Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

ПЛАТФОРМА .NET FRAMEWORK. ЯЗЫК C#

Учебно-методическое пособие

Часть 1

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Платформа .NET Framework.

Язык C#

Учебно-методическое пособие

В двух частях

Часть 1

Составители:
Д. А. Якубович
Е. С. Еропова

Владимир
Издательство «Шерлок-пресс»
2018

УДК 519.254
ББК 74.202.53
П37

Рецензент: кандидат физико-математических наук, доцент кафедры вычислительной техники и систем управления института информационных технологий и радиоэлектроники Владимирского государственного университета им. Александра Григорьевича и Николая Григорьевича Столетовых (ВлГУ)
А.В. Шутов

П37 Платформа .NET Framework. Язык C#. : учеб.-метод. пособие : Сост.: Якубович Д. А., Еропова Е. С. / Мин-во образования и науки Рос. Федерации, ФГБОУ ВО «Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых». – Владимир : Издательство «Шерлок-пресс», 2018, Ч. 1. – 48 с.
ISBN 978-5-6041042-9-3

Содержит целевые установки, системную информацию для эффективности развития и обучения студентов. В издании представлены 11 тем по основам программирования на языке C# для изучения теоретического курса по дисциплинам «Программирование», «Современные языки программирования», «Информатика». Материал систематизирован и содержит многочисленные примеры.

Предназначено для организации самостоятельной работы студентов и слушателей Педагогического института ВлГУ.

УДК 519.254
ББК 74.202.53

ISBN 978-5-6041042-9-3

ОГЛАВЛЕНИЕ

ЧАСТЬ 1

БАЗОВЫЙ СИНТАКСИС ЯЗЫКА C#

Занятие 1.1 Платформа .NET Framework. Язык программирования C#

В настоящее время у разработчиков весьма широкий спектр выбора средств разработки приложений. Язык программирования может быть ориентирован под определенные типы задач и операционные системы. Например, широко используемый Borland Delphi был изначально предназначен для разработки приложений только под ОС Windows. Ряд языков можно использовать в разных операционных системах. Например, C++ позволяет создавать приложения под Windows, Linux, Android и т.д.

1.1.1 .NET Framework

Усложнение практических задач требует развития комплексных и универсальных платформ разработки ПО. Например, до сих пор популярный и мощный язык программирования C++ используются для реализации как небольших, так и крупных проектов, прекрасно подходит для реализации задач, требующих оптимизацию вычислительных ресурсов. Однако он требует качественной подготовки разработчика, понимания механизмов функционирования операционных систем, способность работать с новыми библиотеками.

Кроме того, появление портативных устройств и развитие веб-технологий потребовало решить вопрос кроссплатформенной работы приложений.

Естественная необходимость – создать программную платформу, которая:

1. защитит разработчика от наиболее опасных проблем;
2. решит (хотя бы частично) проблему переносимости ПО;
3. предоставит разработчику широкие возможности и множество уже готовых решений.

Одной из таких разработок становится технология .NET Framework.

***.NET Framework** («дот Нэт Фрэймворк») – программная платформа компании Microsoft (2002).*

Разработчики преследовали следующие цели:

1. обеспечение совместного использования различных языков программирования под единой программной платформой;
2. безопасность и переносимость программ на платформах ОС Windows.

Для достижения этих целей платформа включает следующие важные элементы:

- **CLR** (Common Language Runtime) – общеязыковая среда выполнения. Предназначена для поддержки многоязычного программирования, переносимости и безопасного выполнения кода.
- **FCL** (Framework Class Library) – стандартная библиотека классов. Допускает использование везде, где установлен пакет .NET Framework.

Важно понимать, что .NET Framework – библиотека, поддерживающая выполнение программ и средство разработки в одном лице!

Начиная с Windows Vista компоненты .NET Framework устанавливаются в пакет операционной системы Windows по умолчанию. Обычно это системная папка, находящаяся по адресу

```
C:\WINDOWS\Microsoft.NET\
```

Платформа .NET Framework преодолела большой путь развития. Например, она предоставляет простые и понятные инструменты для организации многопоточных приложений и многоядерных вычислений (см рис. 1.1.):

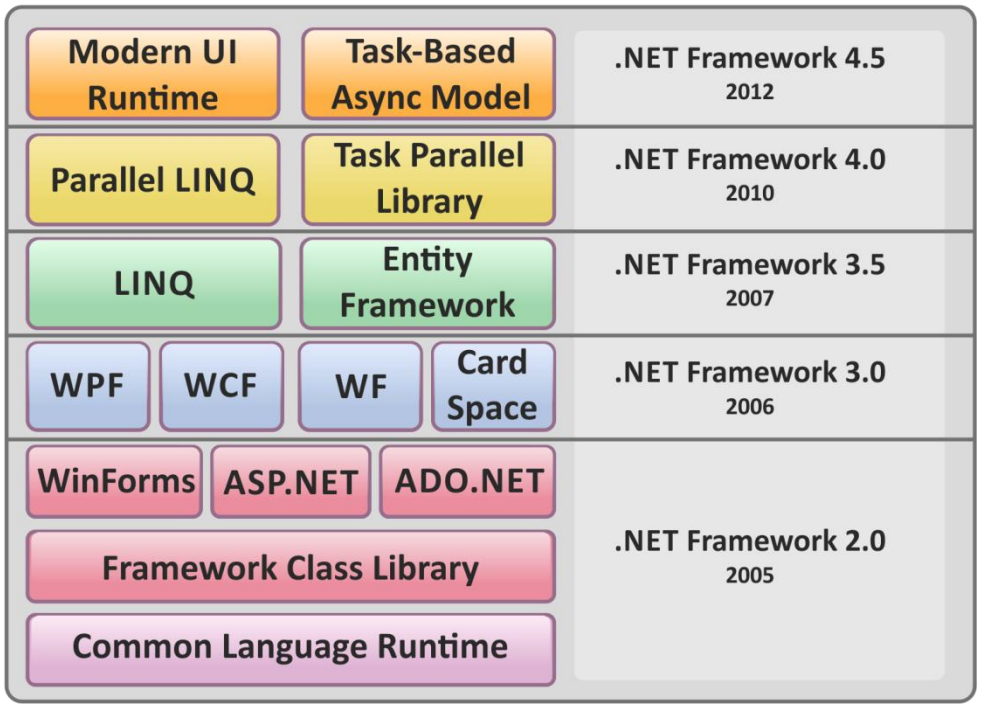


Рис. 1.1. Развитие платформы .NET Framework.

1.1.2 Язык C#. Краткая предыстория

Платформа .NET Framework поддерживает работу нескольких языков программирование. Наибольшую популярность получил язык C#.

C# *(«Си шарп») – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework.*

Новый язык программирования унаследовал синтаксис языка C («Си»):

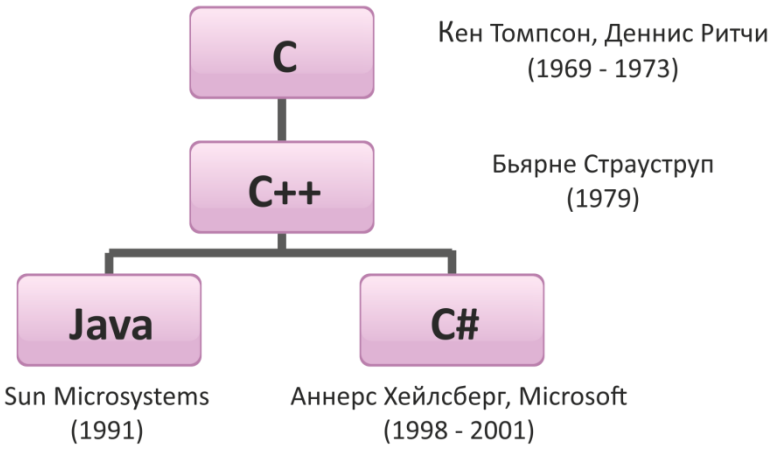


Рис. 1.2. История развития синтаксиса C#.

1.1.3 C# – объектно-ориентированный язык

В программировании сложились различные парадигмы. Они определяются, прежде всего, задачами, которые можно решить выбранным языком. Среди них широкое распространение получили **процедурные** языки программирования. Здесь программа разбита на модули, называемые **подпрограммами** (**процедурами** или **функциями**). Они избавляют нас от повторного написания кода и лучше отражают структуру программы. К таким языкам относятся, например, Си, Basic, Pascal, Фортран.

Однако процедурная парадигма существенно ограничивает возможности абстрагирования. Может потребоваться описание некоторого класса объектов, наделенных одинаковыми свойствами и функциями, но с разными значениями. У процедурной парадигмы, несмотря на ее достоинства, просто нет эффективных возможностей для описания таких задач.

Очевидно, что в программировании потребовалось нечто большее, то, что является объектом, представителем класса однородных объектов. Так возникли объектно-ориентированные языки программирования.

Объектно-ориентированное программирование *(в дальнейшем* ***ООП****) – парадигма программирования, в которой основными концепциями являются понятия* ***объектов*** *и* ***классов****.*

ООП держится на «трех китах» – принципах ООП:

1. инкапсуляция;
2. наследование;
3. полиморфизм.

Инкапсуляция предполагает, что объекты защищены от нежелательного вмешательства, упакованы в единый компонент.

Наследование позволяет создавать новые классы объектов, обладающие возможностями «родителей» и приобретающих новые свойства.

Полиморфизм предполагает, что схожие объекты могут иметь разные формы и реализацию, переопределять базовые возможности.

Кроме того, ключевым является понятие **класса** – абстракции, схемы описание объектов или процессов.

Важно заметить, что парадигма ООП включает процедурную, но реализована в более эффективной и многофункциональной форме.

1.1.4 Средства разработки

Программа – это не только программный код. Необходим целый комплекс мероприятий, который превратит вашу программу в полноценное приложение.

У программиста есть два пути разработки программы:

1. с помощь интегрированной среды разработки;
2. вручную, самостоятельно собирая программу из разных компонент.

Оба способа имеют свои достоинства и недостатки. Среда разработки – «посредник» между программистом и .NET Framework; она берет на себя самую рутинную работу, а программист лишь пишет программный код и конструирует визуальный интерфейс приложения. Ручная сборка полезна программисту, поскольку требует подробного изучения особенностей работы с языком программирования, а также дает независимость от конкретных сред разработок. Мы используем первый подход, поскольку он обычно не требует специальных знаний.

