Министерство науки и высшего образования РФ
ФГБОУ ВО «ДГТУ»

Методические указания к прохождению учебной практики по ПМ.01 «Разработка модулей программного обеспечения компьютерных систем» для студентов специальности 09.02.07 «Информационные системы и программирование»

Махачкала, 2022

Введение

Методические указания по выполнению практических работ по учебной практике по ПМ.01 «Разработка модулей программного обеспечения компьютерных систем» разработаны в соответствии с рабочей программой учебной практики и предназначены для приобретения необходимых практических навыков и закрепления теоретических знаний, полученных обучающимися при изучении учебных дисциплин, обобщения и систематизации знаний перед экзаменом.

Методические указания предназначены для обучающихся специальности 09.02.07 Информационные системы и программирование.

Рекомендации по оформлению практической работы

Задания выполняются обучающимися по шагам. Необходимо строго придерживаться порядка действий, описанного в практической работе.

Результаты выполнения практических заданий необходимо сохранять в своей папке на компьютере.

В случае пропуска занятий обучающийся осваивает материал самостоятельно в свободное от занятий время и сдает практическую работу с пояснениями о выполнении.

Критерии оценки практической работы

• наличие цели выполняемой работы, выполнение более половины основных заданий (удовлетворительно);

• наличие цели выполняемой работы, выполнение всех основных и более половины дополнительных заданий (хорошо);

• наличие цели выполняемой работы, выполнение всех основных и индивидуальных заданий (отлично).

ВВЕДЕНИЕ

Язык C# - новый язык программирования, разработанный компанией Microsoft под платформу .NET (читается и говорится как «Дот Нэт»). Другие языки программирования были созданы до появления платформы .NET. Язык C# специально создавался под эту платформу, и поэтому в нем отсутствуют проблемы совместимости с предыдущими версиями языка.

Для удобства изучения языка C# воспользуемся программой **SharpDevelop 5.1** - свободная среда разработки для C#, Visual Basic .NET, Boo, IronPython, IronRuby, F#, C++. Обычно используется как альтернатива Visual Studio .NET. Существует также форк на Mono/GTK+ — MonoDevelop.

SharpDevelop предоставляет интегрированный отладчик, который использует собственные библиотеки и взаимодействует с исполняющей средой .NET через COM Interop.

СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ C#

Для того чтобы понять структуру приложений на языке C#, сразу рассмотрим пример создания простого консольного приложения.

Запустите программу *Пуск -> Все программы -> SharpDevelop...*

Создайте консольное приложение, для этого выполните следующиедействия:
Выберите команду главного меню:

File -> New -> Solution...

В открывшемся диалоговом окне (рис. 1.1) выберите необходимые настройки для создаваемого проекта: шаблон: Console Application.

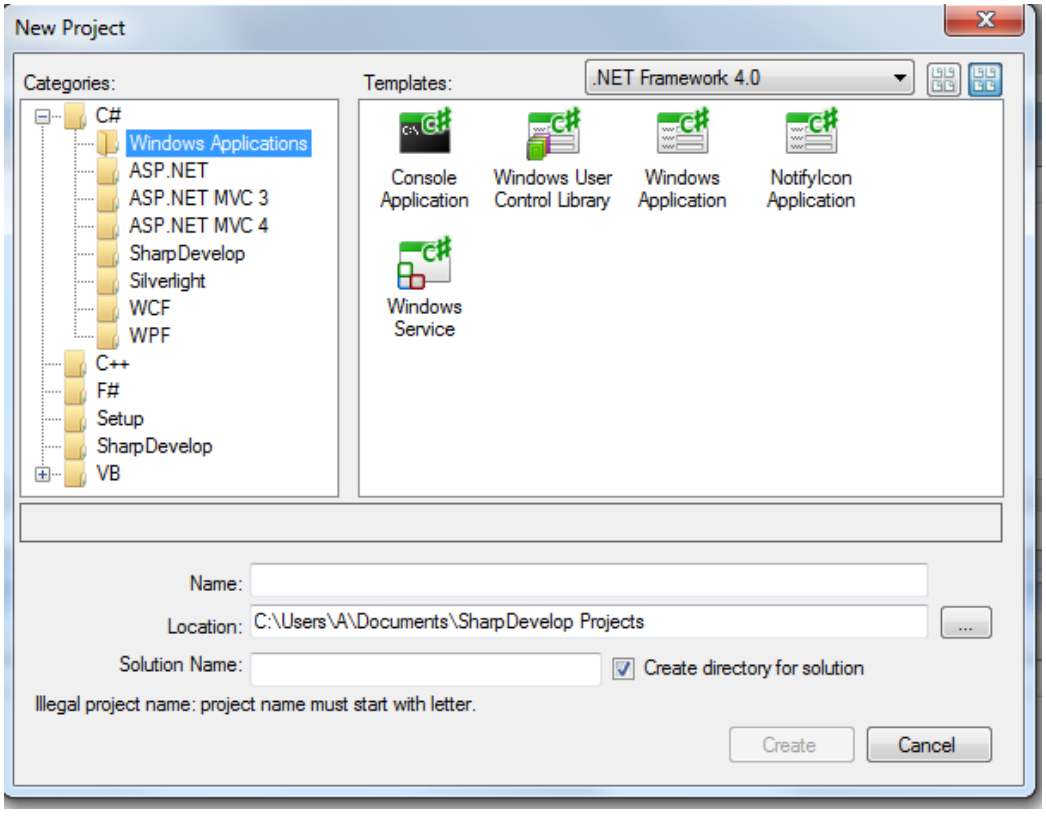


Рисунок 1.1 – Создание нового проекта консольного приложения.

В текстовом поле Name введите имя проекта (например, LR_One).
В текстовом поле Location выберите место сохранения нового проекта.
Установите флажок-переключатель «Create directory for solution».
Нажмите кнопку «OK».

2. После выполнения пункта 1 в среде разработки откроется новыйсозданный проект (рис. 1.2).

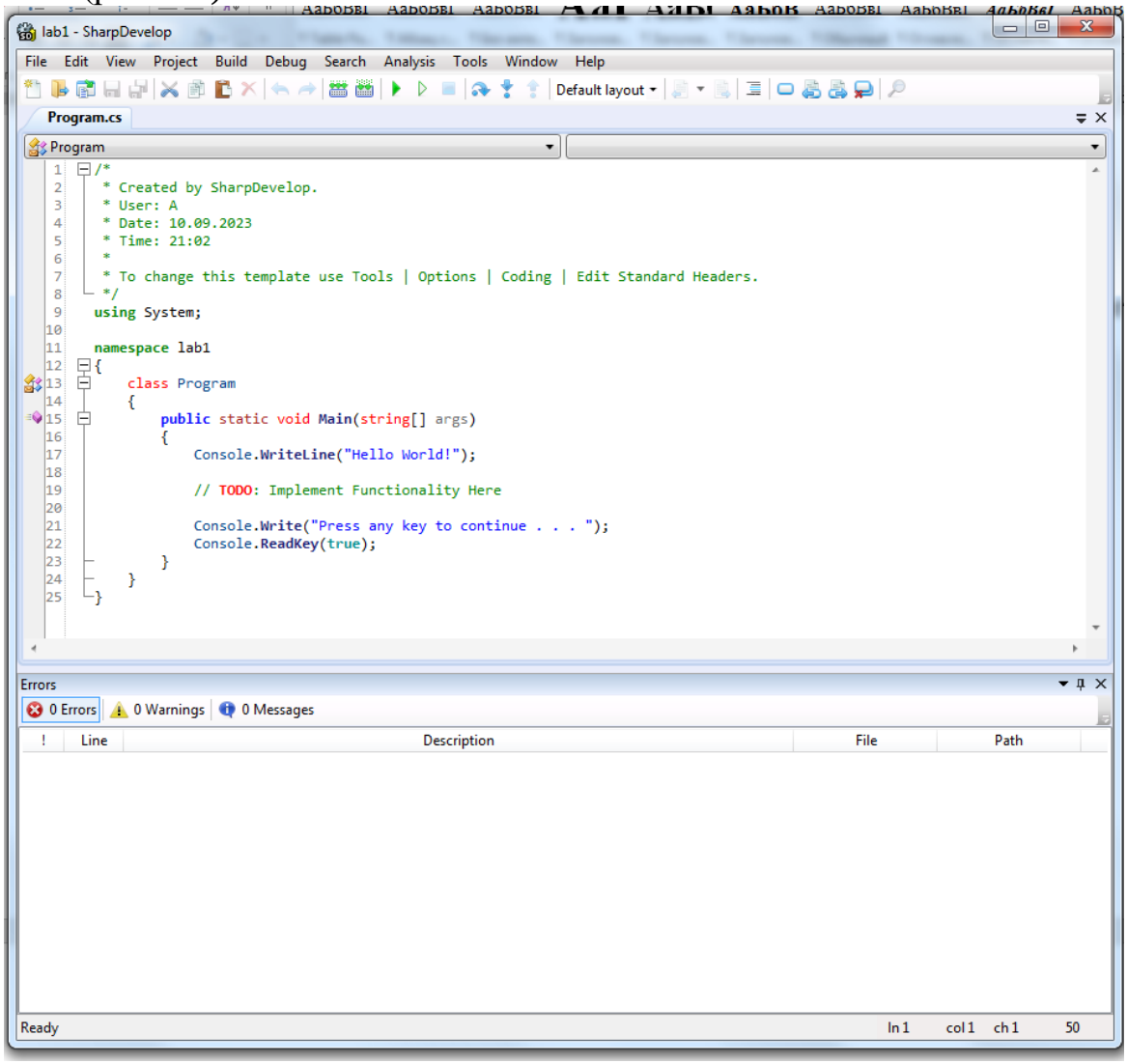


Рисунок 1.2 – Новый проект, загруженный в среду разработки:
открыт файл Program.cs в редакторе кода

3. На данном этапе необходимо ознакомиться со структурой исходного файла консольного приложения (рис. 1.2).

