Оглавление

Практикум 1: Работа с консольным приложением 2
Упражнение 1. Запуск готового приложения. 2
Упражнение 2. Ввод данных. 3
Упражнение 3. Арифметические и математические вычисления. 5
Упражнение 4. Развилки. 6
Упражнение 5. Выбор. 7
Упражнение 6. Цикл с параметром. 7
Теоретические отступления 9
1. Структура программы консольного приложения на Visual C# 9
2. Основы синтаксиса языка Visual C# 11
3. Типы данных. Методы класса Math. Форматный вывод данных 16
4. Условный оператор 22
5. Оператор выбора Switch 23
6. Оператор цикла с параметром 25

Практикум 1: Работа с консольным приложением

В данном практикуме будут рассмотрены особенности интерфейса среды разработки, а также синтаксические особенности языка C#.

Если Вы имеете некоторый опыт написания приложений на языке C#, то можете пропустить данный практикум и перейти к этапу 2 «Введение в визуальное конструирование»

Примечание. К этому моменту среда Visual Studio 2015 (далее в тексте VS) должна быть установлена на Вашем компьютере. В процессе установки в папке ***Документы*** (Мои документы и т.п. в зависимости от версии ОС) была создана **папка *Visual Studio 2015***. Помимо папок со служебной информацией, там находится папка ***Projects***. Это та папка, в которую среда VS будет **по умолчанию сохранять** созданные проекты, если не будет указан при сохранении иной путь.

Далее в тексте будет подразумеваться, что папки с решениями упражнений скопированы Вами и размещены в папке ***Visual Studio 2015 -- Projects***.

Упражнение 1. Запуск готового приложения.

1. Откройте папку *Examples*. Рассмотрите содержимое. В папке находятся файлы одного решения, состоящего из нескольких проектов *Example_1, Example_2 … Example_6*.
2. Исполняемые файлы для каждого проекта находятся внутри папок. Например, файл *Example_1.exe* находится в папке *Exampes -- Example_1 – bin—Debug*[1]. Такой файл может выполняться и на компьютере, на котором не установлена VS. Найдите и запустите исполняемый файл первого проекта.
3. Запустите файл проекта Example_1.exe.
4. Для того чтобы получить доступ к коду программы необходимо открыть решение в среде разработки. Это можно сделать одним из двух способов:

   а) запустить файл *Examples.sln* (Examples -- это имя всего решения), при этом будет сначала запущена среда разработки VS, а затем загружены все проекты данного решения

   б) запустить среду программирования и открыть файл *Examples.sln*, используя команду **Открыть проект** (Рисунок 2).

[1] Этот же файл можно найти и по другому адресу: *Exampes -- Example_1 – obj -- Debug*

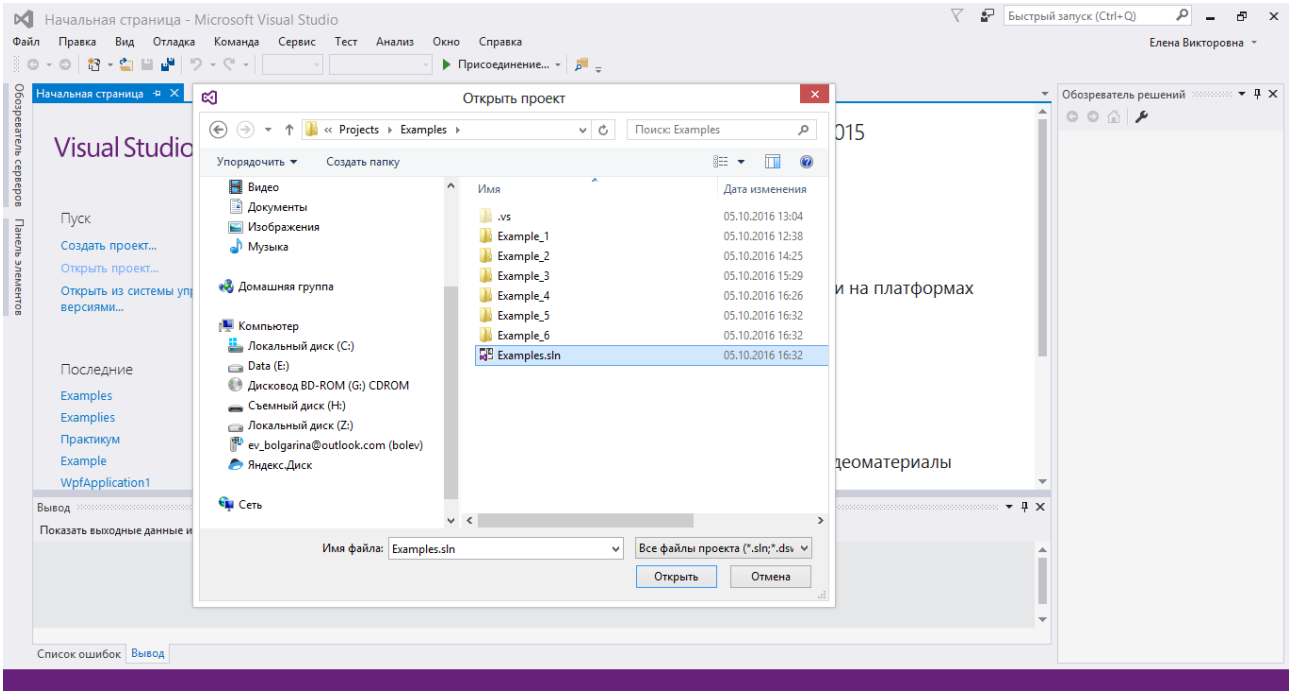

Рисунок 2. – Диалоговое окно среды разработки Visual studio «Открыть проект»

5. Запустите решение *Example.sln* в среде разработки VS.
6. Рассмотрим интерфейс окна проекта (Рисунок 3.)

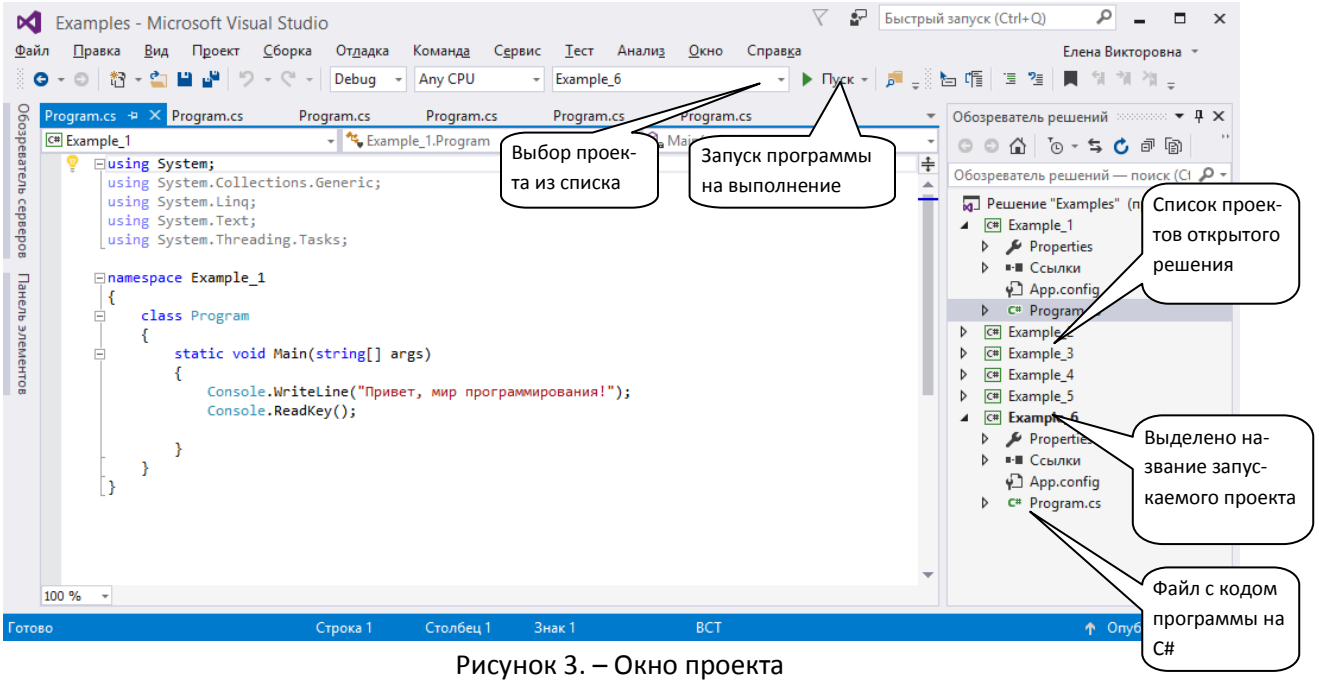

Рисунок 3. – Окно проекта

7. Вы можете запустить на выполнение любой из проектов, выбрав его название из выпадающего списка и нажав на кнопку **Пуск** (*горячая клавиша* **F5**). При этом имя выполняемого проекта в списке *Обозревателя решений* будет выделено полужирным шрифтом.
8. В окне кода представлен вариант кода традиционного первого приложения типа «Hello, World!». На его примере можно видеть типичную структуру программы на языке C#.