1.1.5 Visual Studio

Наиболее мощной средой разработки на C# является **Visual Studio** (компания Microsoft). Она предоставляет множество эффективных возможностей: редактор с подсветкой синтаксиса команд, интеллектуальные всплывающие подсказки, автоматическая сборка, инструменты отладки, анализаторы производительности и др.

Следует отметить, что Visual Studio – проприетарный продукт. Впрочем, есть специальные условия, по которым можно получить бесплатный доступ к пакету (например, студентам), если вы не используете его в корпоративных целях. Также доступна бесплатно распространяемые Visual Studio Express, Visual Studio Community. Это урезанные версии Visual Studio, но для обучения и разработки приложений это прекрасное решение.

1.1.6 SharpDevelop

Это свободно распространяемая среда разработки. В отличии от Visual Studio не требует большого ресурса ПК, а также очень легко превращается в portable-версию: достаточно скопировать установленные файлы, например, на флеш-накопитель, и мы имеем независимую переносную среду разработки.

1.1.7 Online проекты

Стремительное развитие сети Интернет приводит к новому подходу – облачным вычислениям. Эта технология пока еще во многом уступает стационарным средствам разработки, однако очевидно преимущество ее будущего – возможность программировать в режиме online и без установки специального ПО. Все что необходимо – доступ в сеть Интернет.

Перечислим ряд наиболее удобных в использовании сервисов:

- https://dotnetfiddle.net/
- http://rextester.com/runcode
- http://ideone.com/
- http://www.tutorialspoint.com/

Занятие 1.2 Структура программы

1.2.1 Первая программа

Создадим консольное приложение со следующим кодом:

Листинг 1.1 Первая программа

```
/*  Моя первая программа  */
using System;                       // Подключение пространства
имен System

namespace First_Program             // Пространство имен
{
    class Program                           // Основной класс
    {
        static void Main()                  // Точка входа
        {
            Console.Write("Привет, мир!");   // Вывод сообщения
            Console.ReadKey();      // Ожидание нажатия клавиши
        }
    }
}
```

1.2.2 Использование Visual Studio

Запустите Visual Studio. Далее *Файл / Создать / Проект*:

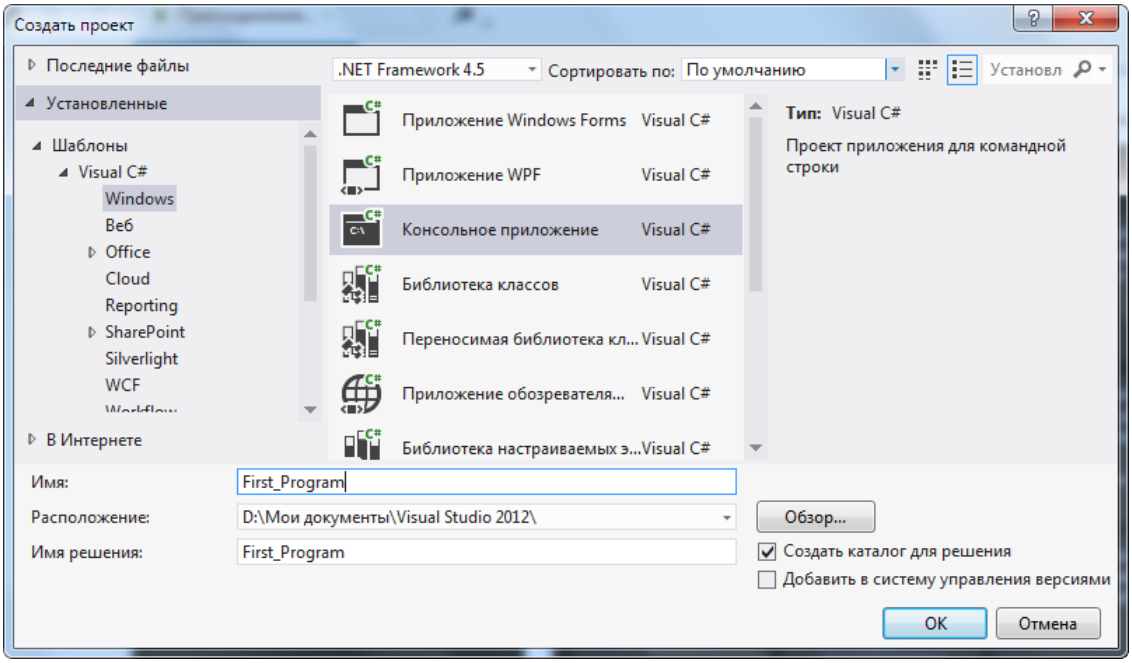


Рис. 1.3. Окно создания нового проекта в Visual Studio.

Выбираем «*Консольное приложение*», задаем имя и меняем при необходимости каталог, в котором сохраняются файлы проекта. Система сформирует шаблон кода:

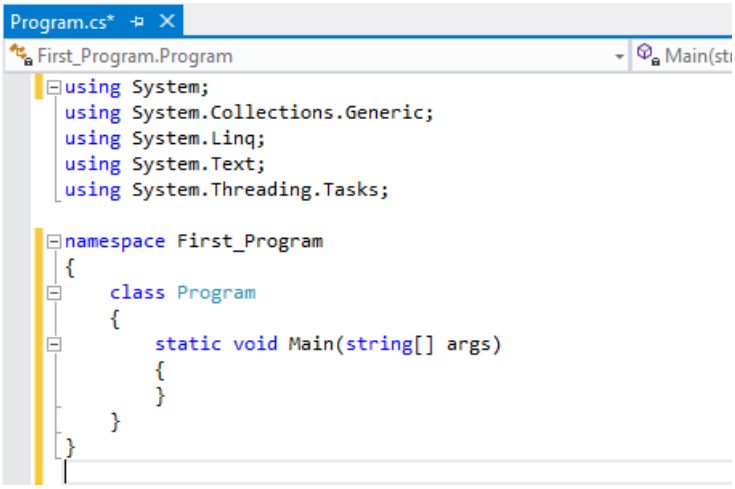


Рис. 1.4. Стандартный шаблон консольного приложения.

Стандартный шаблон немного отличается от нашего, поправим его:

```
Program.cs
First_Program.Program                                   Main()
/*  Моя первая программа  */
using System;                          // Подключение пространства имен System

namespace First_Program                            // Пространство имен
{
    class Program                                  // Основной класс
    {
        static void Main()                         // Точка входа
        {
            Console.Write("Привет, мир!");         // Вывод сообщения
            Console.ReadKey();                     // Ожидание нажатия клавиши
        }
    }
}
```

Рис. 1.5. Код первого примера «Привет, Мир!».

Для компиляции и запуска приложения нажмите на кнопку «*Запуск*»:

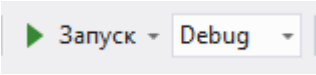


Рис. 1.6. Панель для запуска компиляции проекта.

Если возникнут ошибки, система Visual Studio их отобразит, иначе запускается консоль программы:

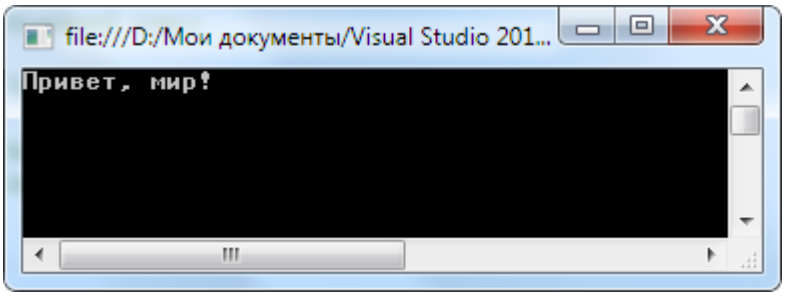


Рис. 1.7. Консоль приложения.

Теперь убедимся, что создано вполне рабочее приложение. Для этого необходимо найти .exe файл приложения; он находится в каталоге проекта (может быть другим):

```
D:\Мои документы\Visual Studio
2012\First_Program\First_Program\bin\Debug
```

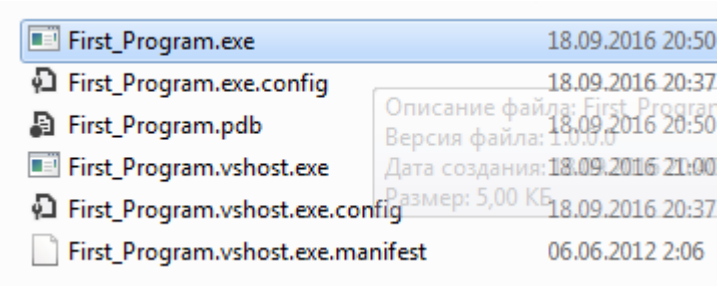


Рис. 1.8. Служебные файлы проекта.

1.2.3 Использование SharpDevelop

Разработка проекта в SharpDevelop похожа на работу с Visual Studio. Создание нового проекта осуществляется командой «*Файл / Создать / Решение*»:

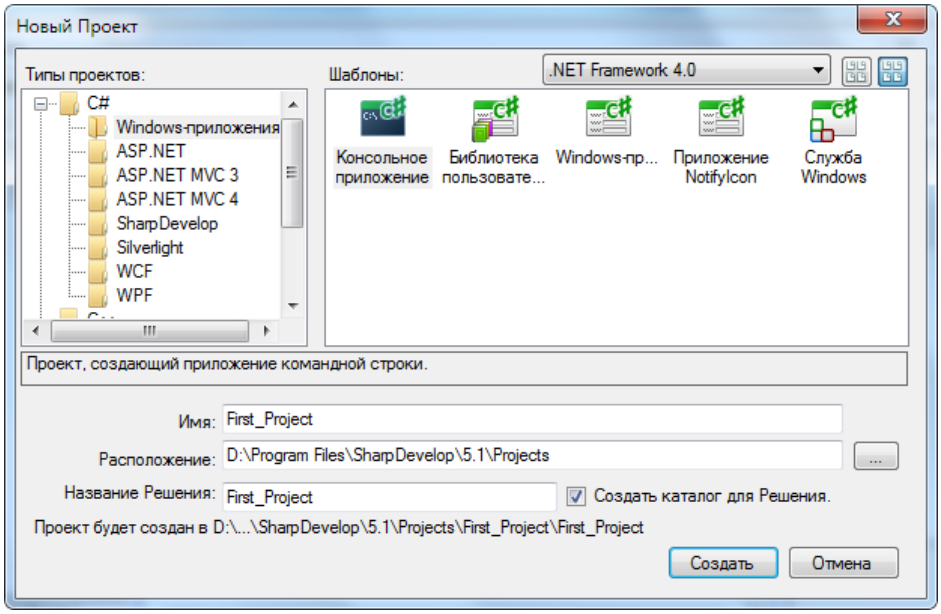


Рис. 1.9. Окно создания нового проекта в SharpDevelop.

