Министерство образования и науки Республики Казахстан

Павлодарский государственный университет им. С. Торайгырова

Кафедра Информатики и информационных систем

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению практических работ
по дисциплине «Современные языки программирования»
для студентов специальностей 050602 – Информатика,
050703- Информационные системы
форма обучения очная и заочная

Павлодар

Лист утверждения к методическим указаниям

Форма
Ф СО ПГУ 7.18.1/05

УТВЕРЖДАЮ
Декан ФФМиИТ
_____С.К.Тлеукенов
«__»_____20___ г.

Составители**:** д.п.н., академик Нурбекова Ж.К.
старший преподаватель Бельгибаева С.А.

Кафедра Информатика и информационные системы

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению практических работ

«Современные языки программирования»
специальностей 050602 – информатика, 050703- Информационные системы

Рекомендована на заседании кафедры от «__»_____20___ г. Протокол №____.

Заведующий кафедрой _____ Нурбекова Ж.К.
(подпись)

Одобрена методическим советом факультета ФМиИТ
«___»_____20___г. Протокол №_____

Председатель МС_____ А.Т.Кишубаева
(подпись)

1 ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

1.1 Цель дисциплины: обучение студентов знаниям, умениям и навыкам применения современных методов, стилей и инструментальных средств при разработке программных продуктов.

1.2 Задачи дисциплины:

- обзор и анализ современных языков программирования;
- изучение методологии современных языков программирования.

1.3 В результате изучения дисциплины студенты должны знать:

- стратегический подход к современному программированию;
- базовые методологические навыки программирования;
- технологию объектно-ориентированного проектирования;
- компонентную технологию проектирование;
- основы синтаксиса и семантики языков программирования, принадлежащих к различным парадигмам.

1.4 В результате изучения дисциплины студенты должны уметь:

- определять в соответствии с постановкой задачи языковое средство для ее решения;
- решать задачи обработки информации с применением средств современных языков программирования.

1.5 Перечень дисциплин, знание которых необходимо для изучения курса: (указать разделы)

информатика, языки программирования, высшая математика, алгебра, теория алгоритмов и автоматов.

Практическая работа № 1
Тема - Разработка модели с БНФ

Цель работы – Научиться разрабатывать модели с использованием формы Бекуса – Науэра

Краткие теоретические сведения

Первое, что отличает один язык программирования от другого — это их синтаксис. Основное назначение синтаксиса — предоставить систему обозначений для обмена информацией между программистом и транслятором. Однако, при разработке деталей синтаксиса чаще исходят из второстепенных критериев, назначение которых: сделать программу удобной для чтения, написания и трансляции, а также сделать ее однозначной. Если удобство чтения и записи программ необходимы для пользователя языка программирования, то простота трансляции и отсутствие разночтений в языке имеют отношение к нуждам транслятора. Эти цели, в общем случае, противоречивы, и нахождение приемлемого компромисса при их решении является одной из центральных задач при разработке языка программирования.

Разработка нового языка программирования начинается с определения его синтаксиса. Для описания синтаксиса языка программирования, в свою очередь, нужен также некоторый класс. Язык, предназначенный для описания другого языка, называют *метаязыком*. Язык, используемый для описания синтаксиса языка, называют *метасинтаксическим* языком. В метасинтаксических языках используется специальная совокупность условных знаков, которая образует нотацию этого языка.

Исторически первым метасинтаксическим языком, который использовался на практике для описания синтаксиса языков программирования (в частности Алгола-60), являются *нормальные формы Бэкуса,* сокращенно обозначают БНФ — бэкусова нормальная форма или бэкусо-науровская форма. Основное назначение форм Бэкуса состоит в представлении в сжатом и компактном виде строго формальных и однозначных правил написания основных конструкций описываемого языка программирования.

Формальное определение синтаксиса языка программирования обычно называется *грамматикой.*

В форме Бэкуса описываются два класса объектов: это, во-первых, основные символы языка программирования и, во-вторых, имена конструкций описываемого языка, или так называемые, *металингвистические переменные.*

Каждая металингвистическая формула (форма) описывает правила построения конструкций языка и состоит из двух частей. В левой находится металингвистическая переменная, обозначающая соответствующую конструкцию. Далее следует *металингвистическая связка ::=*, означающая «определяется как» или «есть». В правой части формулы указывается один или несколько вариантов построения конструкции, определяемой в левой части. Для пост-

роения определяемой формулой конструкции нужно выбрать некоторый вариант ее построения из правой части формулы и подставить вместо каждой металингвистической переменной некоторые цепочки основных символов. Варианты правой части формулы разделяются металингвистической связкой |, имеющей смысл «или».

Сами металингвистические переменные обозначаются словами, поясняющими смысл описываемой конструкции, и заключаются в угловые скобки < >.