☞ 1. Структура программы консольного приложения на Visual C#

Упражнение 2. Ввод данных.

1. Переключитесь на код проекта *Example_2 (Листинг 1)*.

Листинг 1

```
static void Main(string[] args)
        {
            //В данном примере проиллюстрированы некоторые варианты ввода  и вывода данных


            //Описание переменной типа Строка выполнено одновременно с присвоением начально-
го значения
            string name = "";
            Console.Write("Как Вас зовут?");
            name = Console.ReadLine();
            Console.WriteLine("В каком году Вы родились?");

            /*Описание целочисленной переменной в процессе ввода данных
            Считанное значение имеет тип String, поэто в процессе присваивания оно преобра-
зуется в целочисленный тип*/

            int year = Convert.ToInt16(Console.ReadLine());

            //Вывод на экран результата вычисления

            Console.WriteLine("Очевидно, что Вам, {0}, исполнилось {1} или {2} лет(года)",
name, Convert.ToInt16(DateTime.Now.Year) -year, Convert.ToInt16(DateTime.Now.Year) - year-
1);
            Console.ReadKey();
            //


        }
    }
}
```

2. На этом примере изучите реакцию среды разработки на допущенные синтаксически ошибки. Например, уберите символ «;» в конце строки. Сохраните изменение в проекте (лучше использовать команду **Сохранить все** *Ctrl+Shift+S*).

В появившемся окне *Список ошибок* будет выведено сообщение (Рисунок 4.) Если окно сообщений об ошибках автоматически не появилось, то откройте его командой **Вид – Список ошибок** *(Ctrl + \, E)*. Кроме того, в коде программы место с ошибкой будет отмечено красной волнистой линией. При наведении выводится сообщение о допущенной ошибке.

Двойной щелчок мышью по описанию ошибки в окне Списка ошибок перенесет курсор в то место кода, где допущена ошибка.

Способ сообщения об ошибках делает среду разработки одной из самых дружелюбных и значительно упрощает процесс изучения программирования, особенно на первых порах.

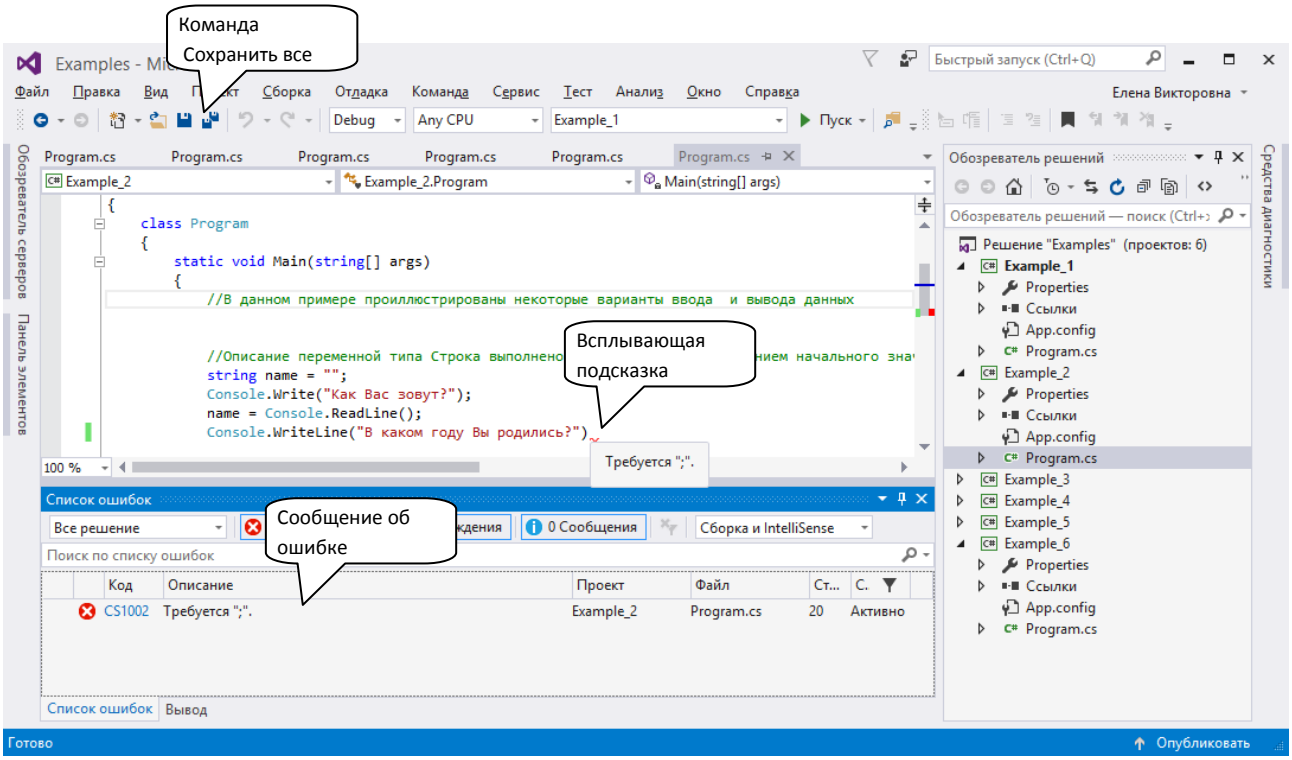

Рисунок 4. – Окно Список ошибок

☞ 2. Основы синтаксиса языка Visual C#

Упражнение 3. Арифметические и математические вычисления.

Переключитесь на код проекта *Example_3 (Листинг 2)*

Листинг 2

```
static void Main(string[] args)
        {
            // В данном коде приведены примеры некоторых математических вычислений и особен-
ности форматированного вывода

            double pi, x,c;
            int a,b, rez;
            pi = Math.PI;
            Console.WriteLine("Примеры вывода форматированных данных");
            Console.WriteLine("Вывод числа ПИ без форматирования {0}", pi);
            Console.WriteLine("В экспоненциальной форме {0:E}", pi);
            Console.WriteLine("С точностью до сотых {0:F2}", pi);
            Console.WriteLine("Только целая часть {0:F0}", pi);

            Console.WriteLine("Примеры работы некоторых методов класса Math");
            Console.WriteLine("Корень из числа ПИ {0}", Math.Sqrt(pi));
            Console.WriteLine("ПИ в квадрате {0}", Math.Pow(pi,2));
            Console.WriteLine("Целая часть числа ПИ в кубе {0} \n\r",
Math.Truncate(Math.Pow(pi,3)));

            //Деление двух целочисленых аргументов возвращает целую часть результата
            Console.WriteLine("Некоторые особенности деления. \n\r Переменные в выражении -
целочисленные, результат описан вещественным типом");
            Console.WriteLine("Выполнено присваивание.\n\r A=5,  B=3");
            a = 5;
            b = 3;
            c = a / b;
            Console.WriteLine("Результат деления числа {0}  на число {1} равен {2} \n\r",
a,b,c);
```

```
            Console.WriteLine("Внесены изменения. \n\r Один из операндов выражения описан
вещественным типом");
            Console.WriteLine("Выполнено присваивание.\n\r X=5.0,  B=3");
            x = 5.0;
            b = 3;
            c = x / b;
            Console.WriteLine("Результат деления числа {0}  на число {1} равен {2}\n\r", x,
b, c);

            Console.WriteLine("Остаток от деления. \n\r Операнды должны быть описаны цело-
численным типом");
            rez = a % b;
            Console.WriteLine("Остаток деления числа {0}  на число {1} равен {2}", x, b,
rez);
            Console.ReadKey();


        }
```

В данном проекте следует обратить внимание на особенности выполнения арифметических операций. Приведены примеры использования методов класса Math. Полезным могут оказаться примеры форматированного вывода данных.

☞ 1. Структура программы консольного приложения на Visual C# данных. Методы класса Math. Форматный вывод данных

Упражнение 4. Развилки.

Переключитесь на код проекта *Example_4 (Листинг 3)*

Данный код иллюстрирует использование оператора условия. Тексты решаемых задач приведены в комментариях кода.