Аналогично удалим код шаблона и вставим наш:

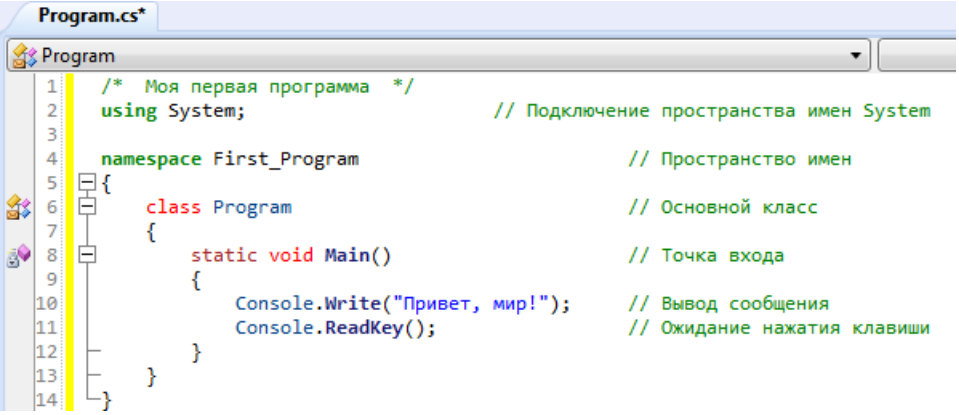


Рис. 1.10. Редактор кода SharpDevelop.

Запуск осуществляется по нажатию на

Рис. 1.11. Кнопка запуска и компиляции.

Важно отметить, что в отличии от Visual Studio, SharpDevelop не отображает ошибки во время набора кода: только в процессе компиляции.

1.2.4 Использование веб-сервисов

Наконец, вы можете поэкспериментировать с онлайн-компиляторами: Перечислим ряд наиболее удобных в использовании сервисов:

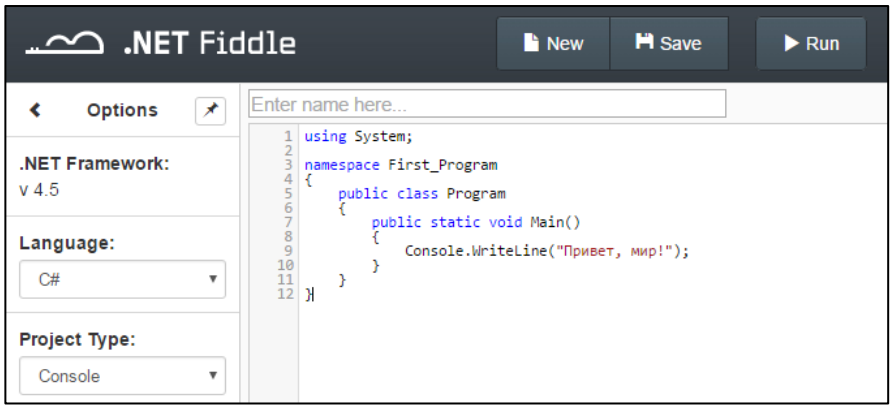


Рис. 1.12. Веб-сервис .NET Fiddle.

1.2.5 Описание команд

Общее пространство кода разделено на **пространства имен**. Внутри каждого пространства имен содержатся **классы** (рис. 1.13.). Пространства имен необходимы, поскольку в программе могут использоваться классы с одинаковым названием, но разной реализацией.

Рис. 1.13. Иерархия пространств имен и классов.

В программе пространство имен определяется командой **namespace**. Имя мы вправе задать самостоятельно (First_Program):

```
namespace First_Program
```

Пара фигурных скобок **{ }** определяет начало и конец блока. Многие конструкции языка C# используют их.

Каждая команда должна завершаться оператором **;**. Исключение – операторные скобки { } некоторого блока.

Класс – составная структура данных и наиболее важный компонент .NET Framework. В полной форме они содержит две части:

- **поля** (количественно-качественная характеристика);
- **методы** (функциональные возможности).

В любой программе C# есть хотя бы один класс. В нашем случае он один, имя также мы также вправе определить самостоятельно:

```
class Program
```

Внутри класса расположен метод **Main**. Он обязателен в любой программе, поскольку является точкой входа в приложение:

```
static void Main()
```

С помощью команды // задается однострочный комментарий. Если необходимо закомментировать несколько строк кода, можно окружить его операторами /* и */.

Команда

```
Console.Write("Привет, мир!");
```

выводит указанный текст: здесь используется **метод** Write класса Console.

Последняя команда Console.ReadKey( ) ожидает нажатия любой клавиши. Это необходимо, чтобы консольное приложение не закрывалось сразу после выполнения.

C# чувствителен к регистру букв! Например, идентификатор Main и main он будет понимать как две разные команды.

Занятие 1.3 Типы данных. Переменные. Арифметические операции

1.3.1 Типы данных в C#

C# – строго типизированный язык.
Это означает, что компилятор не позволит работать с переменной до тех пор, пока не задан ее тип (т.е. переменная не описана).

В C# различают две категории типов данных:

- значимые типы;
- ссылочные типы.

Отличие этих типов заключается в содержимом переменной. Так, переменная значимого типа содержит само значение (например, целочисленная содержит целое число). Переменная ссылочного типа содержит не значение, а ссылку на него. Наиболее ярким примером ссылочного типа является **класс**.

Не менее важным является принцип работы с переменными значимого и ссылочного типа. Так, для переменных значимого типа память выделяется еще на этапе компиляции приложения, что влияет на его размер. Для ссылочных – во время непосредственной его работы, по мере востребованности (ее называют **динамической** памятью).

На рис. 1.14. представлено дерево типов (в скобках указан размер типа в байтах). Отметим, что указанные названия типов являются сокращениями, действующими только для языка C#: вместо них можно брать полное название типов .NET Framework.

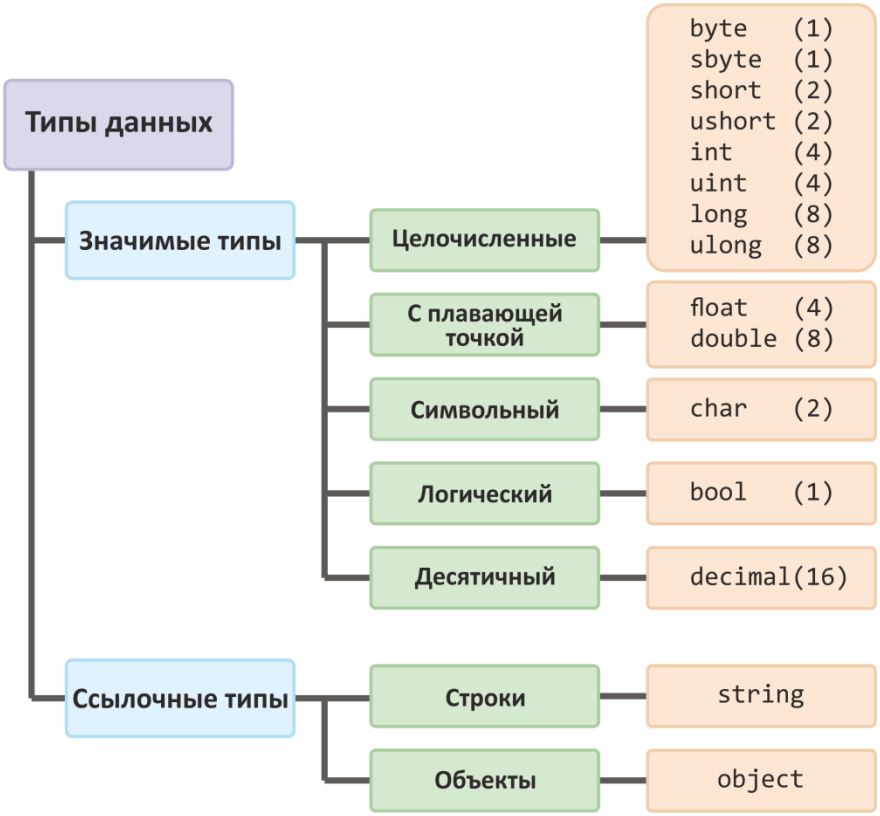


Рис. 1.14. Схема типов данных языка C#.

1.3.2 Целочисленные типы

Тип	Разрядность в байтах	Диапазон
byte	1	0..255
sbyte	1	-128..127
short	2	-32768..32767
ushort	2	0..65535
int	4	-2147483648..2147483647
uint	4	0..4294967295
long	8	-9223372036854775808..9223372036854775807
ulong	8	0..18446744073709551615

1.3.3 Вещественные типы

Тип	Разрядность в байтах	Диапазон
float	4	$\pm 1{,}5 \cdot 10^{-45}$ .. $\pm 3{,}4 \cdot 10^{38}$
double	8	$\pm 5 \cdot 10^{-324}$ .. $\pm 1{,}7 \cdot 10^{308}$

1.3.4 Логический тип

Тип	Разрядность в байтах	Диапазон
bool	1	true либо false

1.3.5 Символьный тип

Тип	Разрядность в байтах	Диапазон
char	2	0..65535

1.3.6 Десятичный тип

Тип	Разрядность в байтах	Диапазон
decimal	16	Порядка $\pm(10^{-28}; 10^{28})$; Точность 28-29 значащих цифр дробной части.

Это особый тип представления чисел с плавающей точкой. Его диапазон меньше, чем у типа double, однако выше точность представления знаков в дробной части, что необходимо для реализации финансовых операций.