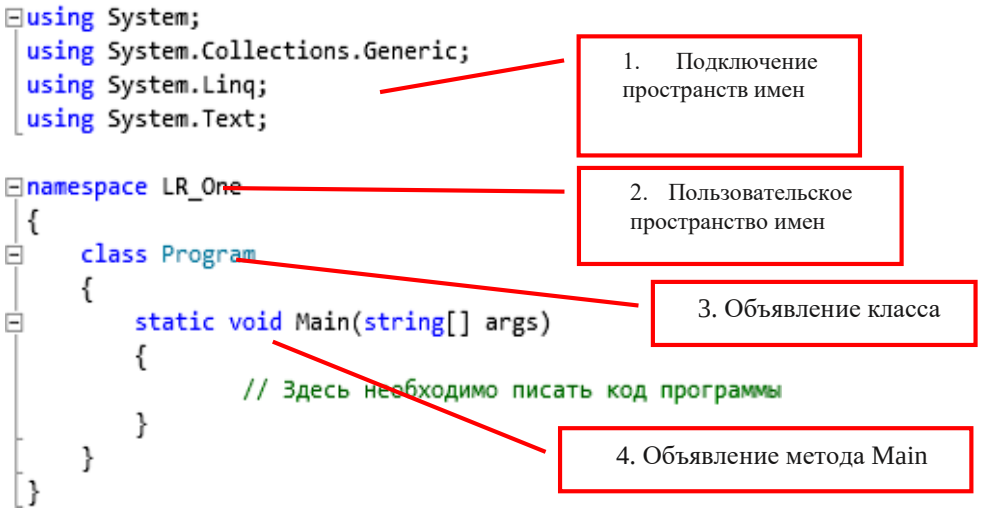


Рисунок 1.3 – Исходный файл консольного приложения.

На рис. 1.3в области 1 показано подключение пространств имен (библиотек, содержащих стандартные инструменты) с использованием зарезервированного слова using.

Программист сам создает свое пространство имен с именем LR_One (2), вкотором объявляется класс с именем Program.

В классе Program объявлен один метод – функция Main (параметры функции не рассматриваем).

Функция Main имеет особенное значение в программировании на языках C, C++ и C#.

Функцию Main называют «точкой входа», то есть началом выполнения программы. Далее мы рассмотрим приложения, содержащие множество функций. Операционная система знает с какой именно функции начать выполнение программы – с функции Main. Очевидно, что имя этой функции менять нельзя. Это должен быть статический метод класса (или структуры), возвращающий либо значение типа int, либо void. Хотя нередко модификатор public указывается явно, поскольку по определению этот метод должен быть вызван извне программы, на самом деле неважно, какой уровень доступа вы назначите методу точки входа. Он запустится, даже если вы пометите его как private.

Когда компилируется консольное или Windows-приложение C#, по умолчанию компилятор ищет в точности один метод Main () с описанной выше сигнатурой в любом классе и делает его точкой входа программы. Если существует более одного метода Main ( ), компилятор возвращает сообщение об ошибке.

Обратите внимание на вложенность конструкций на рис. 1.1: в пространство имен LR_One вложен класс Program, в класс Program вложена функция Main. В свою очередь, в функции Main содержатся инструкции на языке C#-код, который начнет выполняться при старте программы.

В C#, как и в других C-подобных языках, большинство операторов завершаются точкой с запятой (;) и могут продолжаться в нескольких строках без необходимости указания знака переноса. Операторы могут быть объединены в блоки с помощью

фигурных скобок ({ }). Однострочные комментарии начинаются с двойного слеша (//), а многострочные – со слеша со звездочкой (/*) и заканчиваются противоположной комбинацией (*/). В этих аспектах язык C# идентичен C++ и Java.

Следует также помнить о том, что язык C# чувствителен к регистру символов. Это значит, что переменные с именами myVar и MyVar являются разными.

Весь код программы необходимо писать внутри функции Main.

4. Для построения сборки (исполняемого exe-файла) выполните команду главного меню *Build → Build Solution* (или использовать горячую клавишу *F6*). После этого сборка создана, но приложение не будет запущено автоматически.

5. Для создания сборки и последующего запуска программы можно воспользоваться командой *Debug → Run* главного меню среды разработки или нажать кнопку панели инструментов  *Run* (*F5*) . Можно также использовать горячую клавишу *F5*.

6. Запустите приложение на выполнение одним из методов, указанных в выше. Окно консольного приложения появится и исчезнет. Это означает, что приложение выполнило все команды, написанные программистом, и завершило свою работу.

7. Для удержания окна на экране измените исходный файл в соответствии с рисунком 1.4. В функции Main добавлен вызов только одной команды Console.ReadKey () – эта функция останавливает выполнение программы и ждет, когда пользователь нажмет любую клавишу.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.ReadKey();
        }
    }
}
```

Рисунок 1.4 – Исходный файл консольного приложения дляпредотвращения закрытия окна консольного приложения.

8. Запустите измененное приложение, убедитесь, что окно удерживается на экране.

9. Добавьте несколько строк кода в исходный файл (рис. 1.5).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная работа №1");
            Console.WriteLine("");
            Console.WriteLine("Выполнил: Иванов Иван иванович");
            Console.WriteLine("Группа: ИСТБ-121");
            Console.WriteLine("Наименование ЛР: Структура консольного приложения");
            Console.WriteLine("");
            Console.WriteLine("для завершения работы программы нажмите любую клавишу...");

            Console.ReadKey();
        }
    }
}
```

Рисунок 1.5 – Исходный файл консольного приложения для вывода информации на экран

10. Внимательно изучите исходный код примера на рис. 1.5. Запустите приложение и убедитесь, что отсутствуют ошибки и информация выводится.
11. Выполните индивидуальное задание.

Индивидуальное задание.

Измените приложение, созданное в ходе выполнения данной лабораторной работы таким образом, чтобы программа выводила на экран следующую информацию (каждый студент должен использовать персональнуюинформацию о себе):

– Название и номер лабораторной работы;
– ФИО студента;
– Группа студента и шифр специальности;
– Дата рождения студента;
– Населенный пункт постоянного места жительства студента;
– Любимый предмет в школе;
– Краткое описание увлечений, хобби, интересов.

СИНТАКСИС C#

АЛФАВИТ

Алфавит языка C# включает в себя следующие символы таблицы кодов ASCII:

· строчные и прописные буквы латинского алфавита;
· цифры от 0 до 9;
· символ «_»;
· набор специальных символов: « {}, 1 [] + — %/\; ‘:? < > =! & # ~ *-;
· прочие символы.

Символы служат для построения лексем (проще говоря, слов). Существует пять видов лексем:

· идентификаторы;

- ключевые слова;
- знаки (символы) операций;
- литералы;
- разделители.

Комментарии

При написании программ бывает очень важно написать комментарии к программному коду, иначе впоследствии при обращении к какой-то его части вы можете забыть ход своих мыслей.

Комментарии в C# могут быть двух видов:

1) однострочный комментарий. Символ // означает начало комментария, который заканчивается в конце строки;

2) многострочный. Им обычно пользуются для того, чтобы прокомментировать какую-то часть кода или при необходимости написать комментарий из нескольких строк.

Типы данных в C#

Встроенные типы данных

Как уже говорилось в главе «Основные понятии», в платформе .NET существует *CTS — Common Type System* — стандартная система типов, которая является общей для всех языков программирования платформы.

Все типы в C#.NET делятся на *типы значений* и *ссылочные.*

Тип значения (value type) — тип, содержащий реальные данные. Этот тип не может быть равен *null.* Типы значений хранятся в области, известной как стек.

Пример типов значений: int x = 10, long y = 100;

Ссылочный тип (reference type) — тип, содержащий ссылку на значение. Ссылочные типы хранятся в области, которая называется управляемая кучей. Управляемая куча — это область памяти, в которой хранятся объекты, создаваемые с помощью ключевого слова *new*. Фактически это оперативная память вашего компьютера. При переполнении кучи вызывается сборка мусора. Пример ссылочного типа:

string str = “Привет, пользователь!”;

//создали переменную *str* ссылочного типа *string*, которая указывает на объект в памяти, содержащий строку «Привет, пользователь!»

Типы значений подразделяются следующим образом:

- целочисленные типы;
- типы с плавающей запятой;
- символы;
- логический тип;
- десятичный тип.

Ссылочные типы делятся:

- на объекты;
- строки.

Ниже представлена таблица встроенных типов.

Встроенные типы данных языка C#

CTS Тип	Имя в C#	Описание
System. Object	object	Класс, базовый для всех типов (CTS)
System. String	string	Строка
System. SByte	sbyte	8-разрядный байт (со знаком)
System. Byte	byte	8-разрядный байт (без знака)
System. S16	short	16-разрядное число (со знаком)
System. UM16	ushort	16-разрядное число (без знака)
System. Int32	int	32-разрядное число (со знаком)
System. UInt32	uint	32-разрядное число (без знака)
System. Int64	long	64-разрядное число (со знаком)
System. UInt64	ulong	64-разрядное число (без знака)
System.Char	char	16-разрядный символ (Unicode)
System. Single	float	32-разрядное число с плавающей точкой (стандарт IEEE)
System. Double	double	64-разрядное число с плавающей точкой (стандарт IEEE)
System. Boolean	bool	Булево значение (true/false)
System. Decimal	decimal	Данный 128-разрядный тип используется в основном, когда требуется крайне высокая точность (до 28 знака)

Преобразование типов

Преобразование объектов одного типа в другой может быть проведено явно или неявно. Неявные преобразования происходят автоматически, компилятор делает все за вас. Явные преобразования осуществляются, когда вы приводите значение к другому типу с использованием операторов приведения.

Пример неявного преобразования типов:

```
short a = 100;
int b = a;//неявное преобразование.
```

Если при компиляции неявного преобразования возможна потеря информации, то компилятор не станет выполнять такое преобразование.