В качестве примера БНФ приведем определение десятичного целого числа:

1. < десятичное целое число > ::= < число без знака > |
   + < число без знака > | — < число без знака >
2. < число без знака > ::= < цифра > | < число без знака > < цифра >
3. <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

Например, число +294 выводится по формулам следующим образом:

< десятичное целое число >,

+ < число без знака >,

+ < число без знака > < цифра >,

+ < число без знака > 4,

+ < число без знака >< цифра > 4,

+ < число без знака > 94,

+ < цифра > 94,

+ 294.

Особенностью многих металингвистических формул является наличие в них *рекурсий*, т.е. использование для описания конструкций самих описываемых конструкций. Рекурсия может быть явной и неявной. *Явная рекурсия* имеет место, например, в правиле 2 в приведенном выше списке правил описания десятичного числа. *Неявная рекурсия* присутствует в случае, когда при построении конструкции на некотором шаге используется металингвистическая переменная, обозначающая саму эту конструкцию.

Наличие рекурсий затрудняет чтение и понимание металингвистических формул, однако это едва ли не единственный способ, позволяющий с помощью конечного числа правил, описать язык, который может содержать бесконечное число цепочек основных символов. Языки же программирования бесконечны — на них можно записать бесконечное число правильных программ, и при описании их синтаксиса с помощью БНФ всегда будут присутствовать явные или неявные рекурсии.

На практике для описания синтаксиса языков программирования применяются и другие металингвистические языки. Одна из целей их использования — устранить некоторую неестественность представления в БНФ общих синтаксических конструкций для необязательных, альтернативных и повторяющихся элементов правил. Так, для описания синтаксиса таких языков, как КОБОЛ и ПЛ/1, использовалась следующая нотация, являющаяся расширением БНФ:

- Необязательный элемент внутри правила заключается в квадратные скобки [. . .].
- Альтернативные элементы обозначаются вертикальным списком вариантов, заключенным в фигурные скобки {. . .}.
- Необязательные альтернативные варианты обозначаются вертикальным списком вариантов, заключенным в квадратные скобки [...].
- Повторяющийся элемент обозначается списком из одного элемента (заключенного, если это необходим, в фигурные или квадратные скобки) со следующим за ним обычным многоточием ....
- Обязательные ключевые слова подчеркиваются, а необязательные шумовые слова — нет.

Приводимое ранее описание БНФ десятичного числа в данной нотации будет иметь вид:

<десятичное целое > ::= [+ | -] <цифра > ...

0 4
1 5
2 6
3 7
8
9

<цифра>::=

Такое описание более компактно и естественно. Однако оба описания эквивалентны, т.е. любое правило, записанное на этом языке, может быть однозначно представлено в виде одной или нескольких форм Бэкуса, и наоборот.

Формы Бэкуса представляют более формальное описание языка и они, по существу, натолкнули исследователей на внедрение математических средств для системного описания и исследования языков программирования, использование математического аппарата как основы для синтаксического анализа в трансляторах, что позднее получило развитие в разнообразных методах синтаксического анализа, основанных на формальных синтаксических определениях.

Необходимо отметить, что БНФ не позволяет описывать контекстные зависимости в языке программирования. Например, такое ограничение Паскаль программ, как «идентификатор не может быть описан дважды в одном и том же блоке», нельзя описать средствами БНФ. Ограничения такого рода ближе уже к другой характеристике языка — семантике. Поэтому здесь используются другие средства, в общем случае называемые *метасемантическими языками.* Однако, как правило, ядром этих языков является та же БНФ.

Задание

1. Используя грамматику БНФ (Бекус-Науэра форма) - регулярные операций, опишите синтаксис программы (алгоритма).
2.
   - Создать императивную модель программирования.
   - Создать модель параллельного программирования.
   - Создать модель событийно-управляемого программирования.
   - Создать модель объектно-ориентированного программирования.
   - Создать модель функционального программирования.
   - Создать модель логического программирования.
   - Создать модель сочетания парадигм (выше перечисленных) программирования

Практическая работа №2-5
Тема - Операции над данными. Выбор. Циклы

Цель работы – Научить студентов основам программирования на языке Си. Научить разрабатывать программы для решения простых задач.

Краткие теоретические сведения

Операция присваивания

Самой общей операцией является присваивание, например, ratio = a/b или ch = getch(). В Си присваивание обозначается одним знаком равенства (=); при этом значение справа от знака равенства присваивается переменной слева.

Можно применять также последовательные присваивания, например: sum = a = b. В таких случаях присваивание производится справа налево, то есть b будет присвоено a, которая в свою очередь будет присвоена sum, так что все три переменных получат одно и то же значение (а именно, начальное значение b).

Одноместные и двуместные операции

Си поддерживает обычный набор арифметических операций:

- **умножение (*)**
- **деление (/)**
- **целочисленное деление (%)**
- **сложение (+)**
- **вычитание (-)**

Оператор if

Оператор if имеет следующий основной формат:

if (значение) оператор1; else оператор2;

где "значение" является любым выражением, которое приводится или может быть приведено к целочисленному значению. Если "значение" отлично от нуля ("истина"), то выполняется "оператор1", в противном случае выполняется "оператор2".