1.3.7 Строковый тип

Тип	Разрядность в байтах	Диапазон
string	2 на символ	Размер определяется числом символов строки.

Вообще говоря, тип string не является «простым» с точки зрения .NET. Переменные этого типа обладают свойствами и методами, с ними можно работать как с массивами.

1.3.8 Описание переменных

Перед использованием переменную необходимо описать, т.е. задать ей тип.
Переменная описывается следующим образом:

```
тип   имя_переменной;
```

Например,

```
int n;              // целочисленная переменная n типа int
sbyte tiny;         // целочисленная переменная tiny типа sbyte
double x;           // вещественная переменная x типа double
short A1, A2;       // целочисленные переменные A1, A2 типа short
float C_X, C_Y;     // вещественные переменные C_X и C_Y типа float
```

Переменную можно описывать в любом месте кода, там, где она понадобилась, но только один раз[1].

1.3.9 Оператор присваивания

Оператором присваивания в C# является знак «=»:

```
имя_переменной = значение;
```

Например,

```
n = 100;        // переменная n должна быть описана выше
x = 28.04;      // переменная x должна быть описана выше
byte M = 12;    // описание переменной и ее инициализация значением
```

[1] Здесь необходимо поправить, что речь идет об области видимости переменной. Переменные вполне могут иметь одинаковые имена, если они находятся в разных областях видимости.

C# не будет работать с переменной, если ей не присвоено значение!

Задача 1. Вычислить площадь кольца по заданным внешнему и внутреннему радиусам.

Решение.

Площадь кольца равна разности между площадью внешнего и внутреннего круга:

$$S = \pi R^2 - \pi r^2 = \pi(R^2 - r^2)$$

Листинг 1.2

```
using System;

class First
{
    static void Main()
    {
        double R = 1;           // внешний радиус
        double r = 0.5;         // внутренний радиус
        double S = Math.PI * (R * R - r * r);  // площадь
        Console.Write(S);       // вывод результата
        Console.ReadKey();
    }
}
```

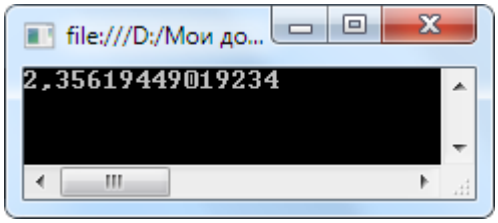


Рис. 1.15. Результат работы приложения.

1.3.10 Арифметические операции

+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления

Задача 2. Описать переменные a, b, c. В переменную sred записать среднее арифметическое этих чисел и вывести его на экран.

Решение.

Листинг 1.3

```
using System;
```

```
class First
{
    static void Main()
    {
        double a = 2;
        double b = 5;
        double c = 1;
        double sred = (a + b + c) / 3;
        Console.Write("Сренее равно " + sred);
        Console.ReadKey();
    }
}
```

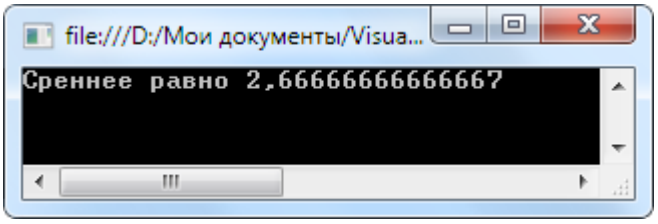


Рис. 1.16. Результат работы приложения.

Занятие 1.4 Ввод и вывод данных

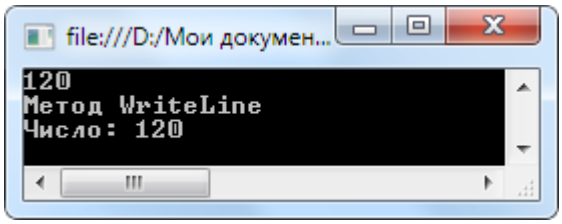
1.4.1 Методы Write и WriteLine

Для организации ввода и вывода информации в консоли используется класс **Console**.

Методы **Write** и **WriteLine** обладают одинаковыми возможностями. Отличие между ними лишь в том, что второй дополнительно переводит курсор на новую строку, а первый обязательно имеет параметр.

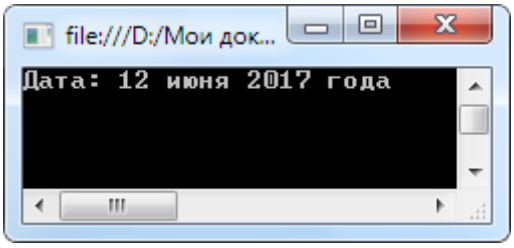
В качестве аргументов им можно передавать строковые и числовые данные напрямую, либо через переменные:

```
int number = 120;
string text = "Метод WriteLine";
Console.WriteLine(number);
Console.WriteLine(text);
Console.WriteLine("Число: " + number);
```



Оператор **+** склеивает параметры в единую текстовую строку.

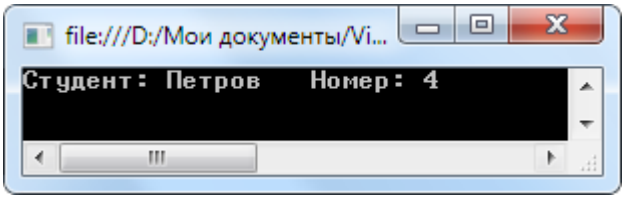
```
byte day = 12;
string month = "июня";
int year = 2017;
Console.WriteLine("Дата: " + day + " " + month + " " + year + "
года");
```



1.4.2 Форматный вывод

С помощью форматного вывода значения можно вставлять в определенное место строки. Для этого укажите это место фигурными скобками и индексом элемента (индексация параметров ведется с нуля):

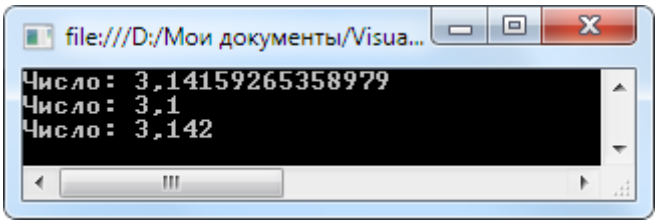
```
Console.Write("Студент: {0}   Номер: {1}", "Петров", 4);
```



Так вместо {0} в строку будет подставлено «Петров», а вместо {1} – число 4. (Индексация с нуля.)

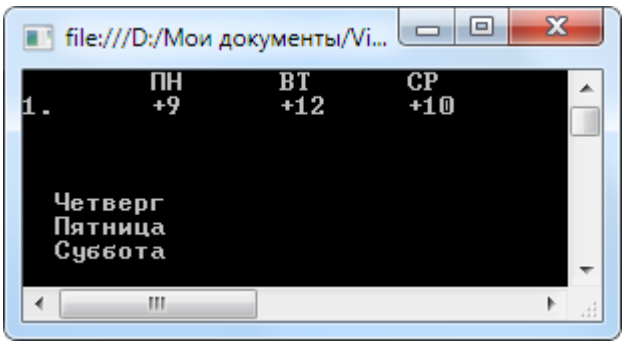
Этот прием можно использовать для форматного вывода вещественных чисел:

```
double pi = Math.PI;
Console.WriteLine("Число: " + pi);
Console.WriteLine("Число: {0:#.#}", pi);
Console.WriteLine("Число: {0:#.###}", pi);
```



В строку также можно включать специальные форматирующие символы. Чаще всего используют \n (переводит курсор на новую строку) и \t (выставляет табуляцию между соседними значениями):

```
Console.WriteLine("\tПН\tВТ\tСР");
Console.WriteLine("1.\t+9\t+12\t+10");
Console.WriteLine("\n\n\n  Четверг\n  Пятница\n  Суббота");
```



1.4.3 Форматный вывод: управляющие последовательности:

Вид	Описание
\a	Звуковой сигнал
\b	Возврат на шаг назад
\f	Перевод страницы
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Обратная косая черта
\’	Апостроф
\”	Кавычки

1.4.4 Форматный вывод: размер поля вывода

Элемент строки вида **{n, m}**, где n определяет номер идентификатора из списка аргументов, а m – количество отображаемых символов (размер поля вывода). Значение выравнивается по правому краю.

Если выделенных позиций для размещения значения идентификатора окажется недостаточно, то автоматически добавиться необходимое количество позиций.

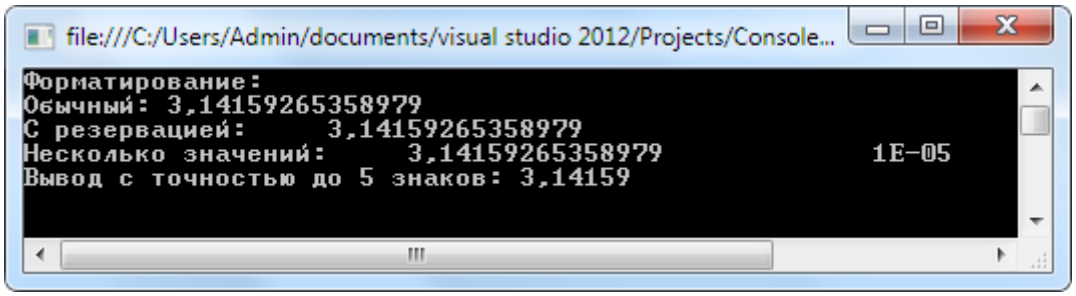
1.4.5 Форматный вывод: вещественные числа

Строка вида **{n:##.###}** – где n определяет номер идентификатора из списка аргументов метода, а ##.### задает формат вывода вещественного числа. Символ # означает один соответствующий разряд (относительно точки дробной части).