Листинг 3

```
static void Main(string[] args)
        {
            //Пример использования сокращенной формы условного оператора
            Console.WriteLine("Задача 1. Вычисление тригонометрических функций.");
            //подсказка для пользователя при вводе значения
            Console.Write("Введите значение аргумента");
            double X = Convert.ToDouble(Console.ReadLine());
            //уточнение о единицах измерения аргумента
            Console.Write("Аргумент задан в градусах (да/нет)?");
            //результат уточнения сохраняется в строковой переменной a
            string a = Console.ReadLine();
            //использование сокращенной формы условного оператора
            if (a == "да") X = X * Math.PI / 180;
            //вывод результатов вычисления
            Console.WriteLine("Значение Cos(X)={0}", Math.Cos(X));
            Console.WriteLine("Значение Sin(X)={0}", Math.Sin(X));
            Console.WriteLine("Для продолжения нажмите любую клавишу...");
            Console.ReadKey();

            //Пример использования полной формы условного оператора и сложного условия

            Console.WriteLine("Задача 2. Пример сложного условия покупка со скидкой");
            Console.WriteLine("покупатель получает скидку на покупку, совершенную до 12 ча-
сов дня n\r в любой день, кроме субботы и воскресенья.");
            DateTime t ; // час покупки
            string day = ""; //день недели

            Console.Write("Введите день недели (пн; вт; ср; чт; пт; сб; вс)");
             day= Console.ReadLine();
            Console.Write("Введите время совершения покупки (в формате 10:25) ");
```

```
            t = Convert.ToDateTime(Console.ReadLine());
            int h = t.Hour;

            //Обратите внимание, что условие заключается в круглые скобки
            //Обратите внимание, на точку с запятой перед else

            if (((day != "вс") && (day != "сб")) && (h<12) ) Console.WriteLine("Вам положена
скидка на товар!"); else Console.WriteLine("К сожалению скидка на товар Вам не положена!");


            Console.ReadKey();

        }
```

Обратите внимание на использование методов класса ***DateTime***.

В целом условный оператор не имеет существенных особенностей, за исключением, возможно, синтаксиса. Логическое равенство обозначается двойным символом «==»

☞ 1. Структура программы консольного приложения на Visual C# данных. Методы класса Math. Форматный вывод данных

Упражнение 5. Выбор.

Переключитесь на код проекта *Example_5 (Листинг 4)*

Использование оператора Switch проиллюстрировано решением задачи об элементной базе поколений ЭВМ

Листинг 4

```
static void Main(string[] args)
        {
            //Пример работы оператора Выбор
            Console.WriteLine("Для получения информации об элементной базе, введите номер
поколения ЭВМ");
            byte n = Convert.ToByte(Console.ReadLine());
            string s;
            switch (n)
            {
                case 1: s = "Элементная база: электронно-вакуумные лампы,\n годы использова-
ния: 1946--1958"; break;
                case 2: s = "Элементная база: транзисторы,\n годы использования: 1958--
1964"; break;
                case 3: s = "Элементная база: интегральные схемы,\n годы использования:
1964--1975"; break;
                case 4: s = "Элементная база: большие интегральные схемы,\n годы использова-
ния: 1975 по наст время"; break;
                case 5: s = "Элементная база: суперкомпьютеры,\n развитие искусственного
интеллекта: в стадии становления"; break;
                default: s = "Вы ошиблись с вводом значения"; break;
            }
            Console.WriteLine("Информация о компьютерах {0}-го поколения", n);
            Console.WriteLine(s);
            Console.ReadKey();

        }
```

Комбинация управляющих символов *\n* приводит к переводу каретки, т.е. переносу строк.

☞ 1. Структура программы консольного приложения на Visual C# данных. Методы класса Math. Форматный вывод данных

Упражнение 6. Цикл с параметром.

Язык Visual C# имеет средства для описания всех, используемых в программировании, типов циклов: с предусловием, с постусловием, с параметром. При создании приложений Windows Forms будут представлены варианты этих циклов. На данном, базовом этапе, познакомимся с циклом с параметром, существующим во всех языках программирования. Цикл с параметром, при гибком его использовании позволит решить практически все задачи с использованием циклических вычислений.

Переключитесь на код проекта *Example_6 (Листинг 5)*

Листинг 5

```
static void Main(string[] args)
        {
            // Пример работы цикла с параметром
            //Задача: Напечатать таблицу квадратов и кубов для первых двадцати положительных
чисел.
            for (int i = 1; i <= 20; i++)
                Console.WriteLine("{0} {1} {2}\t", i, Math.Pow(i, 2), Math.Pow(i, 3));
            Console.ReadKey();

               }
```

Управляя условием и начальным значением параметра можно организовать перебор значений в теле цикла, как в сторону увеличения, так и в сторону уменьшения. Естественно, что в теле цикла переменная-параметр не должна изменять своего значения. Впрочем, компилятор C# своевременно предупредит об этом.

Управляющая комбинация символов *\t* организует вывод данных в табличном виде (в виде колонок).

☞ 1. Структура программы консольного приложения на Visual C# данных. Методы класса Math. Форматный вывод данных

На этом рассмотрение особенностей синтаксиса языка C# можно считать законченным. В дальнейшем Вы будете создавать приложения для Windows – приложения Windows Forms.

Теоретические отступления

1. Структура программы консольного приложения на Visual C#

На рис. 1.1 представлен код шаблона приложения, сгенерированный автоматически средой программирования.

Первые четыре строки – *директивы* подключения стандартных библиотек классов.

using System – это директива, которая разрешает использовать имена стандартных классов из пространства имен `System` непосредственно, без указания имени пространства, в котором они были определены.

Ключевое слово *namespace* создает для проекта свое собственное *пространство имен*, которое по умолчанию называется именем проекта. В данном случае пространство имен называется **Мир**. Однако программист вправе указать другое имя.

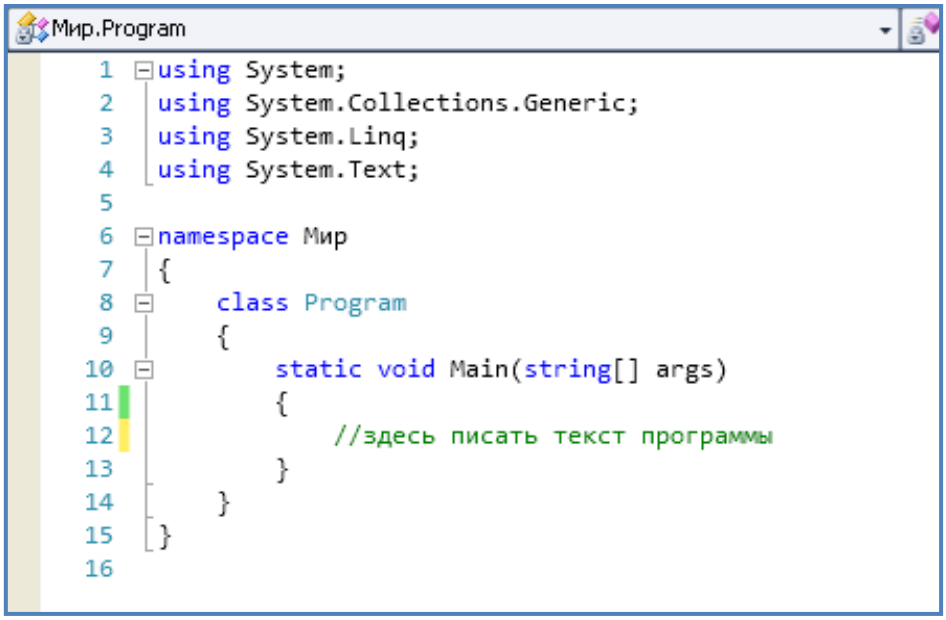

Рисунок 1.1. -- Автоматически сгенерированный код шаблона приложения

Пространство имен ограничивает область применения имен, делая его доступным только в рамках данного пространства. Это сделано для того, чтобы можно было давать имена программным объектам, не заботясь о том, что они совпадут с именами в других приложениях. Таким образом, пространства имен позволяют избегать конфликта имен программных объектов, что особенно важно при взаимодействии приложений.

Пространство имен -- это не более чем группа типов данных, но дающая тот эффект, что имена всех типов данных в пределах пространства имен автоматически снабжаются префиксом -- названием пространства имен. Пространства имен можно вкладывать друг в друга. Например, большинство базовых классов .NET общего назначения находятся в пространстве имен System. Базовый класс Array относится к этому пространству, поэтому его полное имя -- System.Array.

Платформа .NET требует, чтобы все имена были объявлены в пределах пространства имен.

C# – *объектно-ориентированный* язык, поэтому написанная на нем программа будет представлять собой совокупность взаимодействующих между собой *классов*. Автоматически был создан класс с именем Program. Данный класс содержит только один *метод* – метод *Main()*.

Метод Main() является точкой входа в программу, т.е. именно с данного метода начнется выполнение приложения. Каждая программа на языке C# должна иметь метод Main ().

