Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Высшая школа экономики и менеджмента

М. А. МЕДВЕДЕВ
А. Н. МЕДВЕДЕВ

ПРОГРАММИРОВАНИЕ НА СИ#

Учебное пособие

Министерство образования и науки Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

М. А. Медведев, А. Н. Медведев

ПРОГРАММИРОВАНИЕ НА СИ#

Учебное пособие

Рекомендовано методическим советом УрФУ
для студентов, обучающихся по направлениям подготовки
080500 «Бизнесинформатика»,
230700 «Прикладная информатика»

Екатеринбург
Издательство Уральского университета
2015

УДК 004.4СИ#(075.8)
ББК 32.973.22я73
М42

Рецензенты:
завсектором (и. о.) ОВТ ИММ УрО РАН канд. физ.-мат. наук Д. Г. Ермаков;
завлабораторией физики и экологии ИПЭ УрО РАН канд. физ.-мат. наук А. П. Сергеев
Научный редактор — канд. техн. наук, доц. А. В. Присяжный

Медведев, М. А.
М42 Программирование на СИ# : учеб. пособие / М. А. Медведев, А. Н. Медведев. — Екатеринбург : Изд-во Урал. ун-та, 2015. — 64 с.

ISBN 978-5-7996-1561-1

Учебное пособие содержит информацию об основах программирования на СИ#. В пособии рассматриваются такие темы, как структура программы и синтаксис языка СИ#, типы данных, ветвление программы, циклические операторы, массивы, введение в классы. По каждой теме представлены примеры программного кода и задания для самостоятельной работы. Учебное пособие предназначено для бакалавров.

Библиогр.: 13 назв. Рис. 20. Табл. 1.

УДК 004.4СИ#(075.8)
ББК 32.973.22я73

ISBN 978-5-7996-1561-1

Введение

Язык C# (читается и произносится как «Си-Шарп») — 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.

Разработка приложений на языке C# ведется в платформе Visual Studio.Net (читается и говорится как «Вижуал Студио Дот Нет»), куда помимо C# встроены такие языки программирования, как Visual Basic.NET и Visual C++.

NET Runtime — «Среда выполнения». В этой среде выполняется код, полученный после компиляции программы, написанной на C#. Эта среда выполнения построена не на ассемблере (код, который является родным для процессора), а на промежуточном коде. Поэтому для данной «Среды выполнения» возможно использование нескольких языков программирования. В теории программа, написанная для этой среды, может быть выполнена любой операционной системой, в которой NET Runtime установлена, и пока для этого существует только одна ОС — Windows.

Основные понятия .NET

Основными понятиями в .NET являются следующие.

Assembly (сборка) — основа приложений в .NET Framework. Сборка состоит из управляемых модулей. Фактически сборка — это набор каких-либо управляемых модулей, объединенных логически. Сборка может существовать в виде исполняемого приложения (расширение.exe) или, например, библиотеки (расширение .dll). Модули сборки исполняются в CLR — общеязыковой исполняющей среде.

Managed Code (управляемый код) — код, который выполняется в среде CLR.

CTS — Common Type System — стандартная система типов, которая является общей для всех языков программирования платформы. В этой системе прописано, как должен быть определен какой-либо тип (класс, интерфейс, структура, встроенный тип данных) для его правильного выполнения средой .NET.

Namespace (пространство имен) — это некоторая модель организации системы типов, созданная для логической группировки типов в единую группу. В .NET существует общая для всех языков библиотека базовых классов.

NET Runtime — набор базовых классов. Существует несколько тысяч классов. Классы относятся *не* к конкретному языку, а к самой среде выполнения NET Runtime. Таким образом мы получаем общий набор классов для всех языков программирования платформы .NET.

Структура программы на языке C#

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

Первым делом открываем Visual Studio 2010 и видим следующую окно (рис. 1).