Дадим пояснение относительно двух важных моментов по использованию оператора if-else.

Во-первых, часть "else оператор2" является необязательной частью оператора if; другими словами, правомерно употребление следующей формы оператора if:

if (значение)
оператор1;

В этой конструкции "оператор1" выполняется тогда и только тогда, когда "значение" отлично от нуля. Если "значение" равно нулю, "оператор1" пропускается и программа продолжает выполняться дальше.

Во-вторых, что делать если вы хотите выполнить более одного оператора в зависимости от того ложно или истинно выражение, указанное в операторе if? Ответ: используйте составной оператор. Составной оператор состоит из:

- левой или открывающей фигурной скобки ()
- последовательности операторов, разделенных между собой точкой с запятой (;)
- правой или закрывающей фигурной скобки ()

В приведенном ниже примере в предложении if используется один оператор

```
if (b == 0.0)
printf("Отношение не определено\n");
а в предложении else - составной оператор
- 447,448 -
else
ratio = a/b;
printf( "Значение отношения равно %f\n", ratio);
```

Вы можете так же заметить, что тело вашей программы (функции main) является подобием составного оператора.

Цикл while

Оператор while имеет следующий формат:

while (выражение)

оператор

где "выражение" принимает нулевое или отличное от нуля значение, а "оператор" может представлять собой как один оператор, так и составной оператор.

В процессе выполнения цикла while вычисляется значение "выражения". Если оно истинно, то "оператор", следующий за ключевым словом while, выполняется и "выражение" вычисляется снова. Если "выражение" ложно, то цикл while завершается и программа продолжает выполняться дальше.

Цикл for

Основной формат цикла for:

for (выр1; выр2; выр3)

оператор

Так же, как и в цикле while, "оператор" в теле цикла for обычно является одним из операторов программы, но может использоваться и составной оператор, заключенный в фигурные скобки (...).

Заметим, что параметры цикла for, заключенные в скобки, должны разделяться точкой с запятой (позиционный параметр), которая делит в свою очередь пространство внутри скобок на три сектора. Каждый параметр, занимающий определенную позицию, означает следующее:

- выр1 - обычно задает начальное значение индексной переменной;
- выр2 - условие продолжения цикла;
- выр3 - обычно задает некоторую модификацию (приращение) индексной переменной за каждое выполнение цикла.

Задания

1. Вывести на экран сумму всех цифр введенного пользователем натурального числа.
2. Вывести на экран произведение квадратов всех цифр введенного пользователем натурального числа.
3. Вводится натуральное число. Напечатать число, полученное перестановкой его цифр в обратном порядке. Например, введено число 12345. Программа должна вывести число 54321.
4. Вводится натуральное число. Напечатать его цифровой корень. Цифровой корень образуется следующим образом:
   A) Складываются все цифры числа
   B) Если получено неоднозначное число, то его цифры снова складываются
   C) И так до тех пор, пока не получим одну цифру.
   D) Например, ввели число 12345. Складываем его цифры и получаем 15. Снова складываем. Цифровой корень= 6.
5. Напечатать таблицу из двух колонок. В 1-й колонке – натуральные числа в промежутке от 100 до 120, во второй – сумма цифр этих чисел.
6. Вводится натуральное число N. Вывести все трехзначные числа, сумма цифр которых равна введенному числу N.

7. Найти все трехзначные натуральные числа, сумма кубов цифр которых равна самому числу. Например, 153 – одно из таких чисел, т.к. 1^3+5^3+3^3=1+125+27=153.

8. Напечатать все натуральные числа – «перевертыши» в промежутке от 100 до 1000. Перевертышем назовем такое число, которое одинаково читается слева направо и справа налево.

9. Вводится натуральное число. Напечатать его наименьшую цифру.

10. Вводится натуральное число. Определить, является ли оно простым.

Ответы №1- №3

№ 01

```
#include <conio.h>
#include <stdio.h>
//Вывести на экран сумму всех цифр
//введенного пользователем числа:
int main(void)
{  unsigned long N,Sum=0;
   clrscr();
   printf("\n Input number: ");scanf("%ld",&N);
   /*do
     Sum+=(N % 10);
   while ((N=N/10)!=0);*/
   for (Sum=0;N;Sum+=N % 10,N=N/10);
   printf("Sum=%ld",Sum);
   getch();
   return 0;
}
```

№ 02

```
#include <conio.h>
#include <stdio.h>
//Вывести на экран произведене квадратов всех цифр
//введенного пользователем числа:
int main(void)
{  unsigned long N,Prod=1;
   clrscr();
   printf("\n Input number: ");scanf("%ld",&N);
   do
     Prod*=(N % 10);
   while ((N=N/10)!=0);
   printf("Prod=%ld",Prod);
   getch();
   return 0;
}
```