Метод Main() имеет одну важную особенность: перед объявлением типа возвращаемого значения `void` (который означает, что метод не возвращает значение) стоит ключевое слово `static`, которое означает, что метод Main() можно вызывать, не создавая *объект* типа Program.

Добавим в метод следующий код:

```
Console.WriteLine("Здравствуй, Мир программирования!");
```

Здесь *`Console`* – имя стандартного класса из пространства имен *`System`*. Его метод *`WriteLine`* выводит на экран текст, заданный в кавычках.

Обратите внимание: при вводе первых символов служебного слова *`Console`* среда программирования выводит список возможных команд (рисунок 1.2) – это работает механизм интеллектуальных подсказок, имеющийся в Visual Studio.

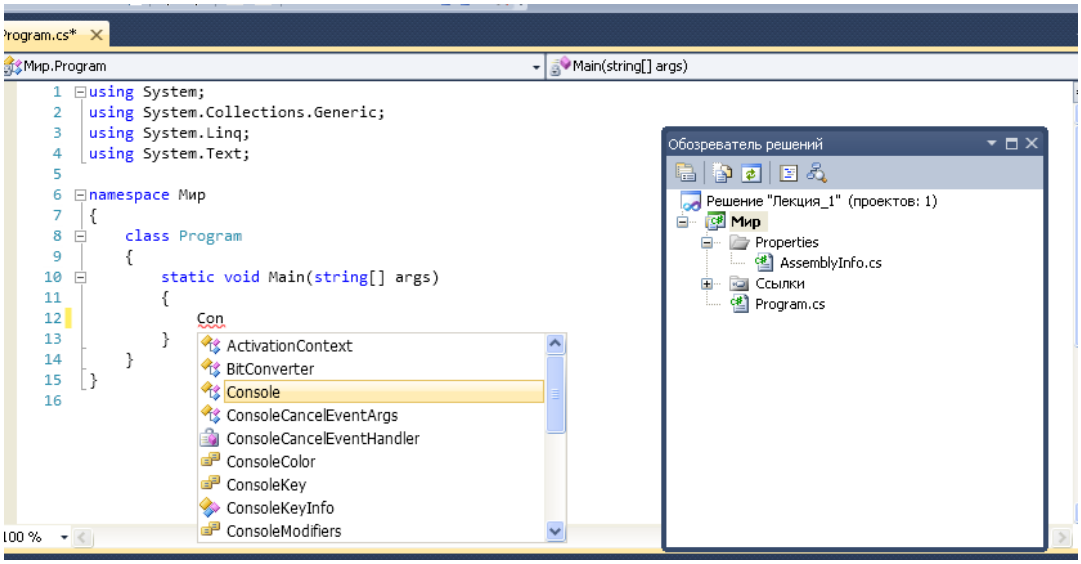

Рисунок 1.2. -- Появление интеллектуальной подсказки

Правильнее будет не набирать код программы до конца вручную, а завершить предложенный в списке вариант нажатием на клавишу **Enter**. Тем самым вы не только ускоряете ввод команд, но и избегаете возможных ошибок ввода. Особенно важно прислушаться к подсказкам при выборе методов класса. После того, как вы набрали имя класса *`Console`* и поставили точку, у вас появляется список всех возможных методов этого класса. И если при вводе программист обнаруживает, что нужного метода в списке нет, то это означает, что он совершил ошибку: либо ему нужен другой метод, либо он выбрал неверный класс.

Для запуска программы следует нажать клавишу F5 или выполнить команду **Отладка – Начать отладку**. Если программа написана без ошибок, то текст выведется в консольное окно, которое мелькнет и быстро закроется. Чтобы этого не произошло и окно задержалось на экране столько, сколько это требуется пользователю (т.е. до нажатия любой другой клавиши), в текст программы добавьте следующую команду:

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

Результатом работы вашего первого приложения будет окно, изображенное на рисунок 1.3.

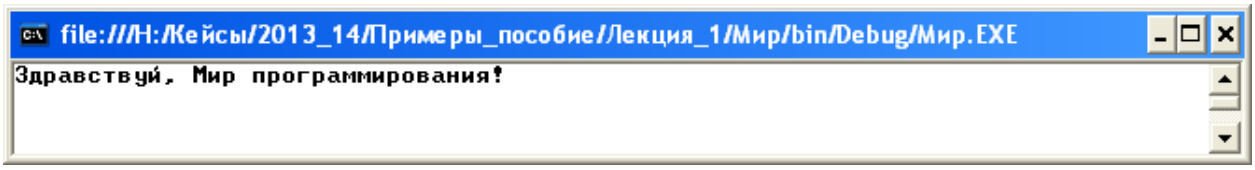

Рисунок 1.3. -- Результат работы приложения

☞Назад к основному тексту (стр. 12)

1. Структура программы консольного приложения на Visual C#

2.1. Идентификаторы

Имена операторов, переменных, констант, типов величин, имя самой программы назначаются программистом и называются ***идентификаторами***. Существуют правила, которым должны отвечать все идентификаторы:

- идентификатор должен быть уникальным, то есть одним и тем же именем разные объекты названы быть не могут;
- идентификатор может иметь ограничение по длине (зависит от конкретной реализации языка на компьютере);
- идентификатор может состоять из символов и цифр, допустимо использовать знак подчеркивания ("_"), регистр букв имеет значение;
- идентификатор не может начинаться с цифры.

```
//допустимые идентификаторы
x  ,  s5  ,  Ax6 ,  D_f  , _Fm5 , xM_5 , _128
```

Для того чтобы облегчить восприятие и понимание последовательностей операторов, желательно создавать составные осмысленные имена. При создании подобных имен в одно слово можно «втиснуть» предложение, которое в доступной форме представит информацию о типе объекта, его назначении и особенностях использования.

Например, `procentInMonth` будет подходящим именем для переменной, хранящей значение процента за месяц при оплате кредита.

Вы можете давать программным объектам любые имена, но необходимо, чтобы они отличались от *зарезервированных слов*, используемых языком (компилятор все равно не примет переменные с «чужими» именами).

Зарезервированные (ключевые) слова имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Далее приведен список наиболее часто встречающихся зарезервированных слов языка C#.

abstract	extern	out	typeof	checked	const	int
event	object	true	catch	goto	implicit	sbyte
new	this	byte	for	public	ref	virtual
struct	bool	fixed	private	un-	ushort	default
as	false	override	uint	checked	continue	interface
explicit	operator	try	char	class	in	sealed
null	throw	case	foreach	if	return	volatile
switch	break	float	protected	readonly	using	
base	finally	params	ulong	unsafe	decimal	

2.2. Константы

Константы представляют собой неизменные значения, известные во время компиляции и неизменяемые на протяжении времени существования программы. Константы незаменимы при использовании в тексте программы многократно повторяемых значений, и удобны в случае необходимости изменения этих значений сразу во всей программе.

Константы объявляются с помощью ключевого слова const; их использование способствует повышению удобочитаемости кода. Например, при решении физических задач в качестве константы можно задать стандартное ускорение свободного падения на поверхности Земли.

В языке существует два вида констант:

- неименованные константы (цифры и числа, символы и строки, множества);
- именованные константы.

2.3. Неименованные константы

Неименованные константы, как следует из названия, не имеют имен, и потому их не нужно описывать. Тип неименованной константы определяется автоматически, по умолчанию:

- любая последовательность цифр (возможно, предваряемая знаком «-» или «+» или разбиваемая одной точкой) воспринимается компилятором как неименованная константа – число (целое или вещественное);
- любая последовательность символов, заключенная в апострофы, воспринимается как неименованная константа – строка;
- любая последовательность целых чисел либо символов через запятую, обрамленная квадратными скобками, воспринимается как неименованная константа – множество.
- Кроме того, существуют две специальные константы – true и false, относящиеся к логическому типу данных.

Примеры неименованных констант:

```
a = -10;
b = 12.075 + x;
c = 'z';
d = "abc" + string44;
```

2.4. Именованные константы

Именованные константы, как тоже следует из их названия, должны иметь имя. Эти имена необходимо описать в специальном разделе const.

Если не указывать тип константы, то по ее внешнему виду компилятор сам определит, к какому (базовому) типу ее отнести. Любую уже описанную константу можно использовать при объявлении других констант, переменных и типов данных. Вот несколько примеров описания именованных констант:

```
const int speedLimit = 55;
const double pi = 3.14159265358979323846264338327950;
```

2.5. Переменные

В языке C# переменные объявляются перед первым их использованием в программе, т.е. для каждой переменной указывается определенный тип данных и идентификатор. Тип указывает точный объем памяти, который компилятор выделит в памяти для хранения значения при выполнении приложения. Компилятор языка C# требует, чтобы любая переменная была инициализирована некото-

рым начальным значением, прежде чем можно было обратиться к ней в какой-то операции. Например:

```
int answer = 42;
string greeting = "Hello, World!";
double bigNumber = 1e100;
Console.WriteLine("{0} {1} {2}", answer, greeting, bigNumber);
```

2.6. Комментарии

В текстах C#-программ допускаются фрагменты пояснительного характера – комментарии. Наличие комментариев не изменяет смысл программы и не влияет на ее выполнение.