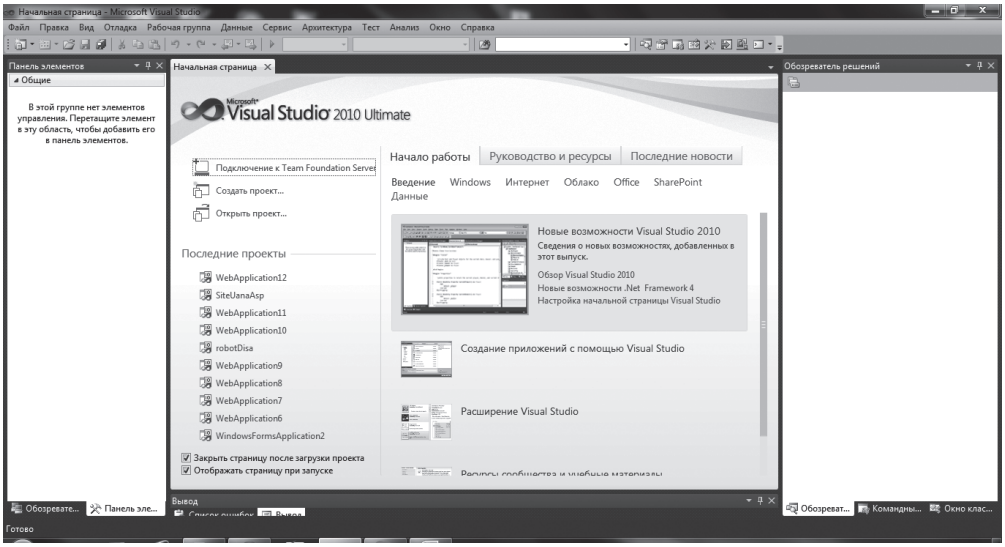

Рис. 1. Начальная страница MS Visual Studio 2010

Это — начальная страница, в которой отображена информация по последним проектам, меню для создания нового проекта или открытия уже созданного и некоторые справочные материалы (руководство и ресурсы, создание приложений с помощью Visual Studio 2010 и т. д.).

Для того чтобы начать писать программный код консольного приложения, следует создать новый проект (меню *Файл->Создать->Проект*) (рис. 2).

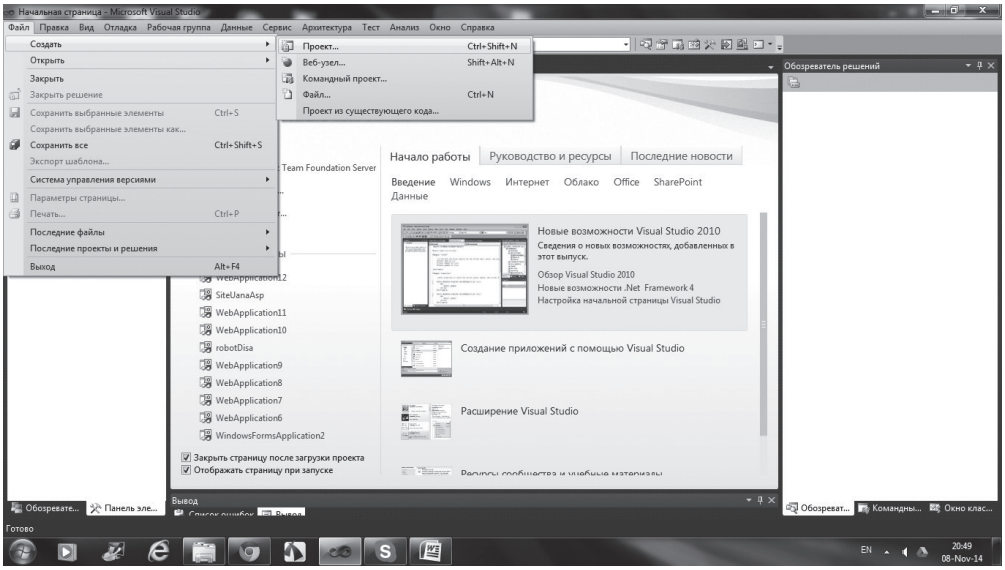

Рис. 2. Создание нового проекта в MS Visual Studio 2010

Далее мы должны выбрать из списка предложенных вариантов проектов «Консольное приложение» (рис. 3), придумать ему имя и нажать на кнопку ОК.