Пример явного преобразования типов:

```
short a;
int b = 100;
a = (short) b;//явное преобразование произойдет успешно.
```

Переменные

Переменные — объекты определенного типа, расположенные в памяти, которые могут изменять свое значение. Переменные могут иметь значения, которыми они проинициализированы, или могут изменять свое значение в ходе выполнения программы.

Пример (рис. 7):

```
using System;
namespace ConsoleApplication3
```

```
{
class Program
{
static void Main ()
{
int a = 5;
Console.WriteLine («Начальное значение переменной равно\t»+a);
Console.ReadKey ();
a = 10;
Console.WriteLine («Теперь значение переменной равно\t»+a);
Console.ReadKey ();
}
}
}
```

Рис. 7. Результат работы программы

Константы

Константы — это объекты, значение которых не может быть изменено. Бывают случаи, когда есть необходимость сохранить значение переменной, чтобы в ходе выполнения программы его нельзя было изменить. Такими константами могут быть всем известное число пи или гравитационная постоянная.

Константы могут быть следующих видов:

- литеральные;
- символические;
- перечисления.

Пример:

$x = 2$;

Значение 2 — это литеральная константа. Значение 2 — это всегда 2. Мы не можем сделать так, чтобы число 2 представляло какое-либо другое значение.

Символические константы определяют имя для некоторого постоянного значения. Для объявления символической константы необходимо использовать следующий синтаксис:

const тип идентификатор = значение;

В виде кода это будет выглядеть так:

const double pi = 3.14;

Пример использования констант (рис. 8).

using System;

```
namespace ConsoleApplication3
{
class Program
{
static void Main ()
{
const double pi = 3.14;
const double g = 9.8;
Console.WriteLine (“Число пи равно\t”+pi);
Console.WriteLine («Гравитационная постоянная равна\t»+g);
Console.ReadKey ();
}
}
}
```

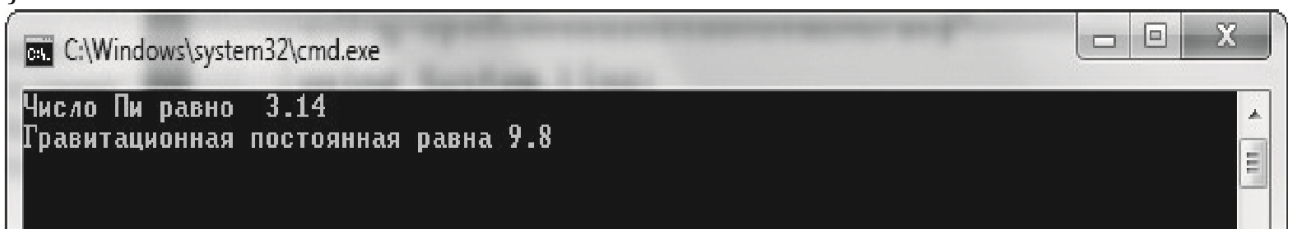


Рис. 8. Результат работы программы

Перечисления

Перечисления используются в качестве некоторой альтернативы обычным константам. Перечисления представляют собой список поименованных констант, объединенных логически, которыми очень удобно пользоваться. Это может быть список дней недели, месяцев, дней рождения родственников.

Так будет выглядеть список констант, содержащий дни недели:

```
enum Week
{
const string pn = “Понедельник”;
const string vt = “Вторник”;
const string sr = “Среда”;
const string cht = “Четверг”;
const string pt = “Пятница”;
const string sb = “Суббота”;
const string vs = “Воскресение”;
}
```

Итак, все дни недели сгруппированы и являются элементами одного перечисления типа Week.

Выражения

Выражение — некоторый код, определяющий значение.

Например:

x = 10;

Оператор «=» устанавливает значение для переменной *x* равным 10.

Также возможна следующая запись: *y* = *x* = 10;

В данном случае переменной *x* присваивается значение 10, а затем оператор присваивания устанавливает вторую переменную *y* с тем же значением.

Инструкции

Инструкция — это некоторое законченное выражение в программном коде. Любая программа на языке C# состоит из последовательности определенных инструкций, после каждой из которых обязательно должен стоять символ «;».

Пример записи инструкций:

```
string str;         //инструкция
a = 5;              //инструкция
double y = x;       //инструкция
```

Также возможна запись составных инструкций — это набор простых инструкций, заключенных в фигурные скобки:

```
{
string str;         //инструкция
a = 5;              //инструкция
double y = x;       //инструкция
}
```

Компилятор будет обрабатывать составную инструкцию как набор простых инструкций в том порядке, в котором они написаны.

Разделители

В языке C# пробелы, табуляция, символ перехода на новую строку являются разделителями. В коде можно поставить один за другим несколько разделителей, но компилятор обработает их как один и проигнорирует лишние разделители. Поэтому в коде программы возможны следующие абсолютно равнозначные записи:

x=1; или *x* = 1;

Из этого правила, как всегда бывает, есть исключение. Разделители в пределах строки (объекта типа string) будут обработаны как отдельные символы строки, т. е. запись

Consol.WriteLine («Хочу быть ПРОГРАММИСТОМ!!!»);

будет обработана компилятором именно в таком виде, в котором она представлена выше в символах « », со всеми пробелами.

Ветвление программы

Для большей гибкости программ и возможности выполнения различных сценариев в языке C# предусмотрены операторы перехода. Существует два вида перехода: условный и безусловный переход.

Безусловные переходы

Безусловный переход может быть осуществлен следующими способами:

- с помощью вызова функции;
- с помощью ключевых слов *goto*, *break*, *continue*, *return* или *throw*.

Пример безусловного перехода с использованием функции (рис. 9):

```
using System;
namespace ConsoleApplication4
{
class Program
{
static void Main ()
{
Console.WriteLine («Запустили программу. Работает метод Main…»);
Function ();
Console.WriteLine («А теперь опять работает метод Main.»);
Console.ReadKey ();
}
static void Function ()
{
Console.WriteLine («Теперь моя очередь работать. Работает функция Func-
tion:)”);
}
}
}
```

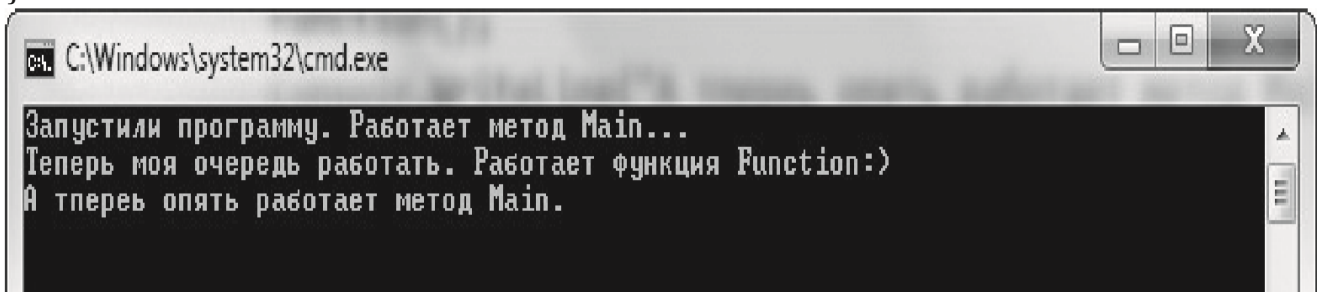


Рис. 9. Результат работы программы

Результат работы программы — в консоль выводится строка: «Запустили программу. Работает метод Main…» Далее программа осуществляет переход к функции Function, определенной ниже, и в консоль выводится строка: «Теперь моя очередь работать. Работает функция Function:)» Затем программа возвращается к выполнению последней инструкции и в консоль выводится еще одна строка: «А теперь опять работает метод Main.».

Пример безусловного перехода с помощью ключевого слова *goto* (рис. 10):

```
using System;
namespace ConsoleApplication5
{
class Program
{
static void Main ()
{
int a = 0;
Console.WriteLine («Выведем на экран последовательно числа от 0 до 10»);
```

```
Label:
Console.WriteLine (“a = “+a);
a = a + 1;
if (a<=10)
 goto Label;
Console.ReadKey ();
}
}
}
```

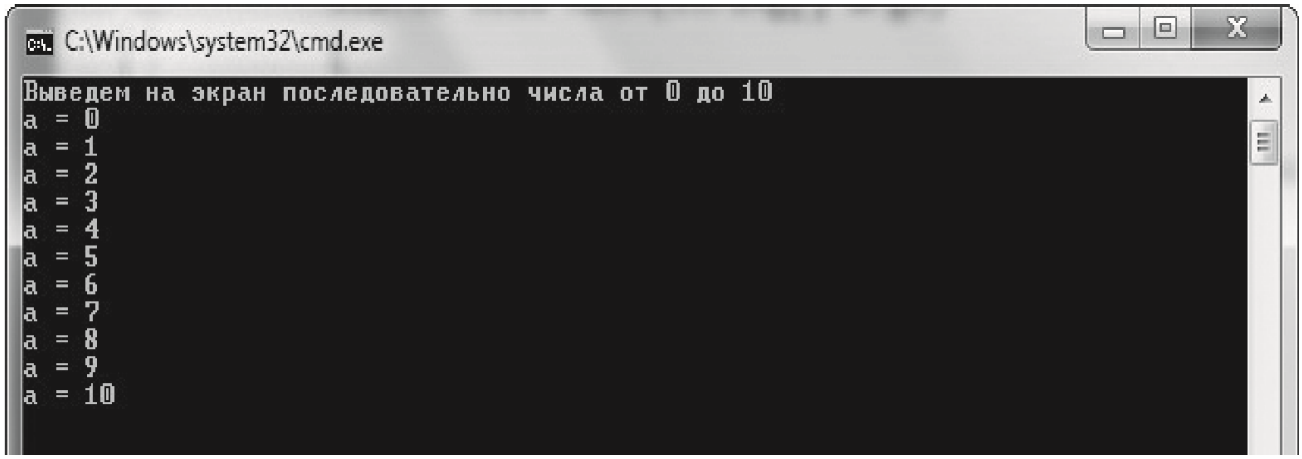


Рис. 10. Результат работы программы

Результат работы программы — в консоль выводится строка: «Выведем на экран последовательно числа от 0 до 10». В консоль выводится строка: «a = 0». Далее идет приращение переменной a на единицу: «a=a+1». Проверяется условие: «Значение «a» меньше 10?» Если меньше, то программа выполняет инструкцию *goto Label* и возвращается к метке *Label*.