В языке C# комментарии представляют собой произвольную последовательность символов, заключенную в фигурные символы /* и */, например:

```
/*Это многострочный
 комментарий*/
```

или строки, начинающиеся с **// ,** например:

```
//Это однострочный комментарий
```

Старайтесь писать комментарии с таким расчетом, чтобы, взяв свою программу через полгода, вы смогли понять, как она работает. **Не экономьте на комментариях.** Комментарии большого размера не ухудшают качества программы. Пример программы с комментариями приведен в листинге 2.1. В языке C# по умолчанию комментарии выделяются зеленым цветом.

2.7. Операторы языка

Операторы в языке программирования – это синтаксические конструкции, предназначенные как для записи алгоритмических действий по преобразованию данных, так и для задания порядка выполнения других действий.

Набор операторов языка программирования составляет минимальное множество конструкций, необходимых для наглядного и однозначного представления алгоритмов в стиле структурного программирования.

2.7.1. Оператор присваивания

Самым простым действием над переменной является занесение в нее величины соответствующего типа, т.е. присвоение переменной конкретного значения. Такая команда (оператор) в общем виде выглядит следующим образом:

```
<Имя переменной> = <Выражение>;
```

Тип переменной и тип выражения должны быть совместимы.

Выражение, указанное справа от знака «=», должно приводить к значению того же типа, какого и сама переменная, или типа, совместимого с переменной относительно команды присваивания. Например, переменной типа `double` можно присвоить значение типа `Int` или `byte` (наоборот делать нельзя). Выражение будет сначала вычислено, затем его результат будет положен в ячейки памяти, отведенные для переменной.

Выражение – это конструкция, возвращающая значение. Например:

```
A = X;            // переменная
B =15;            // целая константа
E = X * Y;        // произведение  X на Y
```

```
F = Z / (1 - Z); // отношение Z к  (1 - Z)
```

Так же можно делать множественное присваивание (во все переменные заносится самое правое значение):

```
a = b = c = d = 18;
```

2.7.2. Операторы ввода

Для того чтобы получить данные, вводимые пользователем вручную с клавиатуры (то есть с консоли), применяется метод ***ReadLine().***

```
<строковая переменная>=Console.ReadLine()
```

Результатом ввода с помощью оператора `Console.ReadLine()` является строка (т.е. значение строкового типа). Для того чтобы *преобразовать* введенное значение к нужному типу данных, можно использовать методы объекта `Convert`:

`Convert.ToInt32`; `Convert.ToDouble`; `Convert.ToBoolean`; `Convert.ToByte`; `Convert.ToChar` и т.п.

Например:

```
Number1 = Convert.ToInt64(Console.ReadLine());
```

В листинге 2.7.1 демонстрируются примеры ввода с клавиатуры значений переменных.

Листинг 2.7.1. *Ввод значений переменных с клавиатуры*

```
using System;
namespace Ex2_1
{
    class Program
    {
        static void Main(string[] args)
        {
     //ввод значений переменных с использованием промежуточной переменной
      Int32 x; Double y; string s;
      s = Console.ReadLine();
      x = Convert.ToInt32(s);
      //без промежуточной строковой переменной
      y = Convert.ToDouble(Console.ReadLine());
  }
    }
}
```

Типы вводимых значений должны совпадать с типами указанных переменных, иначе возникает ошибка. Поэтому нужно внимательно следить за правильностью вводимых данных.

Если запустить консольное приложение на выполнение, то на экране мелькнет окно с результатом и исчезнет. Для того чтобы задержать окно на экране до тех пор, пока пользователь не нажмет какую либо клавишу, можно использовать следующий оператор:

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

2.7.3. Операторы вывода

Ожидая от пользователя ввода значения с клавиатуры, не нужно полагать, что он просто по мерцанию курсора на черном экране догадается, какого типа переменная нужна ожидающей программе. Следует всегда придерживаться правила: перед тем, как запросить что-либо с консоли, необходимо сообщить пользователю, что именно он должен ввести – характер вводимой информации, тип данных, максимальное и минимальное допустимые значения и т.п.

Примером хорошего приглашения служит, скажем, такая строчка: *Введите два вещественных числа (0.1<1000000) – длины катетов*.

Для вывода на экран сообщения воспользуйтесь методами `Console.Write` или `Console.WriteLine`.

Первый из них, напечатав на экране все, о чем его просили, оставит курсор в конце выведенной строки, а второй переведет курсор в начало следующей, например:

```
Console.WriteLine(s);// переменная
Console.Write (55.3);// константа
Console.WriteLine(y*3+7);// выражение
```

Можно выводить списком несколько переменных:

```
Console.WriteLine("Это число A={0} далее B={1} и их сумма {2}", a, b, a + b);
```

В языке C# в консольном выводе функций `Console.Write()` и `Console.Writeln()` применяются универсальные индексированные метки {0}, {1}, {2}, и т.д., содержащие порядковый номер выводимого параметра в списке параметров. Эти подстановочные выражения обозначают *знакоместа* (*placeholde*), куда будет выводиться значение соответствующего параметра из списка. Знакоместо должно начинаться открывающей фигурной скобкой, а завершаться закрывающей: {}.

Листинг 2.7.2. *Пример сложения двух чисел*

```
using System;
namespace example_2_2
{
    class Program
    {
        static void Main(string[] args)
        {
      Console.Write("Введите целое число ");
      Int16 x = Convert.ToInt16(Console.ReadLine());
      Console.Write("Введите вещественное число ");
      Single y = Convert.ToSingle(Console.ReadLine());
      Single z = x + y;
      Console.WriteLine("Сумма двух чисел:{0}+{1}={2}", x, y, z);
      Console.ReadKey();
        }    }}
```

Результат работы листинга 2.7.2 приведен на рисунке 2.1.

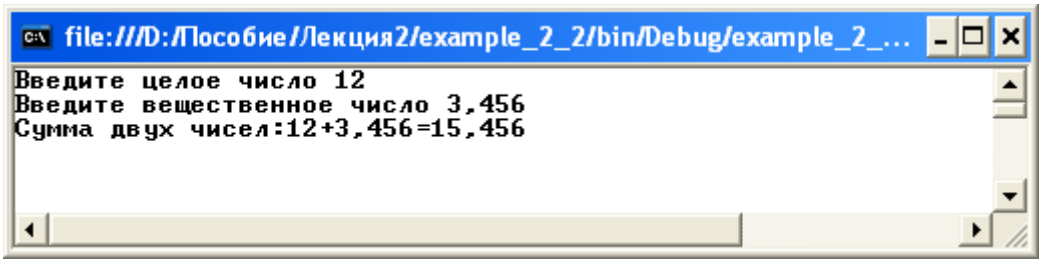

Рисунок 2.1. -- Вид окна выполненной программы листинга 2.7.2

☞ Назад к основному тексту (стр. 13)

1. Структура программы консольного приложения на Visual C# данных. Методы класса Math. Форматный вывод данных

3.1. Типы данных

Данные хранятся в памяти компьютера, но для указания на конкретную информацию очень неудобно все время записывать физические адреса ячеек. Эта проблема в языках программирования высокого уровня решена введением понятия *переменной*.

Переменная – именованный участок памяти для хранения данных определенного типа. Значение переменной (информация в соответствующих ячейках памяти) в ходе выполнения программы может быть изменено. ***Константами*** называются величины, значение которых в ходе выполнения программы изменено быть не может. Конкретные переменные и константы представляют собой объекты уникальные и отличаются друг от друга именем.

В качестве данных в программах могут выступать числа, символы, целые строки символов. С этими различными видами информации выполняются совершенно разные действия. Например, с числовыми величинами производятся арифметические операции, что невозможно сделать с величинами символьными. Кроме того, разные виды данных требуют различного объема памяти для хранения. В соответствии с этими соображениями введено понятие "Тип". ***Тип переменной*** указывает на то, какие данные могут быть сохранены в этом участке памяти и в каких действиях эта переменная может участвовать. Существуют встроенные (зарезервированные, базовые) типы, но есть также возможность создавать свои собственные, определяемые программистом, типы переменных.