№ 03

```
#include <conio.h>
#include <stdio.h>
//Вводится натуральное число.Вывести число,
//полученное перестановкой его цифр в обратном порядке:
int main(void)
{  unsigned long N,N2=0;
   clrscr();
    printf("\n Input number: ");scanf("%ld",&N);
    for (;N;N2=N2*10+(N % 10),N/=10);
    printf("N2=%ld",N2);
```

```
  getch();
  return 0;
}
```

Практическая работа №6-9

Тема - Массивы и строки. Указатели. Структуры

Цель работы - Привить у студентов навыки программирования разных задач с массивами и строками на языке Си.

Краткие теоретические сведения

Большинство языков высокого уровня - включая Си - позволяют определять МАССИВЫ, т.е. индексированный набор данных определенного типа.

Пример:

```
main()
    #define NUMINTS 3
    int list[NUMINTS],i;
    list[0] = 421;
    list[1] = 53;
    list[2] = 1806;
    printf("Список адресов:");
    for(i=0; i<NUMINTS; i++)
    printf("%p ",&list[i]);
    printf("\nСписок значений:");
    for (i=0; i<NUMINTS; i++)
    printf("%4p ",list[i]);
    printf("\n");
```

Выражение **int list[NUMINTS]** объявляет **list** как массив переменных типа **int**, с объемом памяти, выделяемым для трех целых переменных.

К первой переменной массива можно обращаться как к **list[0],** второй - как к **list[1]** и третьей - как к **list[2].**

В общем случае описание любого массива имеет следующий вид:

type name[size]

(тип имя [размер])

где **type** - тип данных элементов массива (любой из допустимых в языке), **name** - имя массива.

Первый элемент массива - это **name[0],** последний элемент - **name[size-1];** общий объем памяти в байтах определяется выражением **size*(sizeof(type)).**

Массивы и строки

Мы говорили о строках в предыдущей главе и обращались при описании строк к двум немного различным способам: мы описывали строку как указатель на символы и как массив символов. Теперь вы можете лучше понять это различие.

Если вы описываете строку как массив типа char, то память для этой строки резервируется автоматически. Если же вы описываете строку как указатель на тип данных char, то память не резервируется: вы должны либо сами выделить ее (используя функцию malloc или ей подобную), или же присвоить ей адрес существующей строки.

Задания

11. Вывести на экран таблицу простых чисел от 1 до 1500

Программа:

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <iostream.h>
//Напечатать таблицу простых чисел от 1 до 1500.
int Prime(unsigned long N)
{unsigned long I=2;
    float sqrtN=sqrt(N);
    while ((I<sqrtN)&&(N % I)) I++;
    if (N%I) return 1;
    return 0;
}
int main(void)
{  unsigned long N,I;
   clrscr();
   for(I=2;I<=1500;I++)
   {
    //if (Prime(I)&&Prime(I+2)) cout<<I<<"<>"<<I+2<<"\t";
    if (Prime(I)) cout<<I<<"\t";
   }
   getch();
   return 0;
}
```

12. Подсчитать количество слов и символов в тексте. Словом считается любая последовательность символов, не содержащая пробелов, табуляций и новых строк.

13. Даны натуральное число n, действительные числа a_1, … , a_n. Получить удвоенную сумму всех положительных членов последовательности a_1, … , a_n.

14. Даны натуральное число n, символы s_1, … , s_n. Выяснить, имеются ли в последовательности s_1, … , s_n такие члены последовательности s_i, s_{i+1}, что s_i – это запятая, а s_{i+1} – тире.

15. Даны натуральные числа n_1, … , n_{20}, действительные числа $x_{1,}$,…,$x_{20.}$ Вычислить

$$\frac{n_1 x_1 + ... + n_{20} x_{20}}{n_1 + ... + n_{20}}$$

Практическая работа № 10-13

Тема – Работа с файлами

Цель работы – Привить у студентов навыки работы с файлами, считывание из файла и запись информации в файл.

Краткие теоретические сведения

В тех случаях, когда программа обрабатывает достаточно большой объем данных, последние обычно организуются и хранятся вне оперативной памяти ЭВМ. Наиболее эффективным устройством для организации внешнего хранения данных являются диски. Прежде чем читать или записывать иформацию в файл, надо открыть его с помощью стандартной библиотечной функции fopen. Программа, использующая эту функцию, должна включать во время компиляции системный файл stdio.h, в котором определен новый тип данных - FILE.

В программе нужно описывать ссылки на файлы и выглядит это, например, так: FILE *fu; Здесь fu означает указатель на FILE, а fopen выдает ссылку на этот файл. Функция fopen имеет следующий заголовок:

FILE *fopen(char *fname, char type);

Обращение к fopen в программе делается так: fu=fopen(fname, type); Строка символов fname содержит имя файла, который надо открыть; type- тоже строкасимволов, заключенная в кавычки и указывающая, как бует использоваться файл: "r"-чтение, "w" - запись, "r+" - чтение с дозаписью, "a" -дозапись. Функция fopen возвращает указатель, с помощью которого мы в дальнейшем будем обращаться к этому файлу. Примеры:FILE *uin, *uout;

uin=fopen("MAK1","r");

uout=fopen("MAK2","w");

Файл с именем MAK1 открывается для чтения и далее идентифицируется как uin; файл MAK2 открывается для записи и связывается с идентификатором uout.

Задания

1. Дан файл, компоненты которого являются действительными числами. Найти:

A) сумму компонент

B) произведение компонент

C) сумму квадратов компонент

2. Введенные сведения о студенте сохранить в файле.

3. Создать тестирующую программу, где вопросы и варианты ответов сохранены в отдельном файле

Практическая работа № 14- 17

Тема - Программирование на языке C#

Цель работы – Привить студентам навыки работы в среде Visual Studio.Net и программирования на языке C#

Краткие теоретические сведения

В программных продуктах .Net за этим именем стоит вполне конкретное содержание, которое предполагает, в частности, наличие открытых стандартов коммуникации, переход от создания монолитных приложений к созданию компонентов, допускающих повторное использование в разных средах и приложениях. Возможность повторного использования уже созданных компонентов и легкость расширения их

функциональности - все это непременные атрибуты новых технологий. Важную роль в этих технологиях играет язык XML, ставший стандартом обмена сообщениями в сети.

Не пытаясь охватить все многообразие сетевого взаимодействия, рассмотрим реализацию новых идей на примере *Visual Studio .Net* - продукта, важного для разработчиков.

Visual Studio .Net - открытая среда разработки

Среда разработки *Visual Studio .Net* - это уже проверенный временем программный продукт, являющийся седьмой версией Студии. Но новинки этой версии, связанные с идеей .Net, позволяют считать ее принципиально новой разработкой, определяющей новый этап в создании программных продуктов. Выделю две важнейшие, на мой взгляд, идеи:

- *открытость* **для языков программирования;**
- принципиально новый подход к построению **каркаса среды - Framework .Net**.

Открытость

Среда разработки теперь является открытой языковой средой. Это означает, что наряду с языками программирования, включенными в среду фирмой Microsoft - **Visual C++ .Net** (с управляемыми расширениями), **Visual C# .Net**, **J# .Net**, **Visual Basic .Net**, - в среду могут добавляться любые языки программирования, компиляторы которых создаются другими фирмами-производителями. Таких расширений среды **Visual Studio** сделано уже достаточно много, практически они существуют для всех известных языков - **Fortran** и **Cobol**, **RPG** и **Component Pascal**, **Oberon** и **SmallTalk**. Я у себя на компьютере включил в среду компилятор одного из лучших объектных языков - языка **Eiffel**.

Открытость среды не означает полной свободы. Все разработчики компиляторов при включении нового языка в среду разработки должны следовать определенным ограничениям. Главное ограничение, которое можно считать и главным достоинством, состоит в том, что все языки, включаемые в среду разработки *Visual Studio .Net*, должны использовать единый *каркас - Framework .Net*. Благодаря этому достигаются многие желательные свойства: легкость использования компонентов, разработанных на различных языках; возможность разработки нескольких частей одного приложения на разных языках; возможность бесшовной отладки такого приложения; возможность написать класс на одном языке, а его потомков - на других языках. Единый каркас приводит к сближению языков программирования, позволяя вместе с тем сохранять их индивидуальность и имеющиеся у них достоинства. Преодоление языкового барьера - одна из важнейших задач современного мира. Благодаря единому каркасу, *Visual Studio .Net* в определенной мере решает эту задачу в мире программистов.

Создание C#

Язык C# является наиболее известной новинкой в области создания языков программирования. В отличие от 60-х годов XX века - периода бурного языкотворчества - в нынешнее время языки создаются крайне редко. За последние 15 лет большое влияние на теорию и практику программирования оказали лишь два языка: Eiffel, лучший, по моему мнению, объектно-ориентированный язык, и Java, ставший популярным во многом благодаря технологии его использования в Интернете и появления такого понятия как виртуальная Java-машина. Чтобы новый язык получил признание, он должен действительно обладать принципиально новыми качествами. Языку C# повезло с родителями. Явившись на свет в недрах Microsoft, будучи наследником C++, он с первых своих шагов получил мощную поддержку. Однако этого явно недостаточно для настоящего признания достоинств языка. Попробуем разобраться, имеет ли он большое будущее?