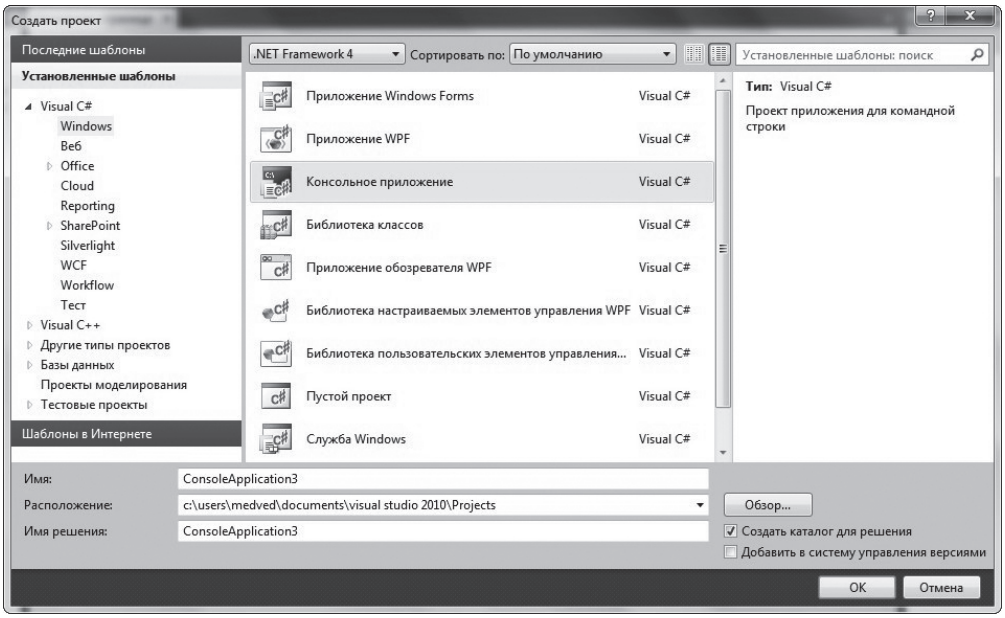

Рис. 3. Выбор варианта проекта «Консольное приложение»

После нажатия на кнопку ОК наш проект будет создан (рис. 4). Как мы видим, главное рабочее окно поделено на несколько зон, в которых отображена различная информация, связанная с проектом. В центре располагается собственно программный код приложения. Сейчас там уже есть несколько строк, которые Visual Studio 2010 сгенерировал автоматически. Дело в том, что после выбора варианта проекта среда генерирует шаблон для данного вида проекта, который содержит информацию о подключении библиотек классов, пространстве имен, основном методе Main (точка входа в программу). Эти строки мы можем дополнить своим кодом.

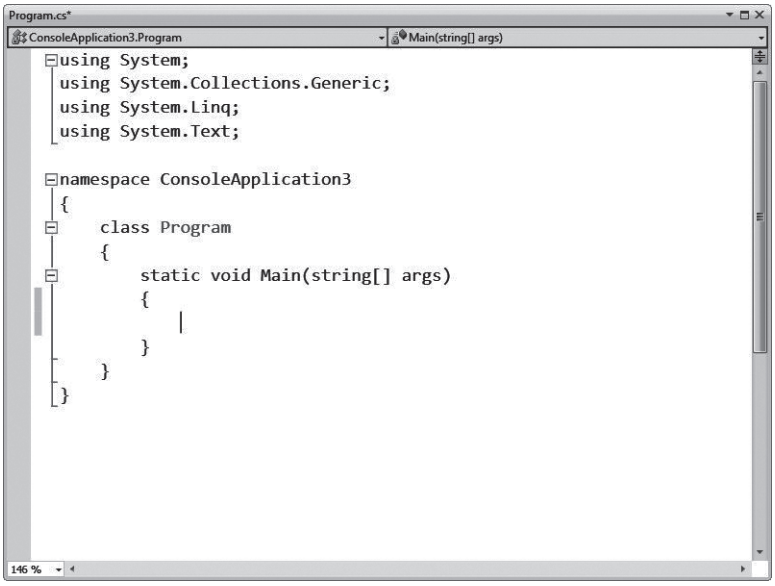

Рис. 4. Рабочее окно проекта «Консольное приложение» после его создания

Справа расположено окно обозревателя проектов и окно свойств. Обозреватель проектов содержит информацию о структуре созданного проекта (файлы, библиотеки, формы, подключенные модули). Окно свойств отображает свойства выделенного в данный момент объекта.

Снизу, по центру, расположено окно списка ошибок, в которое выводятся все текущие ошибки и предупреждения. Слева расположено окно панели инструментов. Для консольного приложения это окно будет пустым, а вот, например, для Windows-приложения оно будет содержать различные инструменты для создания формы приложения.