Операция повторяется, пока a не достигнет значения 10.

Когда значение a достигло 10, программа переходит к инструкции *Console.ReadKey()*, которая ожидает от пользователя нажатия на любую клавишу для завершения работы программы.

Условные переходы

Условные переходы используются для ветвления программы относительно какого-либо условия, установленного разработчиком либо полученного в ходе каких-либо расчетов. Данные переходы реализуются с помощью ключевых слов *if, else, switch*.

Оператор выбора (условный оператор *if … else*)

Оператор *if* относится к операторам ветвления программы. Суть его работы состоит в следующем: анализируется условие, указанное в круглых скобках оператора *if*. Если условие верно (его значение *true*), то программа переходит к выполнению инструкций в блоке *if*. Если условие неверно (значение *false*), программа переходит к варианту инструкций, указанных в блоке оператора *else*. Структура оператора *if* имеет следующий вид:

```
if (условие)
{
```

```
инструкции
}
else
{
инструкции
}
```

При этом оператор *if* можно использовать без дополнения *else*. Также в блоке инструкций оператора *if* можно расположить еще один оператор *if* с каким-либо условием — получится вариант ветвления, в котором второй оператор *if* вложен в первый.

Рассмотрим программу, содержащую оператор условного перехода *if… else* (рис. 11).

```
using System;
namespace Primer1
{
class Program
{
static void Main ()
{
string Name;
int temp;
Console.WriteLine (“Привет! Как тебя зовут?”);
Name = Console.ReadLine ();
Console.WriteLine («Привет,\t»+Name+».\t\nКакая температура на улице?»);
temp = Convert.ToInt32 (Console.ReadLine ());
//определяем условие для оператора условного перехода
if (temp >= 20)
{
Console.WriteLine («Сегодня хорошая погода. Сейчас бы погулять»);
}
//если условие выше не сработало, то определяем альтернативный
//сценарий
else
{
Console.WriteLine («А я думал сегодня теплее. Останусь-ка я дома:)»);
}
Console.ReadKey ();
}
}
}
```

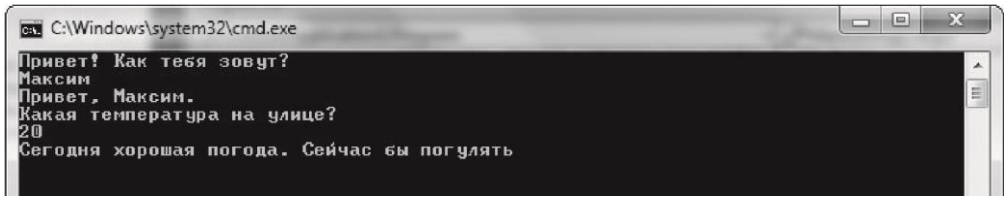


Рис. 11. Результат работы программы

Результат работы программы будет выглядеть следующим образом.

В консоль выводится вопрос: «Привет! Как тебя зовут?» Пользователь вводит свое имя: «Макс».

В консоль выводится следующий вопрос: «Привет, Макс. Какая температура на улице?» Пользователь вводит число, означающее температуру.

Далее сработает оператор условного перехода. Если температура больше 20 градусов, то выводится следующая строка: «Сегодня хорошая погода. Сейчас бы погулять».

Если температура меньше 20 градусов, выводится следующая строка: «А я думал сегодня теплее. Останусь-ка я дома:)».

Пример программы, вычисляющей площадь прямоугольника (рис. 12):

```
using System;
namespace ConsoleApplication6
{
class Program
{
static void Main ()
{
int a, b, s;
Console.WriteLine («Введите сторону «a» прямоугольника»);
a = Convert.ToInt32 (Console.ReadLine ());
Console.WriteLine («Введите сторону «b» прямоугольника»);
b = Convert.ToInt32 (Console.ReadLine ());
if (a > 0 && b > 0)
{
 s = a * b;
Console.WriteLine («Площадь прямоугольника равна\t» + s);
}
else
Console.WriteLine («Значения сторон должны быть больше 0»);
Console.ReadKey ();
}
}
}
```

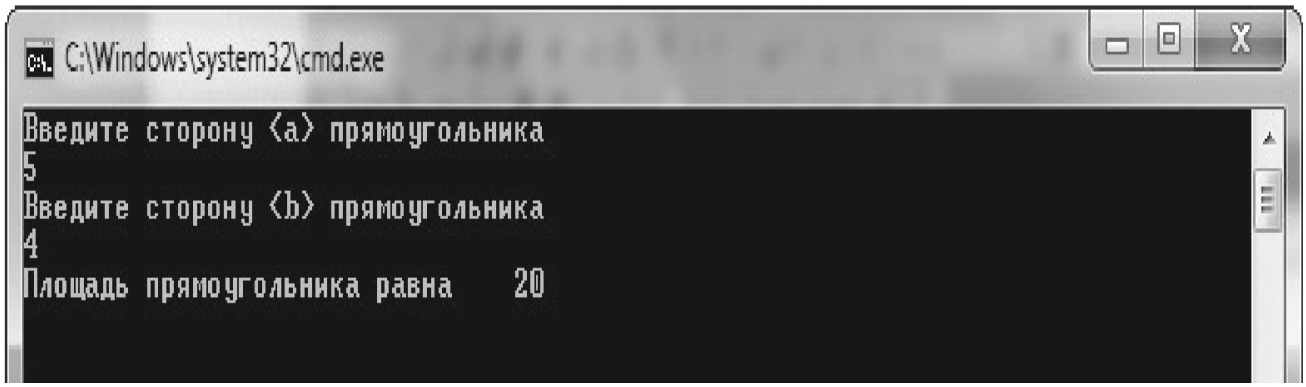


Рис. 12. Результат работы программы

Результат работы программы будет выглядеть следующим образом.

В консоль выводится строка: «Введите сторону <a> прямоугольника». Пользователь вводит значение переменной <a>.

В консоль выводится строка: «Введите сторону <b> прямоугольника». Пользователь вводит значение переменной <b>.

Далее проверяется условие: «<a> и <b> больше нуля?» Если да, то производится расчет площади и в консоль выводится строка: «Площадь прямоугольника равна …» Если значение a или b меньше нуля, в консоль выводится строка: «Значения сторон должны быть больше 0».

Программа ожидает от пользователя нажатия на любую клавишу для завершения работы.

Оператор ветвления *switch*

Оператор ветвления *switch* является альтернативой оператору *if … else*. Его обычно использую в случаях, когда имеется более сложный набор условий, состоящий из нескольких вариантов. Суть его работы состоит в следующем: программа ищет значение, которое соответствует переменной для сравнения, и далее выполняет указанные инструкции. Структура оператора выглядит так:

```
switch (выражение)
{
  case значение1:
    блок1;
    break;
  case значение2:
    блок2;
    break;
    ...
  case значениеN:
    блокN;
    break;
  default:
    блокN+1;
    break;
}
```

Выражение сравнивается последовательно со значениями. Если выражение равно значению – выполняется соответственный блок кода и при достижении ключевого слова break оператор switch заканчивает работу. Если выражение не будет соответствовать ни одному значению, тогда выполнится блок после default.

Рассмотрим пример использования оператора *switch* (рис. 13):

```
using System;
namespace ConsoleApplication8
{
class Program
{
static void Main ()
{
int a;
//предлагаем пользователю выбрать число из списка
Console.WriteLine («Выберите число:\n 100\n 200\n 300»);
//сохраняем набранное пользователем число в переменную «a»
a = Convert.ToInt32 (Console.ReadLine ());
//далее идет выбор варианта действий в зависимости от выбора пользователя
switch (a)
//если пользователь набрал 100
case 100:
Console.WriteLine («Вы выбрали число 100»);
break;
//если пользователь набрал 200
case 200:
Console.WriteLine («Вы выбрали число 200»);
break;
//если пользователь набрал 300
case 300:
Console.WriteLine («Вы выбрали число 300»);
break;
//если пользователь набрал число не из списка
default:
Console.WriteLine («Выберите число из списка0»);
break;
}
//ждем от пользователя нажатия любой клавиши для завершения работы
//программы
Console.ReadKey ();
}
}
}
```

Рис. 13. Результат работы программы

Результат работы программы таков.

В консоль выводятся строки: «Выберите число: 100, 200, 300». Допустим, пользователь выбирает 100. В консоль выводится строка: «Вы выбрали число 100».

Если пользователь выберет другой вариант, в ответ он получит соответствующую строку («Вы выбрали 200» или «Вы выбрали 300»).

Если пользователи введет число не из предложенного списка, в консоль выводится строка: «Выберите число из списка».

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Оператор *if else*

С помощью условного оператора *if… else* выполнить следующее.

1) Вычислить дату следующего дня.

Например: введите сегодняшнюю дату (число, месяц, год): 08 06 2012.

Завтра: 09 06 2012;

2) проверить, является ли год високосным;

3) запросить у пользователя номер дня недели и вывести сообщение, является ли день рабочим или это суббота или воскресенье;

4) вычислить оптимальную для пользователя массу. Сравнить ее с реальной массой и вывести в консоль рекомендацию поправиться или похудеть на определенное количество килограммов. Расчет оптимальной массы производится по формуле: рост (см)–100.

Например:

Ваш рост (см) 175

Ваш вес (кг): 95

Вам необходимо похудеть на 15 кг;

5) вычислить стоимость разговора по телефону с учетом стоимости одной минуты 3 р. и 15 % скидки, которая предоставляется по выходным.

Например:

Телефонный разговор.