Создателем языка является сотрудник Microsoft Андреас Хейлсберг. Он стал известным в мире программистов задолго до того, как пришел в Microsoft. Хейлсберг входил в число ведущих разработчиков одной из самых популярных сред разработки - Delphi. В Microsoft он участвовал в создании версии Java - J++, так что опыта в написании

языков и сред программирования ему не занимать. Как отмечал сам Андреас Хейлсберг, C# создавался как язык компонентного программирования, и в этом одно из главных достоинств языка, направленное на возможность повторного использования созданных компонентов. Из других объективных факторов отметим следующие:

- C# создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности - как FCL, так и CLR;
- C# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами;
- C# является мощным объектным языком с возможностями наследования и универсализации;
- C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования. Общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от C++ к C#;
- сохранив основные черты своего великого родителя, язык стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства C++ как указатели, адресация, разыменование, адресная арифметика;
- благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода;
- мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C#, позволяя легко строить Web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных;
- реализация, сочетающая построение надежного и эффективного кода, является немаловажным фактором, способствующим успеху C#.

Задание Решить задачи на C#

1. Вывести на экран сумму всех цифр введенного пользователем натурального числа.
2. Вывести на экран произведение квадратов всех цифр введенного пользователем натурального числа.
3. Вводится натуральное число. Напечатать число, полученное перестановкой его цифр в обратном порядке. Например, введено число 12345. Программа должна вывести число 54321.
4. Вводится натуральное число. Напечатать его цифровой корень. Цифровой корень образуется следующим образом:
5. Складываются все цифры числа
6. Если получено неоднозначное число, то его цифры снова складываются
7. И так до тех пор, пока не получим одну цифру.
8. Например, ввели число 12345. Складываем его цифры и получаем 15. Снова складываем. Цифровой корень= 6.
9. Напечатать таблицу из двух колонок. В 1-й колонке – натуральные числа в промежутке от 100 до 120, во второй – сумма цифр этих чисел.

Практическая работа № 18- 23

Тема - Программирование на языке Action Script

Цель работы – Привить студентам навыки работы в среде Flash8 и программирования на языке Action Script

Краткие теоретические сведения

Язык ActionScript, который используется в программе Macromedia Flash MX, позволяет Вам создавать клипы, которые действуют точно так, как Вам нужно. Вам не нужно знать все элементы ActionScript для того, что бы начать писать свои скрипты. Если цель Вам ясна, то Вы можете начать строить скрипты, используя простейшие операторы. Затем, по ходу работы, Вы можете внедрять новые изученные элементы языка, что бы выполнять более сложные задачи.

ActionScript позволяет создавать сценарий для клипа, кнопки или кадра. Каждый такой сценарий (то есть фактически программа на языке ActionScript) жестко связан с соответствующим элементом фильма. При публикации фильма текст сценария, как и другие элементы фильма, экспортируется в SWF-файл. Тем не менее, при желании вы можете сохранить его в отдельном файле с расширением .as (это обычный текстовый файл), чтобы впоследствии использовать в каком-либо другом фильме (или подарить другу).

Из сценария вы можете обратиться к любому другому объекту фильма (из числа кнопок, клипов и кадров) и даже к другому фильму или какому-либо внешнему сетевому ресурсу. Например, можно указать, что при достижении считывающей головкой кадра с номером 10 необходимо загрузить звуковой файл, расположенный на сайте www.sound.ru. Поэтому Flash должен знать, в каких «отношениях» состоят объекты, фигурирующие в сценарии. Некорректное обращение к объекту обязательно приведет к ошибке в сценарии. Скажем, запрошенный звуковой файл будет воспроизведен в 10-м кадре временной диаграммы фильма, вместо того, чтобы озвучить 10-й кадр конкретного клипа.

В связи с указанными выше обстоятельствами, перед началом работы с ActionScript полезно разобраться с двумя понятиями: объектная модель языка и контекст выполнения сценария.

Объектная модель ActionScript

Под объектной моделью понимается совокупность типов объектов, которые могут использоваться в сценарии, и отношения подчиненности между ними.

Напомним, что в языках программирования объект описывается набором атрибутов (свойств) и перечнем методов (процедур), которые могут быть применены, к этому объекту. Для каждого класса объектов определен свой набор атрибутов и методов. Например, для объекта «Кнопка» в качестве атрибутов могут использоваться метка, геометрические размеры, координаты, а в качестве метода — реакция на нажатие кнопки. Конкретный объект -- это экземпляр соответствующего класса. Два экземпляра одного и того же класса могут отличаться один от другого значениями атрибутов.

Для описания действий над объектами, а также для указания подчиненности объекта обычно применяется так называемая «точечная нотация». Например, чтобы указать, что к кнопке Button_l следует применить метод onPress(), используется конструкция

Button_1. onPress().

Если же требуется указать на принадлежность кнопки Button_1 клипу Clip_2, то запись может выглядеть так:

Clip_2. Button_1.

Объектная модель, применяемая в ActionScript, во многом аналогична объектным моделям других сценарных языков (например, JavaScript). Вместе с тем, существуют и определенные различия. Основное из них состоит в том, что в ActionScript иерархия объектов строится относительно Flash-плеера, а не относительно HTML-документа, отображаемого в окне Web-броузера.

Например, если HTML-документ содержит форму Form_l, в которой имеется кнопка Input_1, то в JavaScript «подчиненность» кнопки описывается следующим образом:

Documen t. Widow. Form_1. Input_1.

При этом, если обращение к кнопке выполняется из текущего документа, отображаемого в том же окне броузера, то первые два уровня иерархии явно не указываются, а подразумеваются. Соответственно, для обращения к кнопке может быть использована конструкция

Form_1. Input_1.