Теперь приступим к написанию нашего первого консольного приложения. Приложение будет запрашивать имя пользователя, а затем выводить на экран приветствие с ним. Строки, которые помечены жирным шрифтом внутри метода Main, мы написали сами. Остальные строки сгенерированы автоматически в качестве шаблона консольного приложения.

Исходный код программы:

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

namespace ConsoleApplication7
{
    class Program
    {
    static void Main (string [] args)
        {
        string str;
        Console.WriteLine (“Как тебя зовут?”);
        str = Console.ReadLine ();
        Console.WriteLine (“Привет, “+str);
        Console.ReadKey ();
        }
    }
}
```

Результат работы программы (рис. 5):
В консоль выводится строка: «Как тебя зовут?»
Пользователь вводит свое имя, например: «Максим».
В консоль выводится строка: «Привет, Максим».

Рис. 5. Результат работы программы

Разберем код программы более подробно.

Первые 4 строки в исходном коде с ключевым словом *using* подключают библиотеки классов, с методами и свойствами которых мы будем работать:

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

Далее объявляется пространство имен (что-то вроде собственной территории нашего проекта), в котором будет выполняться проект:

```
namespace ConsoleApplication7
{
```

//затем объявляется класс Programm, в котором находится точка выхода в программу — метод Main.

```
class Program
    {
    static void Main (string [] args)
        {
```

Далее идет код самого метода. Здесь прописываются инструкции нашей программы:

//объявление строковой переменной str для хранения имени пользователя.

```
string str;
```

//метод вывода текста в консоль (текст прописывается в фигурных скобках). Метод ***WriteLine*** **()** принадлежит классу *Console*, который, в свою очередь, находится в пространстве имен *System*. Теоретически нужно было бы написать ***System.Console.WriteLine ()***, но в первых 4-х строках мы произвели подключение пространства имен *System*, поэтому далее в коде указывать его нет необходимости.

```
Console.WriteLine («Как тебя зовут?»);
```

//присваиваем переменной *str* значение, которое ввел пользователь.

```
str = Console.ReadLine ();

//с помощью метода WriteLine выводим на экран строку «Привет, имя_пользователя»
Console.WriteLine («Привет «+str);

//далее программа ожидает от пользователя нажатия на любую клавишу для заверешения своей работы. Для этого используется метод ReadKey ().

Console.ReadKey ();
}
}
}//не забываем закрывать фигурные скобки
```

В следующих разделах будут более подробно рассмотрены основные возможности языка C#: синтаксис, типы данных, операторы ветвления программы, циклы, построение классов и создание приложений Windows Forms.

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.

Ниже приведена схема (рис. 6), в которой отражены ключевые слова языка C#, использовать которые в качестве идентификаторов не разрешается.

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.

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

Рис. 6. Ключевые слова C#

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

namespace ConsoleApplication3
{
class Program
{
static void Main (string [] args)
{
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;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication3
{
class Program
{
static void Main (string [] args)
{
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;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication4
{
class Program
{
static void Main (string [] args)
{
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 Function:)");
}
}
}
```

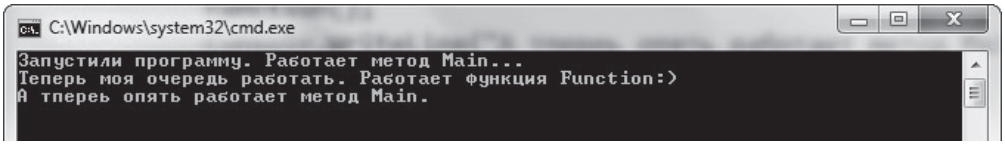

Рис. 9. Результат работы программы

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

namespace ConsoleApplication5
{
class Program
{
static void Main (string [] args)
```