Если выделенных позиций для размещения целой части значения идентификатора окажется недостаточно, то автоматически добавиться необходимое количество позиций.

```
const double PI = Math.PI;
double err = 0.00001;
```

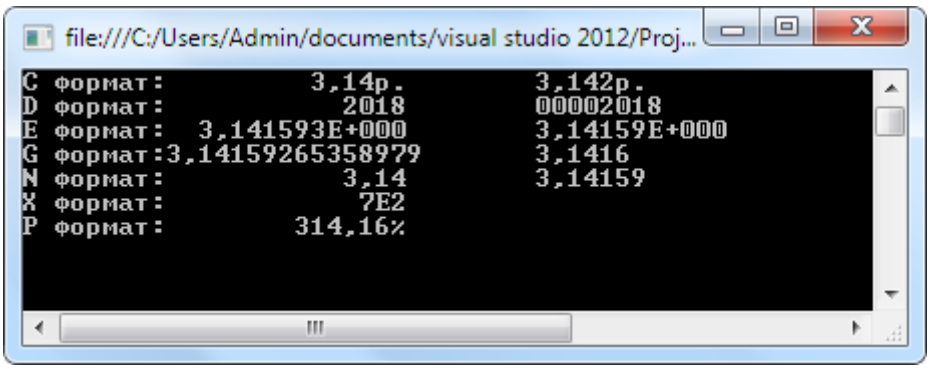
```
Console.Write("Форматирование:\n");
Console.WriteLine("Обычный: " + PI);
Console.WriteLine("С резервацией: {0,20}", PI);
Console.WriteLine("Несколько значений: {0,20} \t{1,10}", PI, err);

Console.WriteLine("Вывод с точностью до 5 знаков: {0:#.#####}",
PI);
```



1.4.6 Форматный вывод: формат числовых данных

Параметр	Формат	Значение
C или c	Денежный. По умолчанию ставит знак р.	Задается количество десятичных разрядов.
D или d	Целочисленный (используется только с целыми числами).	Задается минимальное количество цифр. При необходимости результат дополняется начальными нулями.
E или e	Экспоненциальное представление чисел.	Задается количество символов после запятой. По умолчанию – 6.
F или f	Представление чисел с фиксированной точкой.	Задается количество символов после запятой.
G или g	Общий формат (или экспоненциальный, или с фиксированной точкой).	Задается количество символов после запятой. По умолчанию выводится целая часть.
N или n	Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами.	Задается количество символов после запятой. По умолчанию – 2, если число целое, то ставятся нули.
X или x	Шестнадцатеричный формат.	
P или p	Процентный.	

```
const double PI = Math.PI;
Console.WriteLine("C формат:{0,15:C} \t{0:C3}", PI);
Console.WriteLine("D формат:{0,15:D} \t{0:D8}", 2018);
Console.WriteLine("E формат:{0,15:E} \t{0:E5}", PI);
Console.WriteLine("G формат:{0,15:G} \t{0:G5}", PI);
Console.WriteLine("N формат:{0,15:N} \t{0:N5}", PI);
Console.WriteLine("X формат:{0,15:X} ", 2018);
Console.WriteLine("P формат:{0,15:P} ", PI);
```



1.4.7 Методы ReadLine и Parse

Для ввода данных с клавиатуры используется метод **ReadLine** класса Console.

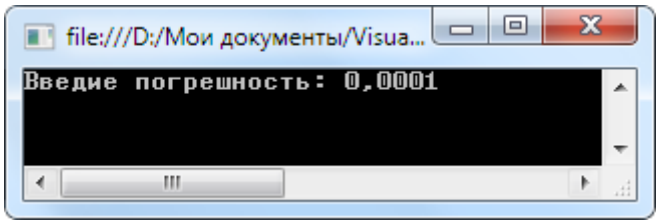
Важно понимать, что C# воспринимает вводимый текст как поток символов (т.е. типа string). Соответственно, для ввода числа необходимо конвертировать информацию в число.

Для преобразования используются специальные методы, привязанные к каждому «простому» типу данных. Метод **Parse** осуществляет преобразование строки к требуемому типу:

```
// конвертировать не нужно
string text = Console.ReadLine();
// конвертируем введенное в целое число
int num = int.Parse(Console.ReadLine());
// конвертируем введенное в вещественое число
double x = double.Parse(Console.ReadLine());
```

Ввод данных необходимо сопровождать поясняющим комментарием, иначе пользователь увидит только мигающий курсор:

```
Console.Write("Введите погрешность: ");
double err = double.Parse(Console.ReadLine());
```



Занятие 1.5 Операции условного выбора. Оператор if

1.5.1 Оператор if

Оператор if в зависимости от истинности условия выполняет соответствующий блок кода:

```
if(условие)
{
    // делаем, когда истина
}
else
{
    // делаем, когда ложь
}
```

Если в блоке только одна команда, то скобки { } можно опустить.

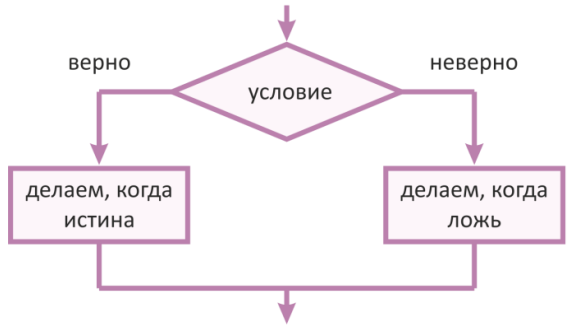
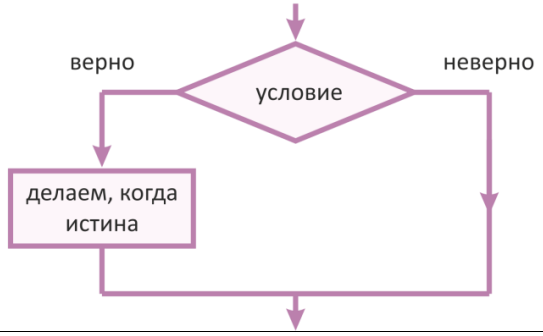
Условие в операторе является **выражением логического типа** (bool), т.е. оно может быть либо истинным, либо ложным.

Круглые скобки в условии являются частью оператора if, их нельзя игнорировать! А фигурные скобки можно не писать, если в блоке один оператор.

Указанная в определении форма оператора if является **полной**. Однако допустима **неполная** форма: блок else отсутствует и в случае ложности условия выполняется следующая за if команда.

Полная форма if	Неполная форма if

`if (условие)`
`{`
`  // делаем, когда истина`
`}`
`else`
`{`
`  // делаем, когда ложь`
`}`
	`if (условие)`
	`{`
	`  // делаем, когда истина`
	`}`

Задача 1. Определить, является или нет текущий год високосным? Год ввести с клавиатуры.

Решение.

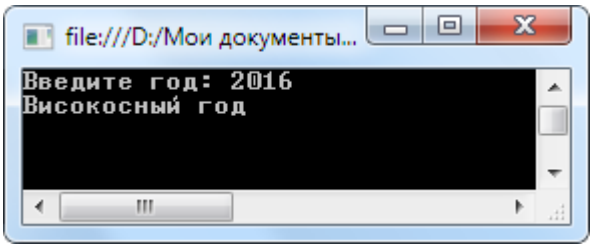
Год является високосным, если он кратен 4:

Листинг 1.4

```
using System;

class First
{
    static void Main()
    {
        // текстовая подсказка
        Console.Write("Введите год: ");
        // ввод данных
        int year = Convert.ToInt32(Console.ReadLine());

        // Если остаток от деления на 4 равен 0, то год високосный
        if (year % 4 == 0)
        {
            Console.WriteLine("Високосный год");
        }
        else
        {
            Console.WriteLine("Обычный год");
        }

        Console.ReadKey();
    }
}
```

Т.к. в каждом блоке по одной команде, то скобки можно опустить и написать так:

```
if (year%4 == 0)
    Console.WriteLine("Високосный год");
else
    Console.WriteLine("Обычный год");
```



Для проверки на равенство используется оператор **==**. Этот оператор называют **оператором отношения**.

1.5.2 Операторы отношения

==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

1.5.3 Логические операторы

Команда	Союз	Описание
&	И	Дает истину, когда истины оба условия
|	ИЛИ	Дает истину в случае истинности хотя бы одного из условий
!	НЕ	Берет противоположное логическое значение
^	Исключающее ИЛИ	Дает истину, когда истина – либо первое, либо второе, но не оба вместе

Задача 2. Определить, входит и цена выбранного ноутбука в диапазон от 20 до 30 тыс. р.

Решение.

Листинг 1.5

```
using System;

class Program
{
    static void Main()
    {
        double price = 25999;   // цена выбранного ноутбука

        // проверяем оба условия
        if (price >= 20000 & price <= 30000)
            Console.WriteLine("Подходит");
        else
            Console.WriteLine("Не подходит");

        Console.ReadKey();
    }
}
```

Занятие 1.6 Оператор if-else-if. Оператор switch

1.6.1 Конструкция if-else-if

Конструкция if-else-if организует последовательный много-критериальный выбор:

```
if (условие_1)
{
    // условие 1 истинно
}
else if (условие_2)
{
    // условие 2 истинно
}
. . .
else
{
    // все условия ложны
}
```

Блок else является необязательным.

На рис. 1.17. изображена блок-схема конструкции if-else-if.

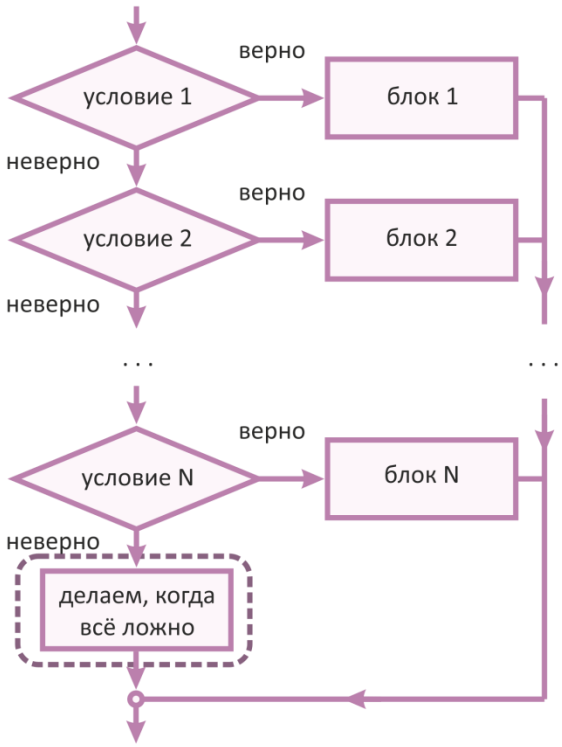


Рис. 1.17. Блок-схема оператора выбора if-else-if.