Аналогичные правила действуют и в ActionScript. Например, чтобы обратиться к клипу, непосредственно вставленному в кадр- основного фильма, можно записать: _root. Clip_l.

Ключевое слово _ root используется в качестве имени основного фильма и в данном случае может быть опущено. Если же клип является частью другого, «родительского», клипа, то для указания его подчиненности может использоваться ключевое слово _parent, например:

_parent. Clip_1.

Подробнее вопросы адресации объектов в ActionScript рассмотрены в следующем подразделе.

Теперь вернемся собственно к объектной модели ActionScript.

Как было сказано выше, корневым объектом, неявно присутствующим в любом сценарии, является Flash-плеер. На следующем уровне располагаются четыре класса объектов (рис. 9.2):

Movie (Объекты фильма);

Core (Объекты ядра);

Client/Server (Объекты клиент-серверной среды исполнения);

Authoring (Объекты среды разработки).

Объекты класса Movie позволяют представить в сценарии взаимоотношения между элементами фильма (то есть структуру фильма), а также управлять параметрами и поведением элементов фильма. К этому классу относятся, в частности, следующие объекты:

Button (Кнопка) — объект, представляющий в сценарии кнопку; для такого объекта может динамически изменяться, например, порядок установки фокуса ввода;

MovieClip (Клип) - объект, представляющий в сценарии клип; для него может динамически изменяться, например, число вложенных клипов;

Mouse (Мышь) — объект, представляющий в сценарии указатель мыши; он позволяет управлять видимостью и стилем указателя;

Key (Клавиатура) — объект, представляющий в сценарии клавиши, используемые для управления фильмом;

Color (Цвет) — объект, представляющий в сценарии палитру клипа и, соответственно, позволяющий изменять эту палитру;

Sound (Звук) - объект, представляющий -в сценарии звуковой символ, связанный с кнопкой или клипом;

Stage (Стол) — объект, предназначенный для управления некоторыми параметрами стола (в частности, масштабом изображения);

TextField (Текстовое поле) - объект, представляющий в сценарии динамическое текстовое поле или поле редактирования;

TextFormat (Формат текста) - объект, позволяющий управлять из сценария параметрами шрифта текстового поля.

Между объектами классов Button и MovieClip в ActionScript весьма сложно определить отношения подчиненности: они могут изменяться от одного фильма к другому и даже от одной сцены к другой. Например, і одной сцене кнопка может входить в состав клипа, а в другой — клиг может использоваться для «оживления» изображения кнопки, остальные же объекты класса Movie (за исключением, пожалуй, TextField) можно считать подчиненными по отношению к объектам Button и MovieClip, поскольку могут входить в их состав, но не наоборот.

Объекты класса Core предназначены для работы с объектами фильма и носят вспомогательную роль. Их следует использовать в том случае, если стандартных методов оказывается недостаточно для управления (изменения свойств) объектами фильма. В класс Core входят, в частности, следующие объекты:

Math (Математика) — объект, используемый в сценарии для работы с числовыми величинами; в отличие от всех других классов языка ActionScript, объект Math является и единственным экземпляром этого класса; то есть методы данного класса могут применяться непосредственно к объекту Math; например, чтобы получить значение косинуса с помощью метода cos, можно записать: Math.cos(3)\

Number (Число) - объект, предусмотренный для выполнения некоторых специальных операций с числовыми величинами; например, с его помощью можно определить наибольшее число в некотором диапазоне;

String (Строка) — объект, используемый в сценарии для работы со строками;

Date (Дата) - объект, используемый в сценарии для работы с календарными датами и временем.

Объекты класса Client/Server предназначены для работы с документами (Web-страницами), написанными на языке XML. Некоторые из этих объектов обеспечивают загрузку, обработку и пересылку XML-документов, другие - обслуживают соединения через сокеты. Тем самым ActionScript реализует поддержку серверной обработки XML-документов, содержащих Flash-фильмы.

Объект класса Authoring предназначен для управления параметрами среды разработки Flash-фильмов. Для него предусмотрены два основных метода: install и uninstall.

Контекст выполнения сценария

Контекст выполнения сценария определяет доступность и относительные адреса объектов и других программных величин (переменных), используемых в сценарии.

Необходимость учитывать контекст появляется в том случае, если в сценарии используется несколько объектов или переменных с одинаковыми именами. Например, если вы создадите переменную framePS для хранения скорости воспроизведения основного фильма, а затем — переменную с таким же именем (чего не бывает) для конкретного клипа, то рано или поздно сценарий сработает не так, как предполагалось.

В связи с эти необходимо учитывать следующую особенность ActionScript. Область видимости переменной в нем определяется временной диаграммой, активной в настоящий момент. Например, если в 10-м кадре основного фильма имеется клип, с которым связан сценарий, то все переменные, имеющиеся в этом сценарии, будут влиять на поведение временной диаграммы этого клипа, а не основного фильма. При условии, что в обращении к переменным отсутствует ссылка на временную диаграмму более высокого уровня (как может выглядеть такая ссылка, было сказано в предыдущем разделе).

С каждой временной диаграммой связан специальный параметр — уровень диаграммы (он обозначается с помощью ключевого слова _ level), который определяет ее положение относительно других временных диаграмм, загруженных в Flash-плеер. По умолчанию временная диаграмма основного фильма имеет нулевой уровень. Каждой вызываемой из нее временной диаграмме присваивается уровень, на единицу больший (рис. 9.3).

Таким образом, область действия переменных в Flash-фильме распространяется сверху вниз: имя переменной, созданной в сценарии временной диаграммы основного фильма, «видно» в сценарии следующего уровня, если в нем нет своей переменной с таким же именем.

Кроме того, внутри программного блока сценария, ограниченного фигурными скобками, могут объявляться локальные переменные, которые «видны» только в пределах этого блока.

Для управления областью видимости переменных в ActionScript используется понятие «путь назначения» — target path. Путь назначения позволяет указать принадлежность объекта или переменной конкретной временной диаграмме, загруженной в Flash-плеер.

Чтобы правильно записать путь назначения, необходимо учитывать объектную модель ActionScript и взаимное положение временных диаграмм. Например, если вы хотите указать, что требуется перейти к 10-му кадру основного фильма и воспроизвести его, вы можете записать такую конструкцию:

```
Jevel0.gotoAndPlay(l 0).
```

Возможны два варианта указания пути назначения: абсолютный и относительный.

Абсолютный путь вычисляется, как правило, на основе уровня временной диаграммы (как в приведенном выше примере). Указав в качестве отправной точки уровень диаграммы, вы можете быть уверены, что Flash-плеер правильно отыщет адресата, даже после того, как вы переместите фрагмент сценария, из которого выполнено обращение.

Относительный путь вычисляется с учетом положения той диаграммы, из которой выполняется обращение. Такой вариант более компактный, но менее надежный. Например, чтобы обратиться к кадру диаграммы, расположенной двумя уровнями выше текущей, можно записать такую конструкцию:

```
_parent._parent.muClip.
```

Однако после перемещения клипа, например, на более высокий уровень, относительная ссылка станет некорректной.

Итак, при создании сценариев на ActionScript возможно использование трех типов переменных, различающихся областью видимости:

глобальные (Global variables), которые доступны в сценарии любой временной диаграммы;

переменные временной диаграммы (Timeline variables), которые доступны из любой временной диаграммы при условии, что для обращения к ним используется путь назначения (target path);

локальные (Local variables), которые «видны» только в пределах того программного блока, в котором они объявлены.

Задание

1. Создать тестирующую программу
2. Создать говорящий алфавит
3. Создать программу перетаскивания объектов
4. Создать интерактивную карту Казахстана

Список литературы

Основная литература

1. Концепция информатизации образования РК;
2. Бобровский С. Технология Петагона на службе российских программистов. Программная инженерия.-СПб.:Питер,2003.-222 с.
3. Материалы сайта http://www.uran.donetsk.ua.
4. Материалы сайта http://www.lib.aswl.ru/books/methodology/programming.
5. Острейковский В.А. Информатика: Учеб. для вузов. – М.: Высш.шк., 1999.-511 с.
6. Симонович С.В. Информатика. Базовый курс. Учебник для вузов. –СПб: Издательство «Питер», 1999.-640 с.
7. Жужжалов В.Е. Основы интеграции парадигм программирования в курсе программирования.-М.: Образование и информатика, 2004. -128 с.
8. Фути К., Судзуки Н. Языки программирования и схемотехника СБИС. Пер. с япон.-М.:Мир,1988.-224 с., ил.
9. Ben-Ari M. Understanding Programming Language. John & Sons Ltd.-New York.1998.

Дополнительная литература

10. Открытое образование - стратегия XXI века для России/ Под ред. В.М. Филиппова и В.П. Тихомирова. М.: Изд-во. МЭСИ, , 2000. 356 с.

Tucker 1991 - Tucker A., et al. Computing Curricula 1991, Report of the ACM/IEEE-CS Joint Curriculum Task Force, ACM Press, N. Y. 1991.