Встроенные типы[2]

Список встроенных типов практически одинаков во всех языках программирования. К ним относятся:

- ***Логический тип*** – *bool* (может принимать два значения: Истинно-*true*, Ложно-*false*), например:

```
bool f = true;
```

- ***Целый тип*** – может принимать как положительные, так и отрицательные числовые значения. Однако пользователь может указывать варианты целочисленного типа, которые могут принимать числовые значения из разных диапазонов. Все диапазоны принимаемых значений целого типа и других типов перечислены в таблице 2.1;

- ***Вещественный*** (действительный) тип (то есть – с дробной частью). Примеры обозначения действительного числа:

```
double x = -25.000452
float a = 0.24
double m = 4.854E-12; //соответствует числу, записанному  в
                      //экспоненциальной форме 4,854·10-12 ;
```

- ***Символьный тип*** – *char*. Ключевое слово *char* используется для объявления символа Юникода в диапазоне, указанном в табл. 3.1. Символы Юникода – это 10-разрядные символы, которые используются для представления большинства известных письменных языков мира. Он содержит внутри себя всего один символ, например, 'w' или '#'. Все следующие операторы объявляют переменную *char* и инициализируют ее символом *X*:

[2] Как и большинство современных языков программирования, C# позволяет определять и пользовательские типы. На данном, начальном уровне, об этом речь не ведется.

```
char c1 = 'X';         // Буквенный символ
char c2 = '\x0058';    // Шестнадцатеричный код символа X;
```

- ***Строковый тип*** – *string*; представляет последовательность из нуля или более символов в кодировке Юникод, по умолчанию – до 2 Гбайт. Например:

```
string a = "hello";
string b = "h";
```

Таблица 3.1. Таблица встроенных типов данных в языке C#

Название	**Ключевое слово**	**Тип .NET**	**Диапазон значений**	**Описание**	**Размер, битов**
Логический тип	bool	Boolean	true, false		
Целые типы	sbyte	SByte	От –128 до 127	Со знаком	8
	byte	Byte	От 0 до 255	Без знака	8
	short	Int16	От –32768 до 32767	Со знаком	16
	ushort	UInt16	От 0 до 65535	Без знака	16
	int	Int32	От -2×10^9 до 2×10^9	Со знаком	32
	uint	UInt32	От 0 до 4×10^9	Без знака	32
	long	Int64	От -9×10^{18} до 9×10^{18}	Со знаком	64
	ulong	UInt64	От 0 до 18×10^{18}	Без знака	64
Символьный тип	char	Char	От U+0000 до U+ffff	Unicode-символ	16
Вещественные	float	Single	От 1.5×10^{-45} до 3.4×10^{38}	7 цифр	32
	double	Double	От 5.0×10^{-324} до 1.7×10^{308}	15–16 цифр	64
Финансовый тип	decimal	Decimal	От 1.0×10^{-28} до 7.9×10^{28}	28–29 цифр	128
Строковый тип	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	

Физически типы данных отличаются друг от друга количеством ячеек памяти (байтов), отводимых для хранения соответствующей переменной. Логическое же отличие между ними проявляется в интерпретации хранящейся информации. Например, переменные типа *Char* и типа *Byte* занимают в

памяти по одному байту, однако в первом случае содержимое ячейки памяти интерпретируется как целое беззнаковое число, а во втором – как код (ASCII) символа.

3.2. Форматированный вывод

Управлять видом выводимых значений можно с помощью строки стандартных числовых форматов. В табл. 3.2 приведены поддерживаемые строки. Строка формата принимает следующую форму: ***Axx***, где ***A*** – *описатель формата*, а ***xx*** – *описатель точности*, который управляет количеством значащих цифр или десятичных знаков форматированного результата.

Таблица 3.2. Параметры форматирования в языке C#

Параметр	**Значение**
C, c	Используется для вывода значений в денежном (**Currency**) формате. По умолчанию перед выводимым значением подставляется символ доллара ($)
D, d	Используется для вывода целых десятичных значений (**Decimal**). После этого символа можно указать количество значащих цифр
E, e	Для вывода значений в экспоненциальном формате (**Scientific**)
F, f	Вывод значений с фиксированной точкой (**Fixed-point**)
G, g	Общий (**General**) формат. Применяется для вывода значений с фиксированной точностью или в экспоненциальном формате
N, n	Стандартное числовое форматирование (**Number**) с использованием разделителей (пробелов или запятых) между разрядами
X, x	Вывод значений в шестнадцатеричном формате (**Hexadecimal**). Если использовать прописную X, то символы в шестнадцатеричном формате также будут заглавными
R, r	Округление (**Round-trip**)

Символы форматирования следуют в индексированных знакоместах сразу за номером подставляемого параметра через двоеточие:

```
Console.Write("{0:C}", 2.5); // $2.50
Console.Write("{0:D5}", 25); // 00025
Console.Write("{0:E}", 250000); // 2.500000E+005
Console.Write("{0:F2}", 25); // 25.00
Console.Write("{0:F0}", 25); // 25
```

3.3. Арифметические операции

В языке C# определены все обычные для языков программирования арифметические операции: «+» (сложение), «–» (вычитание), «*» (умножение), «/», «%» (два вида деления).

Приоритет операций сложения и вычитания ниже, чем умножения, деления и вычисления остатка. Для изменения порядка вычисления используют круглые скобки, например, для умножения на 2 суммы двух чисел A и B можно написать: 2*(A+B).

Смена знака. Унарная операция «-» означает смену знака. Она имеет очень высокий приоритет – выше, чем, к примеру, у операции умножения. Поэтому в выражении

-A*B

вначале выполняется смена знака для A, а затем умножение -A на B.

В C# один знак «/» означает две разные операции. Если один или оба операнда вещественные, то выполняется обычное деление, если оба операнда целые, то выполняется деление нацело и результат будет целого типа. Использование этой операции требует повышенной внимательности, например, если запрограммировать вычисление математического выражения $y = \frac{1}{3}x$

буквально, т.е. так: **1/3*x**,

то результат вне зависимости от значения *x* всегда будет равен нулю, так как выражение 1/3 означает деление нацело. Для решения проблемы достаточно один из операндов сделать вещественным: **1.0/3*x**

Чтобы явно установить вещественный тип, можно поместить десятичный разделитель после числа, как показано в следующем примере.

Листинг 3.3.1 *Пример форматированного вывода результата операции деления*

```
static void Main(string[] args)
        {
            Console.WriteLine(1 / 3);
            Console.WriteLine(1D / 3);
            Console.WriteLine(1.0 / 3);
            Console.WriteLine("{0:F3}",1.0 / 3);
            Console.ReadKey();
        }
```

Результат работы листинга 3.3.1 приведен на рисунке 3.1.

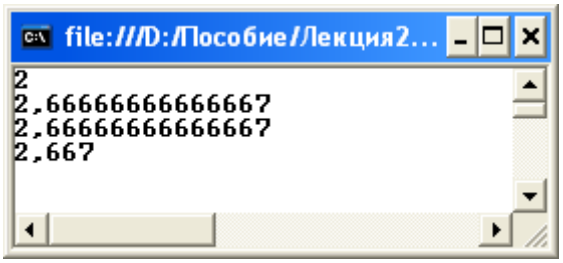

Рисунок 3.1. Вид форматированного вывода результата операции деления

Операция % – остаток от целочисленного деления; так, результат операции 13 % 8 есть число 5. Операции целочисленного деления и нахождения остатка от деления применимы только к целочисленным операндам, их результатом является целое число.

Листинг 3.3.2. *Пример использования операций нахождения остатка от деления и целочисленного деления*

```
static void Main(string[] args)
        {
            Console.WriteLine(24 % 6); // ответ 0
            Console.WriteLine(24 / 6); // ответ 4
            Console.WriteLine(25 % 7); // ответ 4
            Console.WriteLine(25 / 7); // ответ 3
            Console.WriteLine(7 % 7); // ответ 0
```

```
            Console.WriteLine(7 / 7); // ответ 1
            Console.WriteLine(8 % 12); // ответ 8
            Console.WriteLine(8 / 12); // ответ 0
            Console.ReadKey();
        }
```

В языке C# возможна запись некоторых арифметических операций в сокращенном виде, совмещая ее с оператором присваивания. В табл. 3.3 представлены некоторые примеры сокращенной записи арифметических операций и их аналоги в привычной записи.