Задача 1. Написать программу, которая в зависимости от заданного балла определяет оценку учащегося согласно таблице:

0-60	неудовлетворительно
60-74	удовлетворительно
75-89	хорошо
90-100	отлично

Решение.

Наиболее оптимальная стратегия проверки – от наибольшего к наименьшему (либо обратно).

Листинг 1.6

```
using System;

class Program
{
    static void Main()
    {
        byte point = 87;          // баллы
        string mark;              // итоговая оценка

        if (point >= 90)
            mark = "отлично";
        else if (point >= 75)
            mark = "хорошо";
        else if (point >= 61)
            mark = "удовл.";
        else
            mark = "неуд.";

        Console.Write(mark);      // вывод оценки

        Console.ReadKey();
    }
}
```

1.6.2 Оператор switch

Оператор switch в зависимости от значения выражения осуществляет выполнение соответствующего блока кода (аналог Case):

```
switch(выражение)
{
    case значение_1:
        // блок операторов 1
        break;
    case значение_2:
        // блок операторов 2
        break;
    . . .
    default:
```

```
                // блок операторов N
                break;
        }
```

Блок default является необязательным.

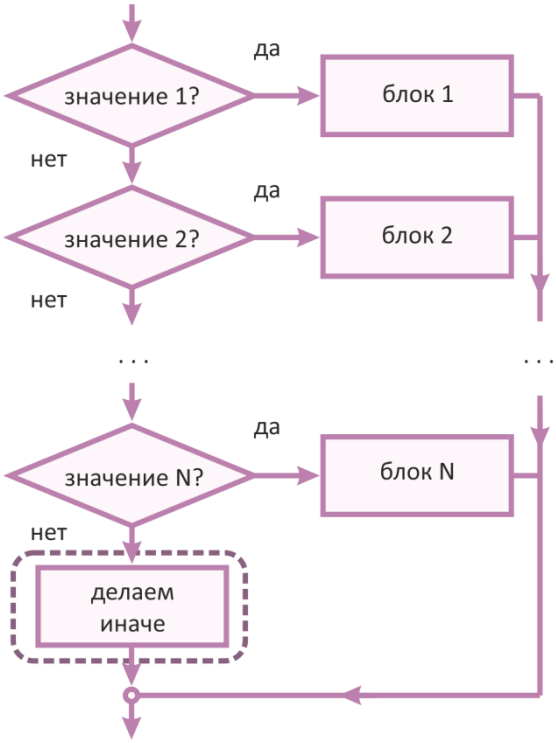


Рис. 1.18. Блок-схема оператора выбора switch.

Заданное выражение в операторе switch должно быть *целочисленного типа* (char, byte, short или int), *перечислимого* или же *строкового*. Команда **break** осуществляет выход из блока[2]. Фигурные скобки для блока в несколько команд не нужны, т.к. компилятор видит расположение команд меду оператором case и break.

Задача 2. Светофор закодирован тремя состояниями: 1 – красный, 2 – желтый, 3 – зеленый. В зависимости от состояния определить цвет светофора.

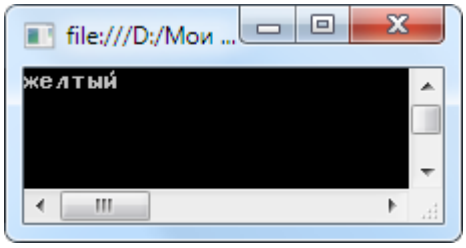
Решение.

Листинг 1.7

```
using System;

class Program
{
    static void Main()
    {
```

[2] Синтаксис оператора switch унаследован от языка C++. Там наличие оператора break в конце блока очень важно. Если его опустить, то осуществляется переход к следующему блоку case без проверки значения. В C# отсутствие оператора break является ошибкой синтаксиса. Однако его может заменить оператор return.

```
            byte color = 2;

            switch (color)  // выбор по значению переменной color
            {
                case 1:                          // color = 1?
                    Console.Write("красный");
                    break;

                case 2:                          // color = 2?
                    Console.Write("желтый");
                    break;

                case 3:                          // color = 3?
                    Console.Write("зеленый");
                    break;

                default:                         // иначе
                    Console.Write("что то пошло не так...");
                    break;
            }

            Console.ReadKey();
        }
    }
```



1.6.3 Синтаксический сахар

Арифметические операторы имеют укороченные аналоги.
Первый набор таких команд – операторы **инкремента** и **декремента**:

++	увеличить на 1
--	уменьшить на 1

Например, следующие команды эквивалентны:

```
i++; или ++i;          // равносильно i = i + 1;
i--; или --i;          // равносильно i = i - 1;
```

Когда оператор пишут слева от переменной, то его называют *префиксным*, справа – *постфиксным*.

Часто возникает необходимость увеличить (уменьшить, умножить, поделить) переменную на некоторое значение или выражение и результат записать в нее же. Например

```
int Sum = 0;
Sum = Sum + 30;        // увеличить Sum на 30
```

Для подобных операций есть сокращенные аналоги:

+=	увеличить на . . .
-=	уменьшить на . . .
*=	умножить на . . .
/=	поделить на . . .

В этом случае код можно переписать так:

```
int Sum = 0;
Sum += 30;             // увеличить Sum на 30
```

1.6.4 Оптимизация проверки условий

Предположим, что мы проверяем 10 условий и истина достигается лишь тогда, когда истинны все условия:

```
if(условие_1 & условие_2 & условие_3 & . . .)
```

Если условие_1 ложно, то все выражение дает ложный результат, проверять остальные 9 смысла уже нет. Если же оно истинно, то проверяется условие_2. В случае ложности опять все условие ложно и т.д. Рационально завершить на этом проверку.

Именно для таких целей были разработаны укороченные логические операторы И и ИЛИ:

&&	проверять до первого ложного условия
||	проверять до первого истинного условия

При использовании укороченных операторов на первые места ставьте наиболее важные условия.

Занятие 1.7 Циклические операции. Оператор for

1.7.1 Циклические структуры

В языке C# существует четыре циклических оператора, которые можно использовать в зависимости от контекста задачи. Три из них взаимозаменяемы.

1.7.2 Цикл for

Оператор for реализует цикл со счетчиком.
В полной форме оператор имеет синтаксис:

```
for (нач_знач; условие; итерация)
{
    // последовательность операторов;
}
```

где

- *нач_знач – начальное значение счетчика;*
- *условие – логическое выражение, указывающее условие работы цикла;*
- *итерация – шаг увеличения / уменьшения счетчика.*

Блок-схема цикла:

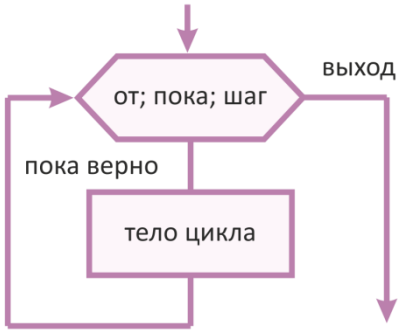


Рис. 1.19. Блок-схема оператора цикла for.

Выполнение цикла for продолжается до тех пор, пока проверка условия дает истинный результат. Прежде всего, в условии может участвовать сама переменная-счетчик. Счетчик цикла можно как увеличивать, так и уменьшать (в зависимости от направления цикла), причем на произвольный шаг.

Задача 1. Найти сумму чисел, кратных 5, в пределах от 0 до 1000.
Решение.

Листинг 1.8

```
using System;

class Program
{
    static void Main()
    {
        int i;                  // счетчик
        int Sum = 0;            // сумма (изначально равна нулю)

        // перебираем числа от 0 до 1000, с шагом 5
        for (i = 0; i <= 1000; i += 5)
            Sum += i;

        Console.Write("Сумма равна " + Sum);

        Console.ReadKey();
    }
}
```

1.7.3 Область видимости переменных

Переменные можно описывать **локально**, т.е. внутри цикла. В этом случае их область видимости ограничивается только телом цикла:

```
for (int i = 0; i <= 1000; i += 5)     // i описана локально
    Sum += i;
```

Стоит отметить, что локально описывать переменные можно не только в операторе цикла for. Операторы if, while, switch, и ряд других также предоставляют такую возможность. Вообще, любые скобки группы { } определяют локальную область для описанных в ней переменных.

Занятие 1.8 Циклические операции. Операторы while и do-while

1.8.1 Оператор while

Оператор while реализует цикл с предусловием.
В полной форме оператор имеет синтаксис:

```
while (условие)
{
    // последовательность операторов
}
```

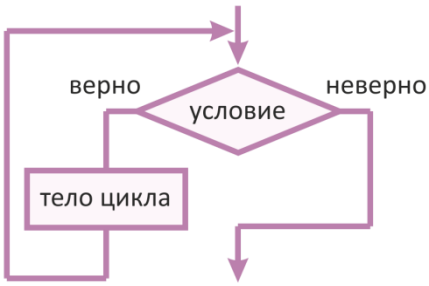


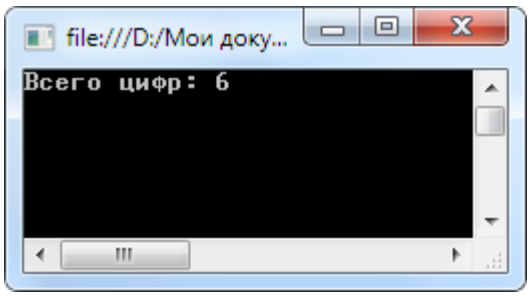
Рис. 1.20. Блок-схема оператора цикла с предусловием while.

Задача 1. Определить количество цифр заданного числа.
Решение.

Листинг 1.9

```
using System;

class Program
{
    static void Main()
    {
        int number = 123456;        // число
        byte digits_counter = 1;    // в любом числе есть хотя бы
                                    // одна цифра
        while (number >= 10)
        {
            number /= 10;           // отбрасываем цифру делением
            digits_counter++;       // увеличиваем счетчик цифр
        }

        Console.Write("Всего цифр: " + digits_counter);
        Console.ReadKey();
    }
}
```

Поскольку переменная number является целочисленной, то операция деления на 10 дает целый результат – неполное частное, что равносильно отбрасыванию последней цифры.