```
{
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 ();
}
}
}
```

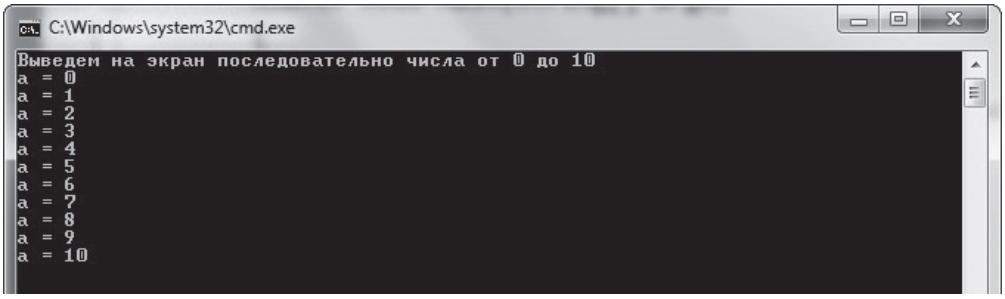

Рис. 10. Результат работы программы

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

namespace Primer1
```

```
{
class Program
{
static void Main (string [] args)
{
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 альтернатив-
ный//сценарий
else
{
Console.WriteLine («Хм… А я думал сегодня теплее. Останусь-ка я дома:)»);
}
Console.ReadKey ();
}
}
}
```

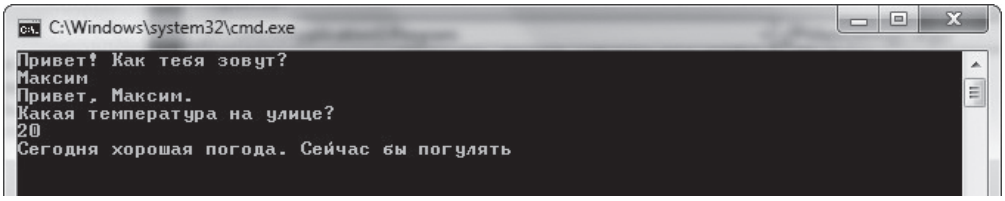

Рис. 11. Результат работы программы

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: «Хм… А я думал сегодня теплее. Останусь-ка я дома:)».

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

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

namespace ConsoleApplication6
{
class Program
{
static void Main (string [] args)
{
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 возможный вариант выражения:
инструкции;
break;//место выхода из case
```

```
case возможный вариант выражения:
инструкции;
break;//место выхода из case
…
}
```

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

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

namespace ConsoleApplication8
{
class Program
{
static void Main (string [] args)
{
int a;

//предлагаем ползователю 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 трамвайного маршрута г. Екатеринбург 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 знак зодиака пользователя. Например:

Укажите месяц рождения: 7
Укажите число рождения: 8
Вы по гороскопу Рак.

15. Написать программу, определяющую номер заданной буквы в алфавите.

16. 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;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication7
{
class Program
{
static void Main (string [] args)
{
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;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication7
{
class Program
{
static void Main (string [] args)
{
//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 ();
}
}
}
```

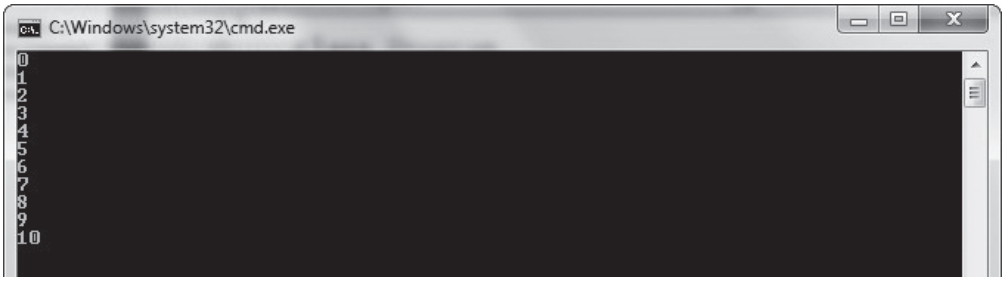

Рис. 15. Результат работы программы

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

namespace ConsoleApplication8
{
class Program
{
static void Main (string [] args)
```

```
{
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 оконочания 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 читателю: использование массивов, создание объектов определенного класса и использование его методов. Об этих моментах будет сказано в следующих главах.

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

namespace ConsoleApplication3
{
class Program
{
static void Main (string [] args)
{
```

```
//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}\nМинимальный элемент = {1}\nМаксимальный 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 $\frac{\pi}{4}$. 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 $b_1 = \frac{1}{2}$ 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:

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

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 (string [] args)
{
//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 элеменов 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.