Количество минут разговора (целое количество минут): 5

День недели (1 — понедельник, ..., 7 — воскресенье): 3

Скидка не предоставляется.

Стоимость разговора: 15 р.;

6) написать программу, которая находит сумму двух данных чисел (если оба числа четные) или произведение (если хотя бы одно из чисел нечетное);

7) написать программу, которая переводит время, указанное в секундах, в минуты и секунды.

Например:

Укажите время в секундах: 380

380 с = 6 мин 20 с

Укажите время в секундах: 12

12 с = 0 мин 12 с;

9) написать программу решения квадратного уравнения (коэффициент при второй степени переменной считать отличным от нуля). В случае, когда дискриминант меньше нуля, вывести соответствующее сообщение;

10) написать программу сложения двух обыкновенных дробей (числители и знаменатели дробей — параметры ввода). Предусмотреть случай, когда знаменатель дроби равен нулю;

11) написать программу умножения двух обыкновенных дробей (числители и знаменатели дробей — параметры ввода). Предусмотреть случай, когда знаменатель дроби равен нулю;

12) написать программу, которая проверяет, является ли введенное пользователем число четным;

13) написать программу, которая переводит время, указанное в минутах, в часы и минуты.

Например:

Укажите время в минутах: 126

126 мин = 2 ч 6 мин

Укажите время в минутах: 15

15 мин = 0 ч 15 мин;

14)переведите расстояние (указанное в метрах) в километры и метры.

Например:

Укажите расстояние в метрах: 3640

3640 м = 3 км 640 м

Укажите расстояние в метрах: 70

70 м = 0 км 70 м;

15) запросить у пользователя номер текущего месяца. Ввести с клавиатуры в консоль соответствующее название времени года. Если введенное значение большее 12, в консоль вывести ошибку:

«Введите корректный номер (от 1 до 12)»;

16) написать программу, реализующую простейший тест. Программа должна вывести вопрос и три варианта ответа, один из которых правильный. Пользователь вводит номер варианта, после чего программа сообщает: «Вы ответили правильно» или «Вы ошиблись»;

17) сравнить два введенных с клавиатуры числа. Необходимо определить, какое число меньше. Если числа равны, вывести в консоль сообщение: «Числа равны»;

18) написать программу деления двух обыкновенных дробей (числители и знаменатели дробей — параметры ввода). Предусмотреть случай, когда знаменатель дроби равен нулю.

Оператор *switch*

1. Написать программу, которая запрашивает у пользователя число из диапазона [1..10], а затем выводит на экран все его делители.
2. Написать программу, реализующую простейший тест. Программа должна вывести вопрос и пять вариантов ответа, один из которых правильный. Пользователь вводит номер варианта, после чего программа сообщает: «Вы ответили правильно» или «Вы ошиблись».
3. Написать программу, которая по номеру дня недели определяет его название или сообщает: «Ошибка ввода данных».
4. Написать программу, которая запрашивает у пользователя число и сообщает, какой цвет кодируется этим числом в языке Си#.
5. Написать программу, которая запрашивает у пользователя номер TV-канала (не более 10) и выдает его название.
6. Написать программу, которая по номеру месяца определяет время года. Например:

Введите номер месяца: 1

Январь — зимний месяц.

7. Написать программу, которая сообщает, какую сумму денег можно заработать в игре «Кто хочет стать миллионером?» за *n* правильных ответов.

Например:

Введите количество правильных ответов: 12

За 12 правильных ответов Вы получаете 125000 рублей.

8. Написать программу вычисления стоимости междугородного звонка. Стоимость минуты разговора определить по своему усмотрению.

Например:

Выберите пункт приема звонка (Мурманск — 1, Сургут — 2, …,

Волгоград — 8): 2

Укажите длительность разговора (мин): 7

Стоимость разговора: 32 р.

9. Написать программу, которая по номеру месяца определяет квартал.
10. Написать программу, которая запрашивает у пользователя число из диапазона [1..10], а затем сообщает, простое оно или составное.
11. Написать программу, которая по номеру месяца определяет его название или сообщает: «Ошибка ввода данных».
12. Написать программу-справочник, которая запрашивает номер автобусного маршрута г. Махачкала и сообщает конечные остановки этого маршрута.
13. Написать программу, которая запрашивает у пользователя число произвольного месяца, а затем сообщает, какие праздники в году приходятся на это число.

Например:

Введите число: 1

1 января — Начало Нового года

1 апреля — День смеха
1 мая — День труда
1 июня — День защиты детей
1 сентября — День знаний
Введите число: 20
Это не праздничный день. Введите число: –40
Ошибка ввода данных.

14. Написать программу, определяющую номер заданной буквы в алфавите.
15. Написать программу, которая выводит на экран кубы первых 10 простых чисел.

Циклические операторы

Оператор цикла с предусловием *while*

Структура оператора с предусловием *while* имеет следующий вид:

```
while (условие)
{
инструкции
}
```

При использовании этого оператора цикл выполняется, пока условие *while ()* имеет значение *true*.

Пример программы с оператором цикла *while* (рис. 14):

```
using System;
namespace ConsoleApplication7
{
class Program
{
static void Main ()
{
int a=0;
//пока «a» меньше либо равно 10
while (a<=10)
{
//выводим значение «a» в консоль
Console.WriteLine (a);
//производим приращение переменной «a» на единицу, и цикл повторяется
a = a + 1;
}
Console.ReadKey ();
}
}
}
```

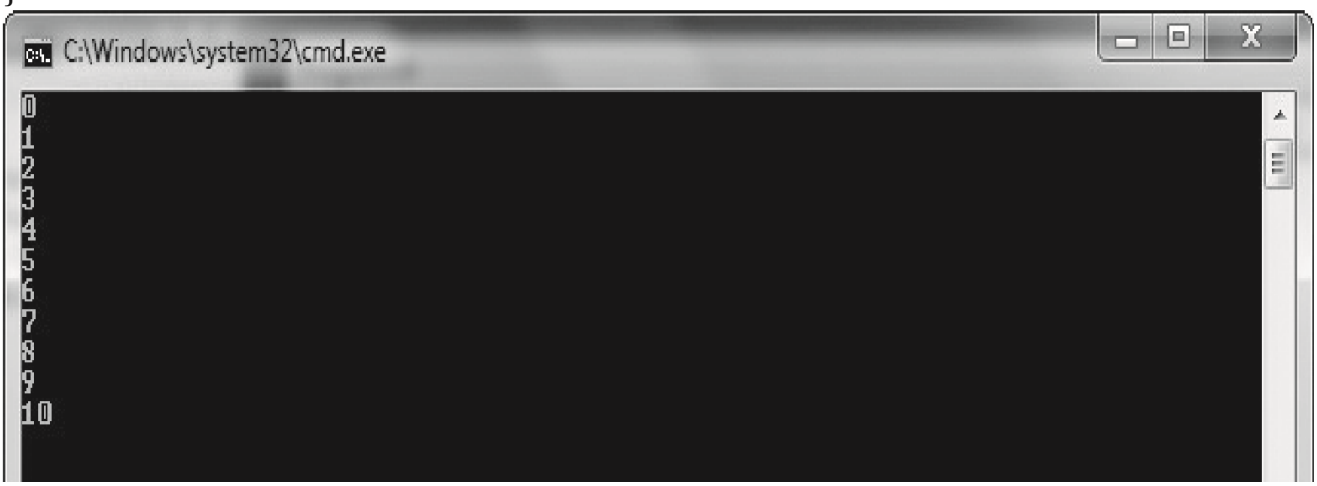


Рис. 14. Результат работы программы

Результат работы программы — в консоль выводится значение переменной a, равное 0. Затем происходит приращение значения a на единицу, и цикл повторяется (в консоль выводится следующее после приращения на единицу значение a, равное 1). Как только условие *while* становится равным *false* (т. е. значение a становится больше 10), цикл останавливается, и программа ждет от пользователя нажатия любой клавиши для завершения работы.

Оператор цикла с постусловием *do … while*

Оператор цикла *do…while* имеет следующую структуру:

```
do
{
Инструкции
}
while (условие);
```

Выражение выше можно прочитать следующим образом: «Выполнить инструкции. Проверить условие. Если условие выполняется, выполнить инструкции еще раз».

Разница между циклом *do…while* и циклом *while* состоит в том, что цикл *do…while* всегда выполняется хотя бы один раз до того, как произойдет проверка условия, что в некоторых алгоритмах принципиально.

Пример использования цикла *do …while* (рис. 15):

Возьмем код программы из предыдущего примера и немного его изменим.

```
using System;
namespace ConsoleApplication7
{
class Program
{
static void Main ()
{
//инициализация переменной «a»
int a = 0;
do
//прописываем инструкции, которые необходимо выполнить
{
Console.WriteLine (a);
 a = a + 1;
}
//прописываем условие, при выполнении которого цикл должен остановиться
while (a <= 10);
//ждем от пользователя нажатия на любую клавишу для завершения работы
//программы
Console.ReadKey ();
}
}
}
```

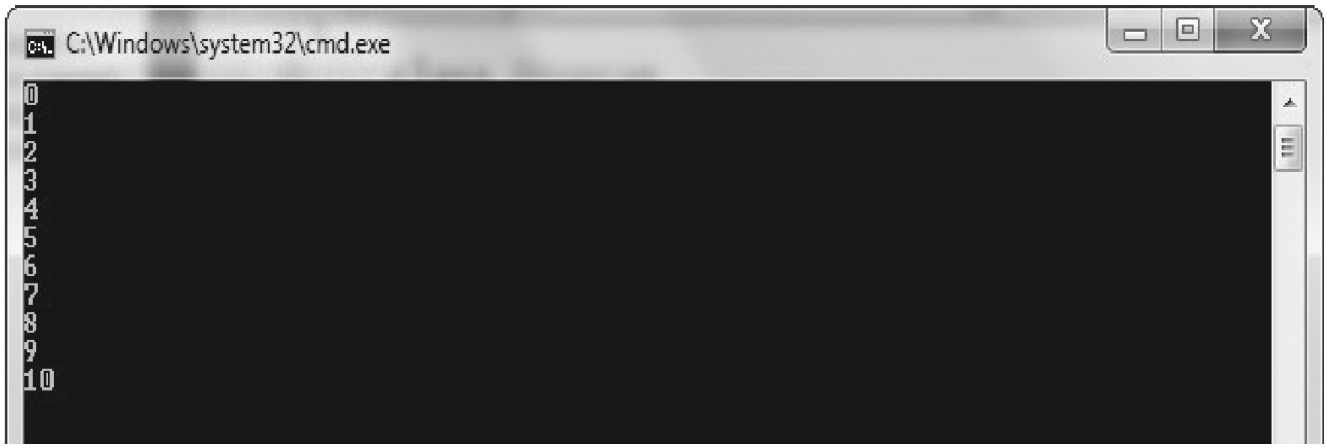


Рис. 15. Результат работы программы

В данном примере цикл *do…while* срабатывает столько же раз, сколько в предыдущем примере. Разница лишь в том, что если при инициализации переменной a мы бы указали значение больше 10 (вместо значения 0), например 12, то цикл *do…while* вывел бы в консоль число 12 и закончил бы работу. Цикл же *while* вообще бы не сработал, т. к. условие при первой же проверке оказалось бы равным *false*, и программа бы не смогла перейти к выполнению следующих инструкций.