Благодаря функциональным возможностям синтаксиса C# можно переписать код цикла следующим образом:

```
while ((number /= 10) > 0)
{
    digits_counter++;       // увеличиваем счетчик цифр
}
```

В начале меняется значение переменной number:

```
number /= 10
```

а затем новое значение сравнивается с нулем (почему не 10?). Теперь переменная меняется в процессе проверки условия, а не в теле цикла.

1.8.2 Оператор do-while

Оператор do-while реализует цикл с постусловием.
В полной форме оператор имеет синтаксис:

```
do
{
    // последовательность операторов
} while (условие);
```

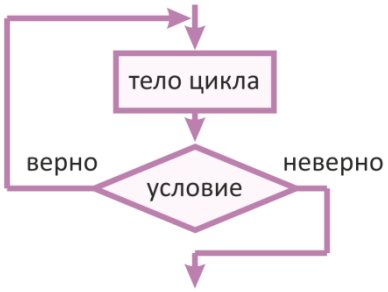


Рис. 1.21. Блок схема оператора цикла с постусловием do-while.

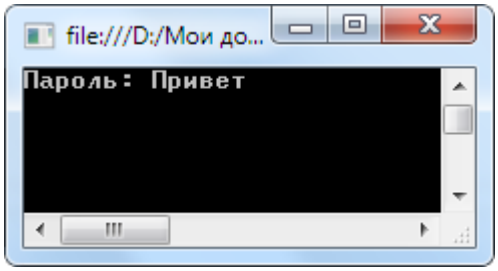
Задача 2. Организовать ввод текста до тех пор, пока он не совпадет с ключевым словом.

Решение.

Листинг 1.10

```
using System;

class Program
{
    static void Main()
    {
        string Pass = "Hello";
        string str;

        do
        {
            Console.Write("Пароль: ");      // подсказка
```

```
            str = Console.ReadLine();        // ввод текста
            Console.Clear();                 // очистка консоли
        } while(str != Pass);                // пока не совпадут
    }
}
```



Промежуточную переменную str можно и не заводить: для этого поместим команду ввода сразу в условие:

```
using System;

class Program
{
    static void Main()
    {
        string Pass = "Hello";

        do
        {
            Console.Clear();            // очистка консоли
            Console.Write("Пароль: ");       // подсказка
        } while(Console.ReadLine() != Pass); // пока не совпадут
    }
}
```

Занятие 1.9 Массивы

Массив *– программная структура, представляющая собой совокупность переменных одного типа. В C# массивы могут быть как одномерными, так и многомерными.*

1.9.1 Одномерный массив

Одномерный массив представляет собой список связанных переменных.
Общий синтаксис описания:

```
тип[] имя_массива = new тип[размер];
```

где размер определяет число элементов массива.

Создадим массив типа byte из 100 элементов:

```
byte[] Mass = new byte[100];
```

Описанный массив можно представить в виде совокупности ста ячеек размером в один байт каждая:

1-й	2-й	3-й	...	i-й	...	99-й	100-й

Для обращения к элементу массива в квадратных скобках необходимо указать индекс этого элемента.
Индексация элементов начинается с нуля!

Например, так выглядит присвоение десятому элементу значения 44:

```
Mass[9] = 44;
```

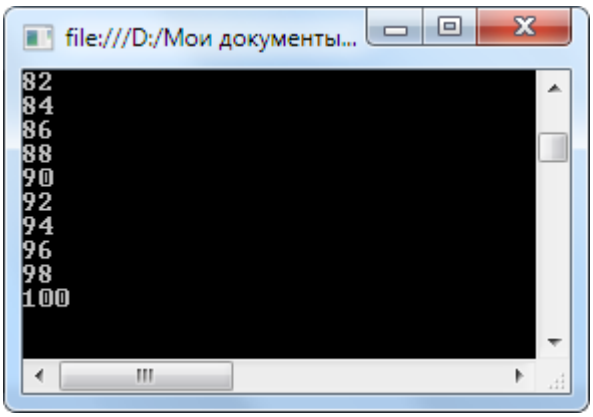
Mass[0]	Mass[1]	Mass[2]	...	Mass[9]	...	Mass[98]	Mass[99]
0	0	0	...	44	...	0	0

Задача 1. Опишите массив типа int на 50 элементов. Заполните массив четными числами, начиная с двойки.

Решение.

Листинг 1.11

```
using System;

class Program
{
    static void Main()
    {
        int[] Mass = new int[50];        // массив на 50 элементов

        for (int i = 0; i < 50; i++)
        {
            // добавляем 1, т.к. счетчик начат с нуля
            Mass[i] = 2 * (i + 1);
            Console.WriteLine(Mass[i]);
        }

        Console.ReadKey();
    }
}
```



Элементы массива можно задать и непосредственно:

```
int[] Mass = { 23, -45, 0, 10, 5, 1 };
```

1.9.2 Свойство Length

Поскольку массивы являются объектами, то они обладают рядом **свойств**. Одно из них – **Length**, возвращающее число элементов массива. Применение указанного свойства избавляет программиста от необходимости отслеживать границы массива.

В следующем примере переменной n присваивается значение количества элементов массива Mass:

```
int[] Mass = {34, 45, 89, 0 ,-23, 2015, 1};
int n = Mass.Length;          // n = 7
```

1.9.3 Операторы break и continue

Оператор **break** прерывает выполнение текущего цикла.

Оператор **continue** прерывает выполнение текущего шага цикла и переходит к следующему шагу.

1.9.4 Многомерные массивы

Массивы в C# могут быть любой размерности. Рассмотрим пример работы с двумерными массивами, которые можно представить как таблицу элементов одного типа.

Двумерный массив представляет собой таблицу проиндексированных переменных.
Общий синтаксис описания:

```
тип[,] имя_массива = new тип[размер_1,размер_2];
```

где размеры определяют число элементов массива по каждой размерности.

Ниже описан массив вещественных чисел размером 10x20:

```
double[,] Mass = new double[10,20];
```

Схематично он выглядит следующим образом[3]:

Mass[0,0]	Mass[0,1]	Mass[0,2]	...	Mass[0,19]
Mass[1,0]	Mass[1,1]	Mass[1,2]	...	Mass[1,19]
Mass[2,0]	Mass[2,1]	Mass[2,2]	...	Mass[2,19]
...	...	...	...	...
Mass[9,0]	Mass[9,1]	Mass[9,2]	...	Mass[9,19]

Элементы можно задавать с консоли, так и программно. Например, запись

```
double[,] Table = {
    {23.4, 22.5, 22.9},
    {28.0, 27.8, 29.7}
};
```

может соответствовать следующей таблице:

23.4	22.5	22.9
28.0	27.8	29.7

Задача 2. В двумерном массиве найти максимальный элемент для каждой строки.

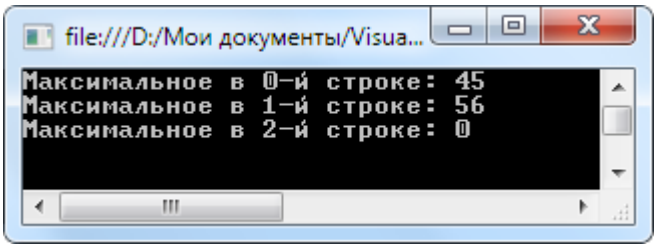
Решение.

Листинг 1.12

```
using System;

class Program
{
    static void Main()
    {
        // описываем двумерный массив целых чисел размером 3x5
        int[,] Table = {
            { 23, 45, -3, 45, 7 },
            { 56, -67, -45, 0, 23 },
            { -5, -1, 0, 0, 0 }
        };
```

[3] На самом деле мы можем первую размерность считать столбцом, а вторую – строкой. Для представления данных это не имеет значение, а требуемую обработку и вывод элементов можно организовать в любом случае.

```
            for (int i = 0; i < 3; i++)   // внешний цикл - по строкам
            {
                int max = Table[i, 0];    // пусть первый в строке i –
                                          // максимальный

                // внутренний цикл – по столбцам
                for (int j = 1; j < 5; j++)
                {
                    // если текущий больше максимального,
                    // перезаписываем
                    if (Table[i, j] > max)
                        max = Table[i, j];
                }

                Console.WriteLine("Максимальное в {0}-й строке: {1}",
i, max);
            }

            Console.ReadKey();
        }
    }
```



Занятие 1.10 Классы Math и Random

1.10.1 Класс Math

Для математических операций часто требуется использование различных функций и констант. Их предоставляет класс **Math**.

Рассмотрим примеры использования:

```
double a = Math.Sqrt(100);        // корень из 100
double Pi = Math.PI;              // постоянное число PI
double b = Math.Cos(2 * Pi);      // cos(2PI)
int max = Math.Max(33, 67);       // максимальное из чисел 33 и 67
```

1.10.2 Класс Random

Для генерации случайных чисел используется класс **Random**. Рассмотрим методы Next и NextDouble.

Метод **Next** имеет три различные вариации.

Первая генерирует целое число на всем допустимом диапазоне целых чисел:

```
Random X = new Random();          // создаем экземпляр класса Random
Console.Write(X.Next());          // генерируем случайное число из
                                  // диапазона int
```

Вторая генерирует целые числа от нуля до указанного:

```
Console.Write(X.Next(10));
```

Третья генерирует числа в указанном диапазоне:

```
Console.Write(X.Next(3,10));
```

Метод **NextDouble** генерирует вещественное число от 0 до 1:

```
Random X = new Random();
Console.WriteLine(X.NextDouble());
```

Занятие 1.11 Подпрограмма

1.11.1 Подпрограмма

В .NET понятие **процедур** и **функций** обобщено в понятие **метода**. Метод не является подпрограммой сам по себе, он всегда является частью какого-либо класса.

Например, требуется написать приложение, которое вычисляет число комбинаций m объектов из n различных объектов. Это комбинаторная задача сочетаний без повторов, поэтому число комбинаций высчитывается по формуле:

$$C_n^m = \frac{n!}{m!\,(n-m)!}$$

Формула требует трижды вычислять факториал (если не использовать некоторые дополнительные преобразования). Очевидно, что в программе будет полезно создать функцию, возвращающую факториал числа.