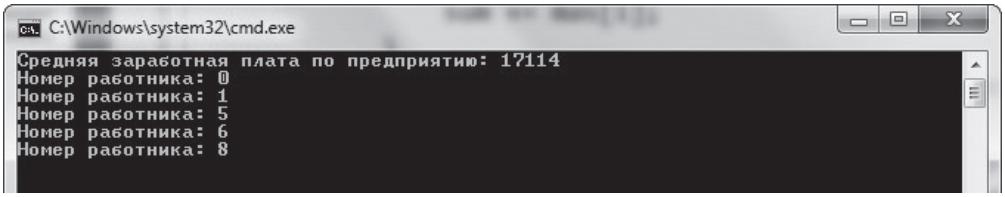

Рис. 19. Результат работы программы

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:

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

6. 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

7. 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

8. Написать программу, находящую минимальный по абсолютной величине элемент массива.

9. Написать программу, определяющую, сколько элементов массива по абсолютной величине больше заданного числа N.

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

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

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

13. 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, которая переставляет строки данной матрицы так, чтобы первые элементы этих строк образовывали возрастающую (неубывающую) последовательность.

Например:

Задайте количество строк матрицы: 5
Задайте количество столбцов матрицы: 3
Введите элементы матрицы:
6 0 1
–2 3 4
4 4 7
1 –2 8
4 0 8

Результат выполнения программы:

```
–2 3 4
1 –2 8
4 4 7
4 0 8
6 0 1
```

5. Написать программу, в которой элементам двухмерного массива присвоить значения соответственно приведенной схеме, после чего вывести массив на экран:

$$\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{matrix}$$

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

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

8. Написать программу, которая из элементов матрицы $C_{12\times1}$ формирует матрицу $D_{3\times4}$, заполняя ее по столбцам.

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

10. Дана последовательность целых чисел $a_1, a_2, \ldots, a_n$ и целочисленная квадратная матрица порядка n. Написать программу, заменяющую нулями элементы матрицы с четной суммой индексов, для которых имеются равные среди членов последовательности.

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

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

13. Написать программу, в которой элементам квадратной матрицы 10-го порядка присвоить значения 1, 2, …, 100 по спирали.

14. Написать программу, которая из элементов массива A (a_1, a_2, …, a_{21}) формирует массив

$$B = \begin{pmatrix} a_1 & 0 & 0 & 0 & 0 & 0 \\ a_2 & a_3 & 0 & 0 & 0 & 0 \\ a_4 & a_5 & a_6 & 0 & 0 & 0 \\ a_7 & a_8 & a_9 & a_{10} & 0 & 0 \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & 0 \\ a_{16} & a_{17} & a_{18} & a_{19} & a_{20} & a_{21} \end{pmatrix}$$

и выводит его на экран.

15. 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).

Введение в классы

Определение, назначение и состав классов

Классы — основа любого объектно ориентированного языка. Класс представляет собой некоторый шаблон, по которому будут создаваться объекты определенного типа. Внутри класса содержатся какие-либо данные и методы их обработки. По существу класс — это ряд схематических описаний способа построения объекта. Важно понимать, что класс является логической абстракцией, т. е. физическое представление класса появится в оперативной памяти только после создания объекта этого класса. Если говорить в терминах объектно ориентированного программирования, то класс — это инкапсуляция данных и методов обработки этих данных.

Общая форма определения класса содержит данные и функции.

Данными класса могут быть поля, константы, события.

Поля — это любые переменные, относящиеся к классу.

Константы могут быть ассоциированы с классом так же, как и переменные. При объявлении константы используется ключевое слово const.

События — это члены класса, которые уведомляют программный код о том, что случилось что-то важное, заслуживающее внимания, например, изменилось какое-либо свойство класса или произошло взаимодействие с пользователем.

Функциями класса могут быть методы, конструкторы, свойства, операции, финализаторы.

Методы — это функции, определяющие поведение объектов класса. Методы служат для обработки данных.

Свойства — это видимые извне признаки класса. Если провести параллель с человеком, то свойствами могут быть цвет глаз и волос, рост, масса, возраст и т. д.

Конструкторы — специальные функции, используемые для инициализации объектов класса в процессе их реализации.

Финализаторы — функции, которые вызываются, когда среда CLR определяет, что объект больше не нужен. Имя этих функций совпадает с именем класса. Невозможно сказать, когда будет вызван финализатор.

