриптов не используются теги SCRIPT. Это обычный файл с использованием определений функций и переменных. Естественно, что использование чужих скриптов может обернуться не только полезными приобретениями, но и непредсказуемыми проблемами. Для обычных пользователей страниц Website подкачка большого количества скриптов может стать просто дополнительным источником затрат. Наиболее вероятным случаем может стать использование одной-двух функций из библиотеки на одной странице, а качать придется всю библиотеку. Другой новой возможностью работы с функциями стало введение нового объекта Function. Объект Function порождается конструктором Function:

```
new_Function = new Function(arg1,arg2,...,argn,
 function_body)
```

Главное отличие от обычного декларирования функции заключается в том, что в данном случае порождена переменная new_Function, с которой можно работать, как с любым другим объектом. При обычном переделении функции такой переменной не порождается. Как любой объект Function имеет свои свойства, но не имеет методов. В качестве свойств функции выступают аргументы и возможность назначения новых свойств через prototype. В заключении разговора о функциях и наследовании хочется еще раз обратить внимание на свойство opener окон и фреймов. Это свойство можно использовать при обращении к объектам страницы-родителя, что позволяет компенсировать отсутствие наследования и глобальных переменных в JavaScript.

### 3. Java, javascript и plug-ins

В новой версии языка есть возможность организовать взаимодействие между Java-applet'ами и JavaScrip-скриптами. Достигается это за счет использования атрибута MAYSCRIPT в контейнере APPLET. Собственно в JavaScript определен объект типа APPLET, к которому можно обращаться либо

по имени, либо по индексу в массиве applet'ов. У этого объекта есть только одно свойство - имя. Ни какие-либо другие свойства или методы для данного типа объектов не определены. Сами детали взаимодействия applet'ов и скриптов лучше всего обсуждать в рамках программирования Java applet'ов, поэтому здесь мы эти особенности опустим.

Кроме applet'ов JavaScript позволяет работать и с Plug-ins. Последние представляют из себя массив соответствующего типа, для которого определен ряд свойств. Используя эти свойства можно определить установленные plig-ins и их соответствия MIME-типам. Назначить plug-ins или манипулировать ими нельзя.

**4. Встраивание в html-документ**

Для встраивания скриптов в тело HTML-документа используется контейнер SCRIPT. Не все программы просмотра способны распознавать и исполнять скрипты, поэтому само тело скрипта помещается в контейнер комментария. Для определенности рассмотрим небольшой пример:

```
TYPE=button VALUE="0-100"
            onClick="max_value=100"></TD>
  <TD><INPUT NAME=i+ TYPE=button VALUE=" + "
            onClick="set_sign('+')"></TD>
  <TD><INPUT NAME=i- TYPE=button VALUE=" - "
            onClick="set_sign('-')"></TD>
      </TR>
```

Приведенный здесь пример содержит датчик случайных чисел (функции init и rand), таблицу, реализующую функции кнопок клавиатуры, и блок проверки результата вычислений. После загрузки программы пользователь должен выбрать тип вычислений (+,-), интервал вычислений (в пределах 10, 20, 100) и нажать кнопку "?" для генерации примера. После ввода с отображаемой клавиатуры числа пользователь нажимает на символ "=", что означает "исполнить", и система проверяет правильность ответа. Если ответ правильный, то программа поздравляет фразой "Молодец!", если нет - "Думай!?". В системе

Windows 3.x нет встроенного датчика случайных чисел, поэтому стандартная функция rand в этой версии JavaScript не реализована. Используемый в данной программе датчик был позаимствован из телеконференции по JavaScript. В скрипте кроме этого используются объект типа "дата" и его методы, а также встроенные функции контроля вводимых данных. Как видно из примера обращение к полям HTML-формы представляет из себя обращение к структуре, корнем которой является объект окно, в котором определен объект документ, внутри которого определена форма и ее поля и атрибуты полей. Не у всех полей можно менять значения атрибутов, так, например, атрибут VALUE в кнопке не меняет своего значения, если только не перезагрузить страницу.

Другим часто встречающимся примером является бегущая строка. Строка может бежать либо в поле статуса (низ экрана), либо внутри поля формы. Рассмотрим такой пример.

**5. Единство в многообразии**

В заключении следует отметить, что JavaScript - это не единственный язык управления сценариями просмотра документов. Microsoft подготовила свою версию аналогичного языка - VBScript на основе Visual Basic. Кроме того, управлять сценарием просмотра можно и из Java-applet'ов, что конечно сложнее, но зато более надежно и безопасно. При этом программист получает все преимущества наследования и прочие атрибуты объектно-ориентированного программирования. В конце концов для создания фреймов и окон можно использовать атрибуты соответствующих контейнеров HTML, которые позволяют делить рабочую область экрана на фрагменты, перекрывать объекты и восстанавливать первоначальный вид страницы. Таким образом, к настоящему времени существует по меньшей мере три способа управления сценариями просмотра гипертекстовых баз данных Web, каждый из которых по своему хорош. Какой из них выбрать - это дело автора Website.