1.11.2 Функция

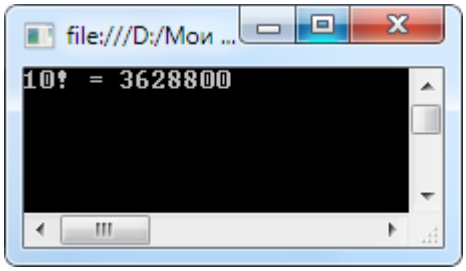
***Функция** – подпрограмма, к которой можно обратиться из другого места программы. Основная задача функции – возвратить некоторое значение.*

Задача 1. Написать функцию для вычисления факториала натурального числа n (передается через параметр).

Решение.

Листинг 1.13

```
using System;

class Program
{
    static void Main()
    {
        int rez = Factorial(10);   // вызов функции для числа 10
        Console.WriteLine("{0}! = {1} ", 10, rez);   // вывод 10!
        Console.ReadKey();
    }

    // Функция, возвращающая факториал числа
    static int Factorial(int n)
    {
        int factor = 1;     // как множитель
                            // (напомним, что 0! = 1)
        for (int i = 1; i <= n; i++)
            factor *= i;

        return factor;      // возврат значения из функции
    }
}
```



1.11.3 Разбор программы

Синтаксис функции:

```
[доступ] [static] возращаемый_тип имя(список_параметров)
{
    // тело функции
}
```

Доступ *– уровень видимости функции.*
Возвращаемый тип *– тип значения, которое возвращается в основную программу.*
Список параметров *– дополнительные данные для работы функции.*

Доступ может быть **открытым** (**public**) и **закрытым** (**private**). Сейчас мы подробно не рассматриваем этот вопрос. Ключевое слово **static** будет рассмотрено в разделах, посвященных классам. Просто запомните, что сейчас мы его должны использовать при описании функций.

Возвращаемый тип должен соответствовать типу значения, которое получается в ходе работы функции. Факториал – натуральное число, поэтому взят тип int.

Список параметров – одна или более переменных, в которых функции передаются дополнительные данные. В нашей программе функции передается число 10, факториал которого требуется вычислить. Каждый параметр указывается вместе с типом. Функции могут быть и без параметров: в этом случае при вызове функции скобки остаются пустыми.

1.11.4 Механизм работы функции

При выполнении команды Factorial(10) осуществляется вызов функции и переход на соответствующий участок кода с ее описанием. Переменная n в списке параметров функции получает копию переданного значения (т.е. 10) и работает с ним. Наконец, сформированный в переменной factor результат возвращается в основную программу с помощью команды возврата:

```
return factor;
```

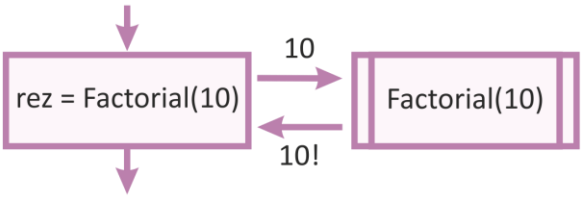


Рис. 1.22. Схема работы функции Factorial.

1.11.5 Формальные и фактические параметры

При обращении к функции в основной программе в скобках указываются передаваемые аргументы (если их больше одного, то через запятую):

```
int rez = Factorial(10);
```

Аргумент может быть непосредственным значением (например, как у нас, числом), а может быть и переменной. Такие параметры называются **фактическими**.

Теперь обратим внимание на заголовок функции:

```
static int Factorial(int n)
```

В скобках описан параметр n, который принимает переданное значение. Это **формальный** параметр. На самом деле формальный параметр – это локальная переменная внутри функции; он существует только во время работы функции.

Важно отметить, что мы передаем параметр по значению[4]. Поэтому формальный параметр является лишь копией фактического аргумента и любое его изменения внутри функции никак не повлияет на переданный («внешний») аргумент.

1.11.6 Оператор return

Функция обязательно должна вернуть определенное значение. Этим занимается оператор return:

```
return factor;
```

С выполнением оператора завершается и выполнение функции, а управление передается обратно в основной (вызывающий) блок.

Примечательно, что в подпрограмме может быть несколько операторов return.

Необходимо, чтобы любая возможная ветвь функции завершалась оператором return. Иначе генерируется ошибка еще на этапе компиляции приложения.

Задача 2. Написать функцию для вычисления числа сочетаний. Использовать функцию вычисления факториала.

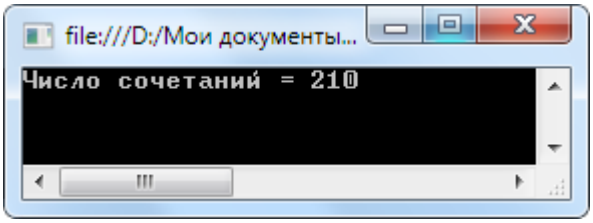
Решение.

Листинг 1.14

```
using System;

class Program
{
    static void Main()
    {
        int n = 10;
        int m = 6;
        int rez = Factorial(n) /
                  (Factorial(m) * Factorial(n - m));
        Console.WriteLine("Число сочетаний = " + rez);

        Console.ReadKey();
    }
```

[4] В отличии от параметров-ссылок, которые позволяют менять аргумент.

```
        // функция, возвращающая факториал числа
        static int Factorial(int n)
        {
            int factor = 1;

            for (int i = 1; i <= n; i++)
                factor *= i;

            return factor;
        }
    }
```



Функция возвращает значение; следовательно ее можно подставлять в другие операции.

1.11.7 Повышение абстракции

Число сочетаний может понадобиться для реализации и многократного использования в более сложных типах вычислений. Поэтому его также можно оформить в виде функции.

Задача 3. Ввести в описание класса функцию по вычислению числа сочетаний. Функция принимает два параметра: общее число элементов n и выбираемое m.

Решение.

Листинг 1.15

```
using System;

class Program
{
    static void Main()
    {
        int rez = Sochet(10, 6); // число сочетаний из 10 по 6
        Console.WriteLine("Число сочетаний = " + rez);

        Console.ReadKey();
    }

    // функция, возвращающая факториал числа
    static int Factorial(int n)
    {
```

```
        int factor = 1;

        for (int i = 1; i <= n; i++)
            factor *= i;

        return factor;
    }

    // функция, возвращающая число сочетаний
    static int Sochet(int n, int m)
    {
        return Factorial(n) / (Factorial(m) * Factorial(n - m));
    }
}
```

1.11.8 Процедуры

Процедура *– подпрограмма, к которой можно обратиться из другого места программы. Основная задача процедуры – произвести логически единый блок операций.*
Процедуры имеют синтаксис, похожий на функции. Главное отличие – все процедуры имеют тип ***void*** *и не возвращают значение.*

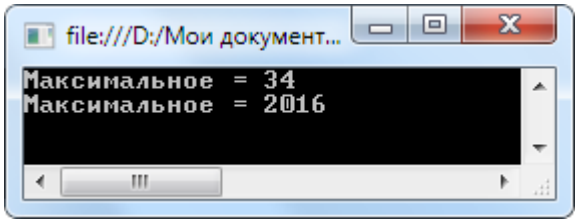
Задача 4. Организовать процедуру печати максимального элемента из двух.

Решение.

Листинг 1.16

```
using System;

class Program
{
    static void Main()
    {
        Max(10, 34);    // вызов процедуры
        int a = 2016;
        Max(a, 2014);   // вызов процедуры

        Console.ReadKey();
    }

    // процедура с двумя параметрами
    static void Max(double a, double b)
    {
        double max = a;

        if (b > a) max = b;

        Console.WriteLine("Максимальное = " + max);
```

```
        }
    }
```



Из примера видно, что процедура не использует оператор возврата return: он ей не нужен, поскольку не требуется возвращать значение. Подпрограмма завершается, достигнув закрывающую фигурную скобку блока процедуры.

Однако процедуры также могут его использовать (неоднократно), чтобы, например, досрочно завершить свое выполнение:

```
return;
```

1.11.9 Процедура или функция?

В языке C# процедура – это функция без возврата.
Поэтому в C# наряду с «термином» метод можно использовать термин «функция».

Список литературы

Основная литература:

1. *Гуриков С. Р.* Введение в программирование на языке Visual C# : учеб. пособие / С. Р. Гуриков – М.: Форум: НИЦ ИНФРА-М, 2013. – 448 с.: 70x100 1/16. – (Высшее образование: Бакалавриат). (переплет) ISBN 978-5-91134-738-3, 500 экз.
2. *Хорев П. Б.* Объектно-ориентированное программирование с примерами на C# : учеб. пособие / П. Б. Хорев – М. : Форум, НИЦ ИНФРА-М, 2016. – 200 с.: 70x100 1/16. – (Высшее образование: Бакалавриат) (Обложка) ISBN 978-5-00091-144-0.
3. *Шакин В. Н.* Базовые средства программирования на Visual Basic в среде VisualStudio Net. Практикум: Учебное пособие / В. Н. Шакин – М. : Форум, НИЦ ИНФРА-М, 2015. – 288 с.: 70x100 1/16. – (Высшее образование: Бакалавриат) (Переплёт 7БЦ) ISBN 978-5-00091-054-2.

Дополнительная литература

1. Практикум на ЭВМ. Часть 1 [Электронный ресурс]: учеб. Пособие / — Электрон. текстовые данные.— М. : Евразийский открытый институт, 2012.— 263 с.
2. *Канцедал С. А.* Алгоритмизация и программирование : учеб. пособие / С. А. Канцедал. – М. : ИД ФОРУМ: НИЦ ИНФРА-М, 2014. – 352 с.: ил.; 60x90 1/16. – (Профессиональное образование). (переплет). ISBN 978-5-8199-0355-1, 500 экз.
3. *Туркин О. В.* VBA. Практическое программирование [Электронный ресурс] / Туркин О. В. – Электрон. текстовые данные. – М.: СОЛОН-ПРЕСС, 2010. – 128 с.

Учебное издание

Платформа .NET Framework. Язык C#

Подписано в печать 31.05.2018. Формат 60x84 1/16. Усл.-печ. л 2,79.
Тираж 200 экз. Заказ 66.

Отпечатано с готового оригинала-макета
ООО «Издательство «Шерлок-пресс»
г. Владимир, ул. Девическая, д. 11