Таблица 3.3. Сокращенная форма записи арифметических операторов

Операция	Сокращенная запись	Эквивалентная полная запись
Инкремент (увеличение значения на 1)	`A++;`	`A=A+1;`
Декремент (уменьшение значения на 1)	`B--;`	`B=B – 1;`
Сложение с присваиванием	`C + = 2;` `C + = D`	`C = C + 2;` `C = C + D;`
Умножение с присваиванием	`C * = 2;` `C * = D`	`C = C * 2;` `C = C * D`
Вычитание с присваиванием	`C – = 2;` `C -= D`	`C = C – 2;` `C = C – D`
Деление с присваиванием	`C / = 2;` `C / = D`	`C = C / 2;` `C = C / D`
Остаток от деления с присваиванием	`C% = 2;` `C% = D`	`C = C %2 ;` `C = C % D;`
Комбинированная запись	`i += 7 * j;` `m /= 3 + k`	`i = i + 7 * j;` `m = m / (3 + k);`

Операция увеличения[3] (++) увеличивает операнд на 1. Оператор увеличения может находиться как до, так и после операнда. В этом случае

Листинг 3.3.3 *Пример использования сокращенной записи оператора присваивания с вычислением*

```
static void Main(string[] args)
      {
          double x,y;
          x = 1.1; y=2.1;
          x++; y--;
          Console.WriteLine("x={0}  y={1}",x,y);//напечатается x=2.1 y=1.1
          x /= 4 + y;
          Console.WriteLine("{0:F3}",x);   ////напечатается 0.412
          Console.ReadKey();
```

[3] Существуют две формы записи операции, ДО операнда (++x) и после операнда (x++). В некоторых случаях они приводят к разным результатам. Разница объясняется тем, что увеличение переменной может производиться до её участия в вычислении, или же сначала будет выполнено вычисление, и только после этого переменная увеличит сое значение. Аналогично и с операцией инкремента.

```
}
```

3.3. Стандартные функции

В языке C# все часто используемые на практике математические функции заранее разработаны в виде методов класса Math. Рассмотрим наиболее часто используемые методы:

- `Math.Abs(x)` – возвращает модуль числа x;
- `Math.Exp(x)` – вычисление *e* в степени x;
- `Math.Floor(x)` – возвращает наибольшее целое число, которое меньше или равно указанному числу x;
- `Math.Log(x)` – возвращает натуральный логарифм числа x;
- `Math.Log10(x)` – возвращает десятичный логарифм числа x;
- `Math.Pow(B,E)` – возвращает число **B**, возведенное в указанную степень **E**;
- `Math.Round(x)` – возвращает округленное до ближайшего целого или указанного количества десятичных знаков значение числа x;
- `Math.Truncate(x)` – возвращает целую часть числа x;
- `Math.Ceiling(x)` – возвращает наименьшее целое число, которое больше или равно заданному числу x;
- `Math.PI` – возвращает число 3,1415..., математическую константу π ;
- `Math.E` – возвращает число 2,7128..., математическую константу e, основание натурального логарифма;
- `Math.Sin(x)` `Math.Cos(x)` – возвращает значение тригонометрических функций синус, косинус числа x в радианах;
- `Math.Atan(x)` – возвращает значение тригонометрической функции арктангенс в радианах;
- `Math.Sqrt(x)` – возвращает значение квадратного корня числа x;
- `Math.Max(x,y)` – возвращает большее из двух чисел x и y.

Важно: большинство методов класса Math возвращает результатом вещественные числа. Это следует учитывать при описании типа результирующей переменной.

Правила записи математических выражений на языке программирования:

1) выражения записываются в строку;
2) очередность выполнения операций регулируется с помощью скобок;
3) используется только один вид скобок – круглые;
4) если вычисляется значение дроби, в числителе или знаменателе которой записано сложное арифметическое выражение, то числитель или знаменатель надо заключить в скобки.

Пример: Реализовать в виде оператора выражение

$$r = \frac{(a+b)\sin x^2 + ab}{\sqrt{\cos^2\left(x+\frac{\pi}{2}\right) - b}}$$

Решение:

```
R=((a+b)*Math.Sin(Math.Pow(x,2))+a*b)/Math.Sqrt(Math.Pow(Math.Cos(x+Math.PI/2),2)-b);
```

☞Назад к основному тексту (стр. 14)

4. Условный оператор

4.1. Сложные условия

В задачах могут встречаться ***сложные условия***, в которых два или более простых условия связаны между собой ***логическими операциями***.

Пример сложного условия: покупатель получает скидку на покупку, совершенную до 12 часов дня в любой день, кроме субботы и воскресенья.

Запишем это условие с помощью логических операций. В задаче участвуют две переменные – время покупки (t) и день недели (d). Для наглядности запишем все простые условия, которые должны выполняться:

1) время покупки меньше 12 часов: t<12;

2) день недели не суббота: d != “суббота”;

3) день недели не воскресенье: d != “воскресенье”.

Перечисленные три *простых* условия надо объединить в одно *сложное* логическое выражение. Сделать это можно с помощью специальных логических операций ***И*** В логических выражениях, так же как и в арифметических, можно использовать скобки для указания очередности выполнения. Приведенное выше условие с использованием логических операций будет иметь следующий вид:

```
t<12 И (d = НЕ “суббота” И d = НЕ “воскресенье”);
```

Результат операции *Условие1 И Условие2* равен `истина (true)`, если *оба* условия равны `истина (true)`, в противном случае результат равен `ложь(false)`.

Результат операции *Условие1 ИЛИ Условие2* равен `истина (true)`, если *хотя бы одно* из условий равно `истина (true)`, в противном случае результат равен `ложь(false)`.

Результат операции отрицания *НЕ*, примененный к значению `истина (true)`, дает `ложь(false)`, и наоборот.

Для логических операций *И*, *ИЛИ* и *НЕ* в языке C# существуют свои обозначения[4]: && (И), || (ИЛИ), ! (НЕ).

Логические операции &&, ||, ! применимы только к значениям типа bool. Их результатом также служат величины типа *bool*.

Таблица 4.1. Обозначение логических операций в языке C#

Оператор	Операция	Пример
!	логическое отрицание (НЕ)	F != C;
&&	логическое умножение (И)	F=D && T;
||	логическое сложение (ИЛИ)	F=A || B;

Зависимость результата логической операции от значения входных параметров описывается *таблицей истинности*. Результат работы логических операций (а, следовательно, и таблица истинности) не зависит от языка программирования, а определяется законами алгебры логики.

Таблица 4.2. Таблица истинности логических операций

Операнды	||	&&
true,true	true	true

[4] Существуют и другие формы логических операций. Более подробно можно прочитать в следующей части пособия.

true,false	true	false
false,true	true	false
false,false	false	false

При записи сложных условий с помощью логических операций следует помнить о приоритете выполнения операций. Приоритет операций сравнения ниже, чем арифметических операций. Например, в выражении:

A+B>=C

сначала будет вычислена сумма значений переменных A и B, а затем результат будет сравниваться со значением переменной C.

Приоритет логических операций

При объединении нескольких логических операций нужно помнить, что они, подобно математическим операциям, подчинены правилам приоритета.

Приоритет	Логическая операция
1 (самый высокий)	!= (отрицание)
2	&& (логическое И)
3 (самый низкий)	|| (логическое ИЛИ)

Использование скобок изменяет очередность операций таким же образом, как и в арифметических выражениях, т.е. действия в скобках выполняются в первую очередь. Например, выражение:

c ***//*** d ***&&*** (b= =20);

будет выполняться в следующей последовательности:

1) b= =20

2) d ***&&*** (b= =20);

3) c ***//*** d ***&&*** (b= =20);

Логические выражения вычисляются слева направо, поэтому в выражении

(a>5) || (b>5) ***&&*** (a<20) ***&&*** (b<30);

Сначала будут вычислены выражения в скобках, затем будет выполнена операция && для скобок (b>5) ***&&*** (a<20) ***&&*** (b<30) , затем выполняется логическое ИЛИ (a>5) || (b>5) ***&&*** (a<20) ***&&*** (b<30).

В примере: ***!*** (a<15) ***// !*** (b<30) логическое ИЛИ выполняется последним, после отрицания.

☞Назад к основному тексту (стр. 15)

5. Оператор выбора Switch

В языке C# алгоритм выбора реализуется с помощью оператора `switch`. Оператор ***switch*** – это оператор управления, выбирающий из списка возможных вариантов *раздел переключения* для выполнения содержащегося в нем кода:

```
switch(выражение)
{
        case константное_выражение_1: [список операторов_1]
        …
        case константное_выражение_K: [список операторов _K]
        [default: операторы_N]
}
```

Выполнение оператора `switch` начинается с вычисления значения `switch`-*выражения (значение ключа)*. Затем оно поочередно в порядке следования `case` сравнивается на совпадение с константными выражениями (*метками*). Как только достигнуто совпадение, выполняется соответствующая последовательность операторов `case` ветви. Поскольку последний оператор этой последовательности является оператором перехода (чаще всего это оператор `break`), то обычно он завершает

выполнение оператора switch. Если ни одно выражение case не совпадает со значением оператора switch, управление передается операторам, следующим за необязательной подписью default. Если подписи default нет, то управление передается за пределы оператора switch.
Структурная схема оператора switch в алгоритмах представлена на рисунке 5.1.