Следуя логике авторов JavaScript и направлениям развития World Wide Web следует ожидать появления новых типов объектов и изменения свойств существующих. Кроме этого, видимо, появятся глобальные переменные и

функции. Порождение абстрактных типов тоже не за горами, хотя от такой новации не очень понятно, кто выиграет. Постепенно совершенствуется и поддержка встроенных функций на разных платформах, так, например, функция random() реализована в настоящее время везде, что не исключает использование и своих собственных датчиков случайных чисел. И еще, скорее всего, следует ожидать компилятора JavaScript для клиента. Будет ли эта возможность встроена в Navigator или это будет отдельный модуль - пока не ясно, но появление библиотек функций - движение в этом направлении, хорошо согласующееся с принципами кэширования гипертекстовых страниц.

# ГИПЕРТЕКСТОВЫЕ ССЫЛКИ И КАРТИНКИ

Гипертекстовые ссылки и картинки - это свойства объекта "документ", который в свою очередь является частью объекта "окно". И гипертекстовые ссылки, и картинки составляют встроенные массивы, к которым можно обращаться по индексу. Рассмотрим несколько примеров программирования массивов гипертекстовых ссылок и картинок.

### 1. Просмотр фотографий образцов через список гипертекстовых ссылок

Вся страница размечается как таблица, состоящая из одной строки в две ячейки. В первой ячейке размещается список выбора альтернатив, во второй ячейке размещаются картинки, выбранные при помощи списка альтернатив. Все элементы списка помечены как гипертекстовые ссылки. У каждой ссылки определено событие onSelect. Для того, чтобы не происходил переход к другой странице, а выполнялась только обработка события, в поле href необходимо определить вызов JavaScript. Ниже приведен пример реализации такой компоновки и пример кода самой страницы.

*Пример.* Изменение картинки через гипертекстовую ссылку

```
<HTML>
<HEAD>
<TITLE>HTTPD\HTDOCS\JAVASCR\EXAMPLE1</TITLE>
<SCRIPT>
<!-- Защитили текст скрипта от старых браузеров
function l_image(a)
 document.images[1].src=a
  // -->   </SCRIPT>
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFCC"
LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<H1>
<FONT COLOR="#000099">Просмотр фотографий
```

```
образцов</FONT>
</H1>
<center>
<TABLE COLS=2 WIDTH="100%" >
<CAPTION>
<FONT COLOR="#000099" SIZE=+2>
Образцы бытовой техники</FONT>
</CAPTION>
<TR>
<TD>
<UL>
<LI><A HREF="javascript:l_image('freeze.gif')"
>Холодильник</A> </LI>
<LI><A HREF="javascript:l_image('dust.gif')"
>Пылесосы</LI>
<LI><A HREF="javascript:l_image('toster.gif')"
>Тостер</LI>
<LI><A HREF="javascript:l_image('cook.gif')"
>Варочный стол</LI>
<LI><A HREF="javascript:l_image('oven.gif')"
>Духовка</LI>
<LI><A HREF="javascript:l_image('wash.gif')"
>Стиральная машина</LI>
<LI><A HREF="javascript:l_image('dishwash.gif')"
>Посудомоечная машина</LI>
</UL>
</TD>
<TD ALIGN=CENTER VALIGN=CENTER><
IMG SRC="dust.gif" NAME="tool" > </TD>
</TR>
```

```
</TABLE>
</center>
</BODY>
</HTML>
```

В данном примере при выборе гипертекстовой ссылки происходит вызов функции l_image(), которая изменяет значение атрибута SRC в контейнере IMG. Таким образом, можно организовать просмотр различных товаров в одном единственном графическом окне.

### 2. Изменение картинки путем выбора предмета из списка

Вся страница размечается как таблица, состоящая из одного столбца в две ячейки. В первой ячейке размещается выпадающее меню выбора альтернатив, во второй ячейке размещаются картинки, выбранные при помощи списка альтернатив. При выборе альтернативы из меню происходит событие onChange, которое вызывает функцию l_image. Ниже приведен пример реализации такой компоновки и пример кода самой страницы.

### 3. Листание картинок и их автоматический просмотр

Вся страница размечается как таблица, состоящая из одного столбца в две ячейки. В первой ячейке размещаются три кнопки управления просмотром картинок "Вперед", "Старт/Стоп", "Назад". Во второй ячейке размещаются картинки, выбранные путем листания. Листание осуществляется кнопками "Вперед" и "Назад". Возможен и другой вариант выбора картинки, когда система сама через некоторый промежуток времени периодически меняет картинки. Для этого следует сначала запустить автоматическую смену картинок, нажав на кнопку "Старт/Стоп", а потом, когда будет показана нужная картинка, остановить автоматическое листание, снова нажав на кнопку "Старт/Стоп". Ниже приведен пример реализации такой компоновки и пример кода самой страницы.

# ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных / Пер. с англ,— М.: Мир,1989. — 360 с., ил.

2. Гринзоу Лу. Философия программирования для Windows 95/NT / Пер.с англ. — СПб.: Символ-Плюс, 1997. — 640 с., ил.

3. Зелковиц М., Шоу А., Гэннон Дж. Принципы разработки программного обеспечения / Пер. с англ. — М.: Мир,

1982. — 386 с., ил.

4. Практическое руководство по программированию / Пер. с англ. Б. Мик,П. Хит, Н. Рашби и др.; под ред. Б. Мика, П. Хит, Н. Рашби. — М.: Радио и связь, 1986. — 168 с., ил.

5. Фокс Дж. Программное обеспечение и его разработка / Пер. с англ. М.: Мир, 1985.-368с., ил.

6. Язык компьютера. Пер. с англ, под ред. и с предисл. В, М. Курочкина. — М.: Мир, 1989. — 240 с., ил.