Оператор цикла с параметром *for*

Если обратить внимание на операторы *while, do… while*, то можно заметить, что в их работе есть повторяющиеся операции: сначала происходит инициализация переменной, далее наращивание переменной, затем проверка переменной выполнения определенного условия. С помощью цикла *for* можно объединить эти операции в одной инструкции.

Оператор *for* имеет следующую структуру:

```
for (a = 0; a < 10; a ++)
{
инструкции;
}
```

В цикле *for* инициализируется переменная a, равная 0, затем идет проверка выполнения условия (a < 10). Если условие имеет значение *true*, то происходит наращивание переменной a на единицу (a++) и переход к выполнению инструкций в теле цикла. После выполнения всех инструкций в теле цикла программа снова возвращается к проверке условия. Как только значение условия становится *false*, цикл останавливается. Можно использовать любой шаг наращивания переменной. Для этого необходимо вместо выражения a++, написать, например, a+=2 — наращивание переменной на 2 единицы.

Пример использования цикла *for* (рис. 16):

Напишем программу, которая выводит в консоль числа от 0 до 10.

```
using System;
namespace ConsoleApplication8
{
```

```
class Program
{
static void Main ()
{
int a;
//в цикле for происходит инициализация переменной «a», проверка
//условия a < 11 и наращивание переменной «a» на единицу
for (a = 0; a < 11; a++)
{
//вывод на экран значения переменной «a»
Console.WriteLine («a = «+a);
}
//ждем от пользователя нажатия на любую клавишу для окончания работы
//программы
Console.ReadKey ();
}
}
}
```

Результат работы программы: вывод в консоль строки: «a = 0»; вывод в консоль строки: «a = 1»;

…

вывод в консоль строки: «a=10».

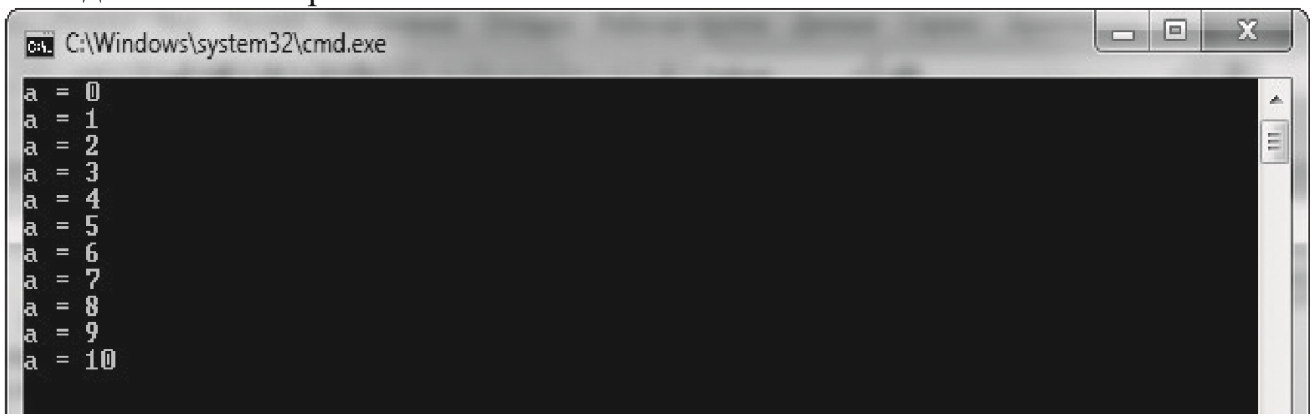


Рис. 16. Результат работы программы

Цикл *foreach*

Цикл *foreach* (англ. — для каждого) используется для обработки массивов, коллекций и других контейнеров, в которых хранится множество данных.

Оператор *foreach* имеет следующую структуру:

```
foreach («элемент» in «контейнер»)
{
инструкции;
}
```

Если записать эту структуру словами, то получится примерно следующее: *для каждого элемента в контейнере исполнить следующие инструкции* (которые находятся в теле цикла).

Контейнер — это некоторое хранилище данных. Элемент представляет собой элемент данных из контейнера (например, элемент массива).

Пример использования цикла *foreach* следующий.

Напишем программу, которая создает массив из 50 элементов, наполняет его случайными целыми числами и находит сумму элементов, а также максимальный и минимальный элементы.

Некоторые моменты в этом приложении еще незнакомы пользователю: использование массивов, создание объектов определенного класса и использование его методов. Об этих моментах будет сказано далее по тексту.

```
using System;
namespace ConsoleApplication3
{
class Program
{
static void Main ()
{
//Создаем массив на 50 целых чисел
int [] mas = new int [50];
//Создаем объект rnd класса Random, с помощью которого будем
//наполнять массив случайными числами
Random rnd = new Random ();
//Наполняем массив
for (int i = 0; i < 50; i++)
{
mas [i] = rnd.Next ();
}
//Сумма элементов
long S = 0;
//Инициализируем максимальный и минимальный элементы.
//Полагаем, что это первый элемент массива
int Min = mas [0], Max = mas [0];
//Цикл перебора элементов массива. Использование цикла foreach
foreach (int i in mas)
{
//Расчет суммы элементов. Идет перебор элементов массива.
//Каждый последующий прибавляется к предыдущему
S += i;
//Находим максимальный элемент массива
if (i > Max)
Max = i;
//Находим минимальный элемент массива
```

```
else if (i < Min)
Min = i;
}
//Выводим результат на экран
Console.WriteLine (“Сумма = {0}\n Минимальный элемент = {1}\n Максимальный элемент = {2}”, S, Min, Max);
//Ждем от пользователя нажатия на любую клавишу для завершения
//работы программы
Console.ReadKey ();
}
}
}
```

Результат работы программы показан на рис. 17.

В консоль выводится три строки:

«Сумма = »

«Минимальный элемент = »

«Максимальный элемент = »

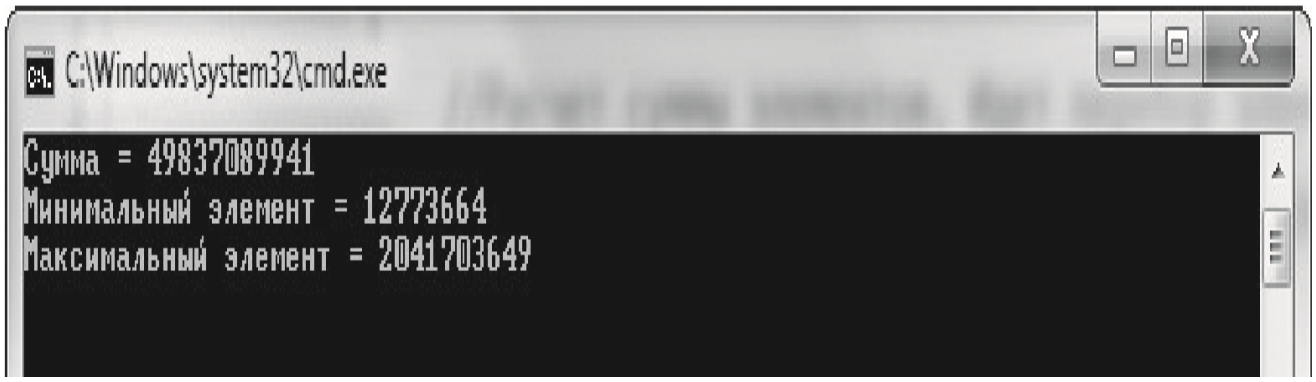


Рис. 17. Результат работы программы

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Оператор *for*

1. Вывести в консоль таблицу умножения на *n*.
   Например:

```
Введите число n: 7
7*1=7
7*2=14
7*3=21
7*4=28
7*5=35
7*6=42
7*7=49
7*8=56
7*9=63
7*10=70
```

2. Написать программу приближенного вычисления интеграла функции $f(x) = 5x^2 - x + 2$ методом трапеций.

3. Написать программу, находящую сумму n членов ряда:

$$\sum_{k=1}^{\infty} \frac{3k-1}{7k^2+9}.$$

4. Написать программу, которая преобразует введенное пользователем десятичное число в двоичное.

Например:

Введите целое десятичное число: 49

Данному десятичному числу соответствует двоичное 00110001

5. Написать программу, определяющую максимум в последовательности из n данных чисел.

6. Известно, что сумма ряда:

$$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots$$

при достаточно большом количестве слагаемых приближается к значению $\pi/4$.

Написать программу, вычисляющую значение числа π с заданной пользователем точностью.

7. Написать программу вычисления среднего арифметического заданных n чисел.

8. Написать программу приближенного вычисления интеграла функции $f(x) = 5x^2 - x + 2$ методом прямоугольников.

9. Написать программу, которая запрашивает последовательность из пяти чисел и после ввода каждого числа выводит среднее арифметическое введенной части последовательности.

10. Вывести в консоль 50 чисел, сгенерированных случайным образом в диапазоне от 1 до 200.

Операторы *while, do while*

1. Написать программу, которая запрашивает у пользователя число в диапазоне от 1 до 10. Затем компьютер генерирует числа в этом же диапазоне и выводит их на экран до тех пор, пока не угадает заданное пользователем число или не будет нажата клавиша ENTER.

2. Написать программу, вычисляющую значение выражения $\frac{(2n-1)!!}{5n!}$ при заданном значении n.

3. Написать программу, которая запрашивает у пользователя два целых числа: делимое и делитель — и выводит на экран значения частного и остатка.

4. Написать программу, вычисляющую наименьшее общее кратное двух данных целых чисел.

5. Написать программу, которая выводит пример на умножение столбиком с пропущенной цифрой и предлагает пользователю угадать эту цифру. Процедура повторяется до тех пор, пока не будет введена нужная цифра.

6. Написать программу, находящую сумму всех четных чисел, меньших заданного числа *N*.

7. Написать программу для вычисления значения выражения $2n^{10}(n^{5}-1)$ при заданном значении *n* (операцию возведения в степень реализовать через многократное умножение).

8. Написать программу, определяющую наибольший член ряда $\sum_{k=1}^{\infty}\frac{2}{(k+1)^2+3}$, не превосходящий заданного числа *E*.

9. Написать программу-игру «Угадай число». Суть игры состоит в следующем: компьютер генерирует число в диапазоне от 1 до 10 и предлагает пользователю угадать это число за 5 попыток. После ввода очередного числа программа должна выдавать сообщение: «Вы угадали» или «Вы не угадали».

10. Написать программу, определяющую минимум в последовательности вводимых с клавиатуры чисел (считать, что количество чисел заранее неизвестно).

11. Известно, что $N = 2^k$. Написать программу, определяющую по заданному *N* значение показателя степени *k*.

12. Написать программу, выводящую на экран таблицу степеней числа 3 от нулевой до *n*-й. Например:

Введите значение показателя степени: n= 3
3^0=1
3^1=3
3^2=9
3^3=27

13. Написать программу, определяющую наименьший член ряда $\sum_{k=1}^{\infty}k^2\cdot 2^k$, который больше заданного числа *E*.

14. Написать программу, вычисляющую сумму первых *n* членов
геометрической прогрессии с первым членом $b_1 = 1/2$ и знаменателем $q = 5$.

15. Написать программу, которая выводит на экран слово с пропущенной буквой и предлагает пользователю ввести эту букву. Процедура повторяется до тех пор, пока не будет введена нужная буква.

Массивы

Массивы — это набор данных, состоящий из некоторого фиксированного числа элементов, структурированных по типу.

Синтаксис массивов в C# несколько отличается от синтаксиса других C-подобных языков.

Приведем пример задания одномерного массива:

```
//задание массива с именем mas, состоящего из трех целых чисел.
int [] mas = new int [3];
```

Задать элементы массива можно следующим образом:

```
mas [0] = 5;
mas [1] = –7;
mas [2] = 89;
//нумерация элементов в массиве начинается с 0.
```

Следующая запись позволяет вывести элементы массива в консоль (на экран):

```
Console.WriteLine (mas[1].ToString ());//вывод второго элемента массива на экран.
```

Элементы массива можно также задать следующим способом:

```
int [] mas = new int {5, -7, 89};//задание элементов при объявлении массива
```

Задать двумерный массив можно следующим образом:

```
int [,] mas = new int [2, 2]; //задание двумерного массива (две строки и два столбца).
```

В данном массиве четыре элемента.

Нумерация первого элемента *mas* [0,0], нумерация последнего элемента *mas* [1,1].

Заполнить двумерный массив можно так:

```
mas [0, 0] = 5;
mas [0, 1] = –9 и т. д.; //задание элементов двумерного массива.
```

Перебор элементов одномерного массива происходит в цикле. Для двумерного массива придется использовать два цикла *for*, один вложен в другой.

Рассмотрим пример использования одномерного массива.

Программа создает одномерный массив, заполняет его случайными числами и находит сумму элементов массива.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication3
{
```

```
class Program
{
static void Main ()
{
//Массив на 50 целых чисел
int [] arr = new int [50];
//Случайное целое число
Random rnd = new Random ();
//Наполняем массив
for (int i = 0; i < 50; i++)
{
arr [i] = rnd.Next ();
}
//Сумма элементов long S = 0;
//Инициализируем максимальный и минимальный
//элементы. Полагаем, что это первый элемент массива = минимальному
//и максимальному
int Min = arr [0], Max = arr [0];
//Цикл перебора элементов массива
foreach (int i in arr)
{
//Расчет суммы элементов. Идет перебор элементов массива
//каждый последующий прибавляется к предыдущему
S += i;
}
//выводим в консоль сумму элементов массива
Console.WriteLine («Сумма элементов массива = {0}», S);
//ожидаем от пользователя нажатия любой клавиши для завершения работы
// программы
Console.ReadKey ();
}
}
}
```

Результат работы программы представлен на рис. 18.

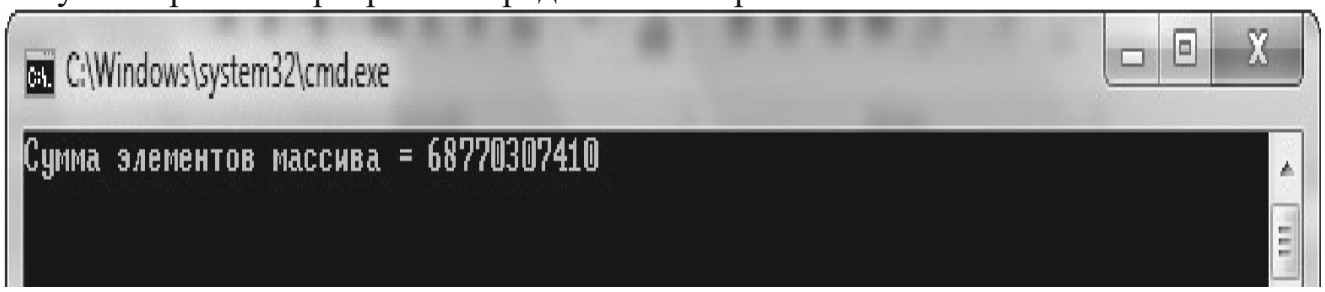


Рис. 18. Результат работы программы

Рассмотрим еще один пример с использованием массивов.

В массив записаны заработные платы работников некоторого предприятия. Определить количество работников, заработная плата которых ниже средней заработной платы по предприятию, и вывести на экран номера элементов массива, которые соответствуют таким работникам.

```
using System;
using System.Collections.Generic;
using System.Linq; using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main (string [] args)
//создаем массив, состоящий из заработных плат работников
int [] mas = {14553, 16756, 19522, 23456, 22858, 11256, 14552, 18673, 12400};
//инициализируем переменные для хранения суммы элементов массива
//и среднего значения
double sum = 0;
double av = 0;
//начинаем перебор элементов массива для расчета суммы его элементов
for (int i = 0; i < mas.Length; i++)
{
//находим сумму элементов массива
sum += mas [i];
}
//находим среднее значение элементов массива
av = sum/mas.Length;
//выводим в консоль значение средней заработной платы
Console.WriteLine («Средняя заработная плата по предприятию: {0}», av);
//начинаем перебор элементов массива для определения номеров
//элементов, значение которых ниже среднего значения
for (int i = 0; i < mas.Length; i++)
{
if (mas [i] < av)
{
//выводим в консоль номера элементов, значение которых ниже среднего
//значения
Console.WriteLine («Номер работника: {0}», i);
}
}
//ожидаем от пользователя нажатия любой клавиши для завершения работы
//программы
Console.ReadKey ();
}
}
```

}

Результат работы программы представлен на рис. 19.

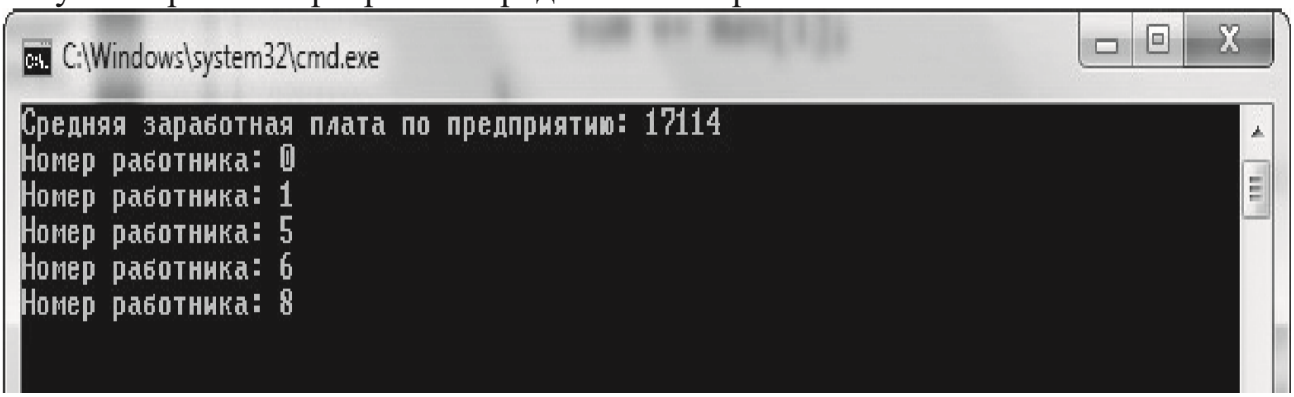


Рис. 19. Результат работы программы

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Одномерные массивы

1. Написать программу, которая определяет, состоят ли два массива из одинаковых элементов (без учета порядка следования).
2. Написать программу, определяющую в заданном массиве максимальное количество подряд идущих положительных элементов.
3. Вычислить среднее арифметическое отличных от нуля элементов массива и вывести их в консоль.
4. Написать программу, группирующую элементы массива следующим образом: в начале массива все отрицательные, затем нулевые и в конце все положительные элементы (с сохранением порядка следования в каждой группе).
5. Написать программу, располагающую элементы массива a [n] в следующем порядке:

$$a_1 a_{n-1} a_3 a_{n-3} a_5 \ldots a_{n-4} a_4 a_{n-2} a_2 a_n.$$

Написать программу, которая определяет, сколько элементов с равными значениями содержится в массиве.

Например:

Задайте количество элементов массива: 7

Введите элементы массива: 3 6 –1 3 7 3 7

В данном массиве два элемента равны 7, три элемента равны 3

6. Написать программу, которая каждый элемент массива, начиная со второго, заменяет суммой всех предшествующих ему элементов.

Например:

Введите элементы массива: 3 4 0 5 –7

После замены получен массив: 3 3 7 7 12

7. Написать программу, находящую минимальный по абсолютной величине элемент массива.
8. Написать программу, определяющую, сколько элементов массива по абсолютной величине больше заданного числа N.

9. Написать программу, группирующую элементы массива следующим образом: в начале массива все нечетные, а затем все четные элементы (с сохранением порядка следования в каждой группе).

10. Написать программу, которая упорядочивает элементы массива по убыванию.

11. Написать программу, определяющую, какой элемент массива повторяется максимальное количество раз. Если таких элементов несколько, то вывести на экран их все.

12. Написать программу, определяющую, сколько четных и нечетных элементов содержится в массиве.

Двухмерные массивы

1. Написать программу, находящую сумму элементов a_{ij} матрицы $A_{m\times n}$, для которых модуль разности индексов $|i–j|$ равен заданному числу k.

2. Написать программу, которая в данной матрице $A_{m\times n}$ ($m\geq 3$, $n\geq 3$) заменяет значение каждого элемента на сумму окружающих его восьми элементов, оставляя граничные элементы без изменений.

3. Написать программу, вычисляющую в матрице $B_{4\times 4}$ алгебраическое дополнение заданного элемента.

4. Написать программу, находящую обратную матрицу для заданной квадратной матрицы 3-го порядка.

5. Написать программу, которая все элементы квадратной матрицы, лежащие выше главной диагонали, заменяет единицами.

6. Написать программу, которая из элементов матрицы $C_{12\times 1}$ формирует матрицу $D_{3\times 4}$, заполняя ее по столбцам.

7. Написать программу, которая все элементы квадратной матрицы, лежащие ниже главной диагонали, заменяет нулями.

8. Написать программу, находящую сумму элементов квадратной матрицы, лежащих ниже обеих ее диагоналей.

9. Написать программу, находящую произведение двух заданных матриц (если умножение невыполнимо, вывести соответствующее сообщение).

10. Написать программу, которая в квадратной матрице 10-го порядка определяет номер столбца и номер строки, на пересечении которых находится наибольший элемент (заполнить матрицу случайными числами из диапазона [0..20] и вывести ее на экран).

Методы (функции) в C#. Оператор return

Метод (функция) являет собой небольшую подпрограмму. Если просто программа - это решение какой-то прикладной задачи, то функция – это тоже решение, только уже в рамках программы и, соответственно, она выполняет задачу «попроще». Функции позволяют уменьшить размер программы за счет того, что не нужно повторно писать какой-то фрагмент кода - мы просто вызываем сколько угодно и где нужно объявленную функцию.

Функции в Си-шарп также называют методами. Между этими двумя понятиями разница небольшая, и тут мы будем использовать термин функция.

До этого, мы весь код писали в функции main. Функция main является главной функцией приложения и точкой входа программы. Любая функция в Си-шарп может быть объявлена только в рамках класса, так как C# - полностью объектно-ориентированный язык программирования (ООП). Объявление пользовательской функции внутри другой функции (например main) недопустимо.

Объявление функции имеет следующую структуру:

```
[модификатор доступа] [тип возвращаемого значения] [имя функции]
([аргументы])
{
// тело функции
}
```

Модификатор доступа (области видимости) может быть public, private, protected, internal (назначение истекает из ООП). Пока везде использовать public.

Между модификатором и типом может стоять ключевое слово static, что означает, что функция будет статичной. Из статичной функции можно вызывать другие функции, если они тоже статичные. Главная функция main – всегда static, поэтому все функции будем объявлять также статичными.

Функция может возвращать значение или не возвращать. Если функция, например, возвращает целое число, то нужно указать тип int. Если функция не возвращает никакого значения, то для этого используется ключевое слово void. Функции, которые не возвращают значение, еще называют процедурами.

Называть функции стоит так, чтобы имя отображало суть функции. Используйте глаголы или словосочетания с глаголами. Примеры: GetAge(), Sort(), SetVisibility().

Аргументы – это те данные, которые необходимы для выполнения функции. Аргументы записываются в формате [тип] [идентификатор]. Если аргументов несколько, они отделяются запятой. Аргументы могут отсутствовать.

Первая строка функции, где указываются тип, имя, аргументы и т.д. называется **заголовком функции**.

Пример функции, которая не возвращает значение

Напишем простую функцию, которая будет заменять в массиве строк указанное имя на другое. Данная функция будет принимать три аргумента: массив строк, имя, которое необходимо заменить и новое имя. Так как функция не будет возвращать значение, указываем тип void перед именем функции.

```
  public static void ReplaceName(string[] names, string name, string newName)
{
  for (int i=0; i < names.Length; i++)
  {
   if (names[i] == name)
   names[i] = newName;
  }
}
```

Сама функция очень простая. Проходим в цикле по элементам и смотрим, равен ли элемент указанному имени. Если да, то заменяем его на новое имя.

Функция написана, и теперь используем ее:

```
class Program
{
  public static void ReplaceName(string[] names, string name, string newName)
  {
    for (int i=0; i < names.Length; i++)
    {
      if (names[i] == name)
        names[i] = newName;
      }
    }
  static void Main(string[] args)
  {
    string[] names = { "Sergey", "Maxim", "Andrey", "Oleg", "Andrey", "Ivan",
"Sergey" };
     ReplaceName(names, "Andrey", "Nikolay"); // вызов функции. Все строки
    //"Andrey" в массиве будут заменены на "Nikolay"
     ReplaceName(names, "Ivan", "Vladimir");
  }
}
```

После вызова функции два раза в этой программе, массив будет выглядеть так: "Sergey", "Maxim", " Nikolay ", "Oleg", " Nikolay ", " Vladimir ", "Sergey".

Пример функции, которая возвращает значения

Напишем функцию, которая будет находить максимальное число в массиве. Аргумент у этой функции будет один – массив целых чисел. Тип возвращаемого значения – целое число int.

```
public static int GetMax(int[] array)
{
  int max = array[0];
  for (int i = 1; i < array.Length; i++)
  {
    if (array[i] > max)
      max = array[i];
  }
  return max;
}
```

Логика функции проста. Создаем переменную max, в которую записываем первый элемент массива. Дальше в цикле сравниваем каждый элемент со значением в max, если элемент больше, чем max, то записываем в max этот элемент. В конце функции используем оператор return, чтобы вернуть наш результат.

Оператор return должен быть обязательно в функции, которая возвращает значение.

Используем нашу функцию:

```
class Program
{
  public static int GetMax(int[] array)
  {
    int max = array[0];
    for (int i = 1; i < array.Length; i++)
    {
      if (array[i] > max)
        max = array[i];
    }
    return max;
  }
  static void Main(string[] args)
  {
    int[] numbers = { 3, 32, 16, 27, 55, 43, 2, 34 };
    int max;
    max = GetMax(numbers); //вызов такой функции можно использовать при присваивании значения
    Console.WriteLine(GetMax(numbers)); // вызов функции также можно использовать как аргумент при вызове другой функции. WriteLine() – тоже функция.
    Console.ReadKey();
  }
}
```

Оператор return

Когда встречается этот оператор, происходит выход из функции и код ниже (если он есть) выполняться не будет (например, в функцию передан такой аргумент, при котором нет смысла выполнять функцию). Он похож на оператор break, который используется для выхода из циклов. Этот оператор также можно использовать и в функциях, которые не возвращают значение. Оператор return допустимо использовать несколько раз в функции, но этого делать не рекомендуется.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

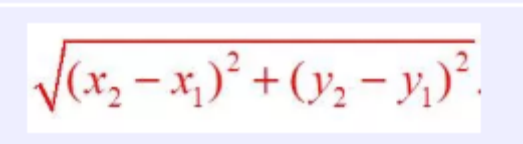
1) Напишите функцию, которая будет менять в массиве целых чисел все элементы, которые равны указанному значению (аргумент) на противоположное значение по знаку. Например, все элементы массива которые равны 5, будут меняться на -5.

2) Вводятся три числа — длины трех сторон треугольника. Создайте функцию Perimeter(), которая вычисляет и возвращает периметр треугольника по длинам трех его сторон.

3) Создайте метод GetPow(), который принимает два целочисленных аргумента — число и степень. Метод возвращает результат возведения числа в степень.

4) Создайте метод Distance(), который вычисляет расстояние между двумя точками на плоскости. Координаты точек вводятся (переменные x1,y1 — для первой точки, и x2,y2 — для второй).

5) Для подсчета расстояния между двумя точками на плоскости используйте формулу:



6) Создайте метод Minmax() который принимает два целочисленных аргумента по ссылке (ref) и меняет их значения таким образом, что первый становится максимальным, а второй — минимальным (меняет значения аргументов, если требуется). Создайте перегруженный метод Minmax для трех параметров.

7) Напишите функцию, которая будет менять в массиве целых чисел все элементы, которые равны указанному значению (аргумент) на противоположное значение по знаку. Например, все элементы массива, которые равны 5, будут меняться на -5.

8) Напишите функцию, которая будет находить минимальное число из трех.

9) Напишите функцию, которая будет находить максимальное число из четырех.