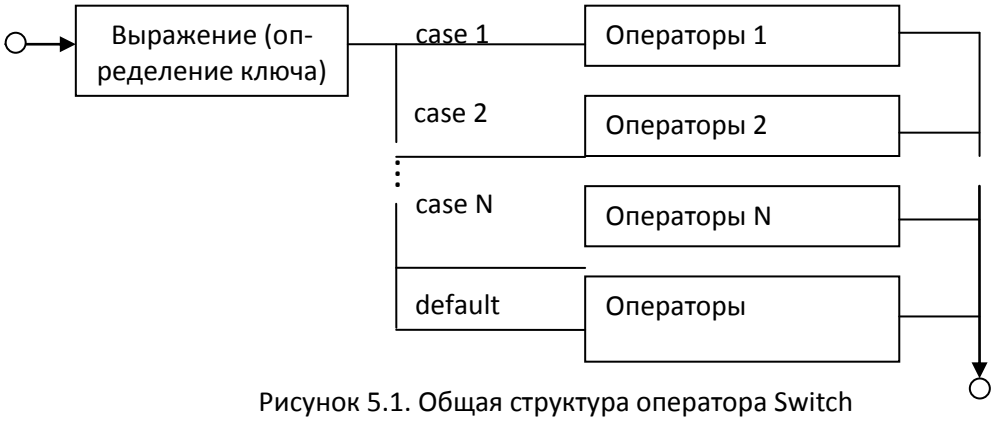

Рисунок 5.1. Общая структура оператора Switch

Можно говорить, что оператор switch передает управление той группе операторов, метка которой совпадает со значением ключа. Следует обратить внимание, что ключ может быть не только числовым значением, но и строковым, символьным, логическим.

В теле оператора switch возможна запись пустых ветвей case, т.е. за ключевым словом case нет последующего оператора, но записана следующая ветвь case. Тогда в случае совпадения константного выражения этой ветви со значением switch-выражения будет выполняться первая непустая последовательность очередной case-ветви.

Например, в коде:

```
Switch (n)
{ case 1:
   case 2: Console.WriteLine(“Ветвь 1”); break;
   case 3:
   case 4: Console.WriteLine(“Ветвь 2”); break; }
```

сообщение “Ветвь 1” будет выведено при **n** равном 1 или 2, а сообщение “Ветвь 2” будет выведено при **n** равном 3 или 4. Такую запись для пустых ветвей в операторе switch называют «проваливанием» значений.

Приведенные далее примеры иллюстрируют возможные особенности использования оператора switch.

Задача 5.1. Составить программу для расчета стоимости заказного междугородного телефонного разговора (стоимость зависит от количества минут соединения и от расстояния до города), а также сообщения кода города. Вспомогательная информация приведена в таблице. Считается, что разговор может быть заказан только в указанные страны.

Страна	**Телефонный код**	**Стоимость 1 минуты**
Беларусь	375	9,50
Украина	380	10,30
Казахстан	77	12,5
Молдова	373	12,1
Чехия	420	17,0
Польша	48	15,0

Решение. Входные данные для решения задачи представляют собой строковую переменную – название города. Эта переменная является значением ключевого выражения. Количество заказанных минут разговора вводится с клавиатуры. Выходными значениями будут: сообщение о телефонном коде города и рассчитанная стоимость разговора. В данной программе после `case` записан блок операторов, заключенных в скобки{ }.

Листинг программы для решения задачи 5.1:

```
static void Main(string[] args)
  {
   Console.WriteLine("Введите название страны: Беларусь, Украина, Казахстан, Молдова,
Чехия, Польша");
    string country = Console.ReadLine();
    Console.WriteLine("Введите продолжительность разговора (в минутах)");
    int t= Convert.ToInt16(Console.ReadLine());
    double res=0; int kod=0;
    switch (country)
    {
      case "Беларусь": res = t * 9.5; kod = 375; Console.WriteLine("Код {0}: {1}, стои-
мость разговора= {2} ", country, kod,res); break;
      case "Украина": res = t * 10.3; kod = 380; Console.WriteLine("Код {0}: {1}, стои-
мость разговора= {2} ", country, kod, res); break;
      case "Казахстан": res = t * 12.5; kod = 77; Console.WriteLine("Код {0}: {1},
стоимость разговора= {2} ", country, kod, res); break;
      case "Молдова": res = t * 12.1; kod = 373; Console.WriteLine("Код {0}: {1}, стои-
мость разговора= {2} ", country, kod, res); break;
      case "Чехия": res = t * 17; kod = 420; Console.WriteLine("Код {0}: {1}, стоимость
разговора= {2} ", country, kod, res); break;
      case "Польша": res = t * 15; kod = 375; Console.WriteLine("Код {0}: {1}, стои-
мость разговора= {2} ", country, kod, res); break;
      default: Console.WriteLine("Пожалуйста, вводите только страны из указанного спи-
ска") ; break;
    }
   Console.ReadKey();
  }
```

Результат работы программы решения задачи 5.1 приведен на рисунке 5.1.

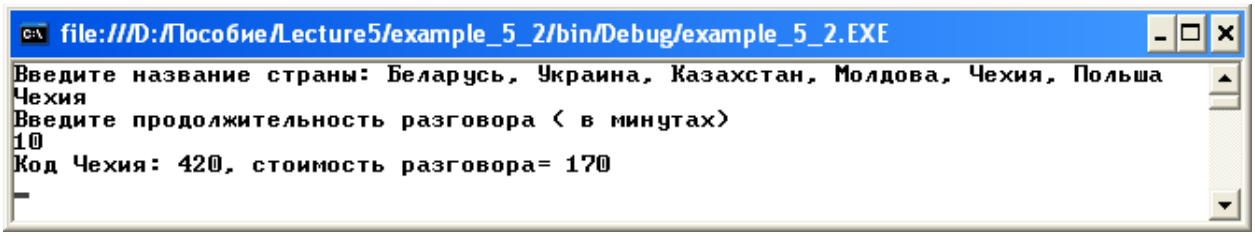

Рисунок 5.1. Результат работы программы для решения задачи 5.2

☞ Назад к основному тексту (стр. 15)

6. Оператор цикла с параметром

Циклы с параметром составляют класс циклов, в которых выполнение повторяющихся операторов (*тела цикла*) должно повторяться заранее определенное число раз с постоянным шагом (*шаг параметра*). Цикл заканчивается, когда параметр принимает определенное значение.

В языке C# для описания такого цикла имеется специальная конструкция:

```
for (заголовок цикла)
 {
  Операторы тела цикла
 }
```

В ***заголовке цикла*** задаются правила изменения параметра. В программировании принято в качестве параметра использовать независимую переменную, которая объявляется и инициализируется в заголовке цикла. Тем самым гарантируется независимость параметра от случайных изменений. Следует избегать изменения параметра в теле цикла. Несмотря на то, что компилятор не будет регистрировать ошибку, может быть нарушена логика работы цикла с параметром. Объявленная в заголовке переменная-параметр является *локальной*, т.е. *область* ее *видимости* (область использования) ограничена телом цикла.

Примеры описания заголовков цикла:

```
1) for (int i=0; i<10, i++)
  {
// тело цикла повторится 10 раз
//параметр увеличивается на 1 за каждую итерацию
  }

2) for (int j=5; j=>0, i-=0.5)
  {
// тело цикла повторится 11 раз
//параметр уменьшается на 0,5 за каждую итерацию
  }
```

При переходе к обработке оператора цикла **for** параметру присваивается заданное начальное значение. Затем в цикле выполняется исполнительный оператор (или *тело цикла*). Каждый раз после выполнения тела цикла управляющая переменная увеличивается на величину шага, проверяется выполнение логического условия, записанного в заголовке (условия окончания цикла). Если условие окончания цикла не достигнуто, итерации тела цикла продолжатся. Цикл завершается при достижении управляющей переменной своего конечного значения.

В качестве параметра можно использовать все типы числовых переменных, включая и вещественные, а также переменные типа `char`.

В том случае, когда тело цикла представлено одним оператором, фигурные скобки для отделения тела цикла можно не ставить.

Примеры описания оператора цикла с параметром:

```
1)   for (int i = 1; i <= 5; i++)
       Console.WriteLine(i);   //тело цикла - всего один
                               //оператор и скобки {} не нужны

2)   for (double j = 3.2+0.6; j > 2.1; j=j-0.3)
      { //в качестве параметра использована вещественная переменная
      Console.WriteLine(j);
      }

3)   for (char i = 'a'; i <= 105; i++)
       Console.Write (i);   //символьная переменная в качестве параметра.
                           // Будут выведены символы abcd
```

☞Назад к основному тексту (стр. 16)