Операции — это простейшие действия типа сложения или вычитания. В языке C# имеется возможность указать, как существующие операции будут взаимодействовать с пользовательскими классами (это еще называют перегрузкой операций).

Модификаторы доступа

Модификаторы доступа используются для задания степени видимости или доступности какого-либо члена класса для кода программы, который находится за пределами класса.

Модификаторы доступа могут быть следующих видов:

public — член класса доступен вне определения класса и в производных классах;

protected — член класса невидим за пределами класса, но к нему могут обращатся производные классы;

private — член класса не доступен за пределами класса. При этом доступ к этим членам для производных классов также закрыт;

internal — члена класса доступен только в пределах текущей единицы компиляции.

В языке C# для членов класса по умолчанию используется модификатор доступа private, т. е. если явно не указать модификатор доступа переменной, то будет использован модификатор *private*.

Инициализация классов и конструкторы

Рассмотрим синтаксис определения класса. Ниже представлено определение класса, который содержит только переменные и методы:

```
class имя_класса {
//Объявление переменных экземпляра.
доступ тип переменная1;
доступ тип переменная2;
```

```
//…
доступ тип переменнаяА;
//Объявление методов.
доступ возвращаемый_тип метод1 (параметры)
{
//тело метода
}
доступ возвращаемый_тип метод2 (параметры)
{
//тело метода
}
//...
}
```

Рассмотрим пример определения класса, описывающего некоторого студента вуза. Код будет выглядеть следующим образом:

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

namespace ConsoleApplication12
{
class Student
{
//описываем характеристики объекта класса
//характеристики доступны только внутри данного класса
private string name;
private string sename;
private int kurs;
private string adress;
private int tel;

//определяем конструктор класса для создания объектов данного класса
//в скобках указаны параметры, которые передаются методу для создания//объекта с
//указанными свойствами
```

```
public Student (string name, string sename, int kurs, string adress, int tel)
{
this.name = name;
this.sename = sename;
this.kurs = kurs;
this.adress = adress;
this.tel = tel;
}
//определяем метод для объекта класса (его поведение)
//метод будет выводить на экран информацию о студенте

public void Print ()
{
Console.WriteLine  (“Имя\t”+name+”\nФамилия\t”+sename+”\nКурс\
t”+kurs+”\nАдрес\t”+adress+”\nТелефон\t”+tel+”\n”);
}
}

class Program
{
static void Main (string [] args)
{
//создаем первый объект нашего класса
Student St1 = new Student («Иван»,»Иванов»,1,»Ленина, 50–25»,3251565);
//создаем второй объект нашего класса
Student St2 = new Student («Петр», «Петров», 2, «Мира, 15–3», 2256580);

//используем метод вывода на экран информации о студентах,
//определенный ранее в классе
St1.Print ();
St2.Print ();
Console.ReadKey ();
}
}
}
```

Результат работы программы показан ниже (рис. 20).

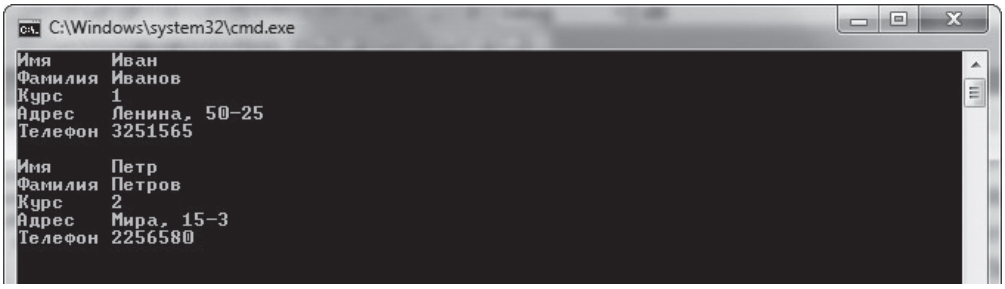

Рис. 20. Результат работы программы

Задания для самостоятельной работы

1. Описать класс «Почтовый адрес организации». Необходимо создать следующие методы класса:

- изменение составных частей адреса;
- создание и уничтожение объектов этого класса.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

2. Описать класс «Комплексное число». Необходимо создать следующие методы класса:

- операция сложения комплексных чисел;
- операция вычитания комплексных чисел;
- операция умножения комплексных чисел.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

3. Описать класс «Векторы, задаваемые координатами концов в трехмерном пространстве». Необходимо создать следующие методы класса:

- операции сложения и вычитания векторов с получением нового вектора;
- вычисление скалярного произведения двух векторов и длины вектора;
- операция вычисления косинуса угла между векторами.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

4. Описать класс «Одномерный массив целых чисел». Необходимо создать следующие методы класса:

- обращение к отдельному элементу массива с контролем выхода за границы массива;
- возможность задания размерности массива при создании объекта и выполнения операций сложения (вычитания) массивов;
- умножение элементов массива на число;
- вывод в консоль элемента массива по заданному индексу и всего массива.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

5. Описать класс «Домашняя библиотека». Необходимо создать следующие методы класса:

- работа с произвольным числом книг;
- поиск книги по какому-либо признаку (например, по автору или по году издания);
- добавление книг в библиотеку, удаление книг из нее;
- сортировка книг по разным полям.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

6. Описать класс «Записная книжка». Необходимо создать следующие методы класса:

- работа с произвольным числом записей;
- поиск записи по какому-либо признаку (например, по фамилии, дате рождения или по номеру телефона);
- добавление и удаление записей;
- сортировка по разным полям.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

7. Описать класс «Студенческая группа». Необходимо создать следующие методы класса:

- работа с переменным числом студентов;
- поиск студента по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона);
- добавление и удаление записей;
- сортировка по разным полям.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

8. Описать класс «Предметный указатель». Каждая компонента указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, — от одного до десяти. Необходимо создать следующие методы класса:

- формирование указателя с клавиатуры и из файла;
- вывод указателя;
- вывод номеров страниц для заданного слова;
- удаление элемента из указателя.

Продемонстрировать работу с этим классом. Обязательно наличие меню, через которое можно проверить работу всех методов класса.

Рекомендуемый библиографический список

Мейер Б. Объектно ориентированное конструирование программных систем / Б. Мейер. М. : Русская Редакция, 2005. 1204 с.

Буч Г. UML / Г. Буч, А. Якобсон, Дж. Рамбо. СПб. : Питер, 2006. 656 с.

Троелсен Э. C# и платформа.NET / Троелсен Э. СПб. : Питер, 2006. 796 с.

Шилдт Герберт. C# 3.0, 4.0. Полное руководство / Герберт Шилдт. СПб. : Питер, 2011. 1056 с.

Шилдт Герберт. C# 3.0: руководство для начинающих / Герберт Шилдт. СПб. : Питер, 2009. 992 с.

Шилдт Герберт. Полный справочник по C / Герберт Шилдт. СПб. : Питер, 2004. 752 с.

Уотсон Карли. Visual C# 2008. Базовый курс / Карли Уотсон. Киев : Диалектика, 2009. 1216 с.

C# 4.0 и платформа.NET 4 для профессионалов / Нейгел Крис [и др.]. Киев : Диалектика. 2011. 1440 с.

Microsoft ASP. NET 2.0 с примерами на C# 2005 для профессионалов / Мэтью Макдоналд. Киев : Диалектика, 2011. 1408 с.

Язык программирования C# 2005 и платформа.NET 2.0.2007 / Эндрю Троельсен. Киев : Диалектика, 2011. 1168 с.

Либерти Джесс. Создание .NET приложений. Программирование на C# / Джесс Либерти. Киев : Диалектика, Вильямс, 2003. 686 с.

Первые шаги [Электронный ресурс]. Режим доступа: http://www.firststeps.ru. Загл. с экрана.

SIMPL C# [Электронный ресурс]. Режим доступа: http://simple-cs.ru. Загл. с экрана.

Оглавление

Учебное издание

Медведев Максим Александрович,
Медведев Александр Николаевич

ПРОГРАММИРОВАНИЕ НА СИ#

Редактор И. В. Меркурьева
Верстка О. П. Игнатьевой
Набор М. А. Медведевой

Подписано в печать 21.10.2015. Формат 70×100/16.
Бумага писчая. Плоская печать. Гарнитура Charter.
Уч.-изд. л. 3,5. Усл. печ. л. 5,2. Тираж 80 экз.
Заказ 279

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343)375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru