

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования «Казанский (Приволжский) федеральный университет»

Набережночелнинский институт (филиал)

**Кафедра Экономика предприятий и организаций**

## **Основы программирования на языке C#**

*учебно-методическое пособие*

Набережные Челны  
2020 г.

УДК 004.94  
ББК 65.054

Печатается по решению учебно-методической комиссии экономического отделения Набережночелнинского института (филиала) федерального государственного автономного образовательного учреждения высшего образования «Казанский (Приволжский) федеральный университет», от «27» 01 2020 г. (протокол № 2)

Рецензенты:

Доктор экономических наук, профессор А.С. Пуряев

Доктор экономических наук, профессор А.Н. Макаров

Гареева Г.А. Основы программирования на языке С#: учебно-методическое пособие / Гареева Г.А., Григорьева Д.Р. – Набережные Челны: Изд-во Набережночелнинского института КФУ, 2020. – 59 с.

Учебно-методическое пособие предназначено для проведения практических и лабораторных занятий по дисциплине. Составлено с учетом требований Федеральных Государственных образовательных стандартов высшего профессионального образования.

Учебно-методическое пособие предназначено для использования в учебном процессе студентами технических направлений в экономике и экономического отделения дневной, заочной и дистанционной форм обучения.

© Гареева Г.А., Григорьева Д.Р., 2020

© НЧИ КФУ, 2020

© Кафедра Экономика предприятий и организаций, 2020 г.

## Содержание

<b>ВВЕДЕНИЕ</b> .....	<b>4</b>
<b>1. НАХОЖДЕНИЕ МИН/МАКС ЗНАЧЕНИЙ В МАССИВЕ</b> .....	<b>5</b>
<b>2. ОПЕРАТОР ВЫБОРА SWITCH-CASE</b> .....	<b>7</b>
<b>3. ТИПЫ ДАННЫХ</b> .....	<b>9</b>
ЦЕЛОЧИСЛЕННЫЕ ТИПЫ .....	10
ТИПЫ С ПЛАВАЮЩЕЙ ТОЧКОЙ.....	13
ДЕСЯТИЧНЫЙ ТИП ДАННЫХ .....	14
СИМВОЛЫ.....	15
ЛОГИЧЕСКИЙ ТИП ДАННЫХ .....	16
КОНСТАНТЫ .....	16
<b>4. ФУНКЦИИ ДЛЯ ВВОДА/ВЫВОДА В ЯЗЫКЕ C#</b> .....	<b>18</b>
<b>5. ОПЕРАТОР ПРИСВАИВАНИЯ</b> .....	<b>19</b>
<b>6. УСЛОВНЫЕ ОПЕРАТОРЫ</b> .....	<b>21</b>
<b>7. ЦИКЛЫ FOR И WHILE</b> .....	<b>23</b>
ЦИКЛ FOR .....	23
ЦИКЛ WHILE .....	26
<b>8. ЦИКЛЫ DO...WHILE И FOREACH</b> .....	<b>28</b>
ЦИКЛ DO. . . WHILE .....	28
ЦИКЛ FOREACH .....	29
<b>9. МЕТОДЫ</b> .....	<b>32</b>
ИСПОЛЬЗОВАНИЕ ПАРАМЕТРОВ.....	35
<b>10. МАССИВЫ</b> .....	<b>37</b>
НЕЯВНО ТИПИЗИРОВАННЫЕ МАССИВЫ .....	39
ОПРЕДЕЛЕНИЕ МАССИВА ОБЪЕКТОВ.....	40
СВОЙСТВО LENGTH .....	41
<b>11. ОПЕРАТОРЫ ПЕРЕХОДА</b> .....	<b>42</b>
ОПЕРАТОР GOTO .....	42
ОПЕРАТОР BREAK.....	44
ОПЕРАТОР CONTINUE .....	45
ОПЕРАТОР RETURN .....	46
<b>ПРИМЕРЫ РЕШЕНИЯ ТИПОВЫХ ЗАДАЧ</b> .....	<b>49</b>
<b>СПИСОК ЛИТЕРАТУРЫ</b> .....	<b>71</b>

## ВВЕДЕНИЕ

C# — это типизированный, объектно-ориентированный, простой и в то же время мощный язык программирования, который позволяет разработчикам создавать многофункциональные приложения. Разработан в 1998-2001 годах группой инженеров под руководством Андерса Хейлсберга как основной язык разработки приложений для платформы Microsoft .NET.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. C# - это фактически гибрид разных языков. Переняв многое от своих предшественников и опираясь на практику их использования, C# синтаксически не менее чист, чем Java, так же прост, как Visual Basic, и обладает практически той же мощностью и гибкостью, что и C++.

Особенности C#:

- Полный и хорошо определенный набор основных типов.
- Встроенная поддержка автоматической генерации XML-документации.
- Автоматическое освобождение динамически распределенной памяти.
- Возможность отметки классов и методов атрибутами, определяемыми пользователем.
- Полный доступ к библиотеке базовых классов .NET, а также легкий доступ к Windows API.
- Указатели и прямой доступ к памяти, если они необходимы.
- Поддержка свойств и событий в стиле Visual Basic.
- Простое изменение ключей компиляции. Позволяет получать исполняемые файлы или библиотеки компонентов .NET, которые могут быть вызваны другим кодом так же, как элементы управления ActiveX.
- Возможность использования C# для написания динамических web-страниц ASP.NET.

В C# отсутствуют некоторые ключевые моменты, необходимые для создания высокопроизводительных приложений, в частности подставляемые функции и деструкторы, выполнение которых гарантируется в определенных точках кода.

## 1. НАХОЖДЕНИЕ МИН/МАКС ЗНАЧЕНИЙ В МАССИВЕ

Часто в задачах возникает необходимость найти максимальное (минимальное) число во множестве (массиве) элементов. Для этого используется простой алгоритм, использующий один цикл и одну переменную для хранения этого числа.

Для решения данной задачи, можно использовать следующий алгоритм:

Запись решения на естественном языке:

1. Начало программы.
2. Получаем множество чисел, в котором будем искать максимальное (минимальное) значение.
3. Получаем длину массива, чтобы запустить цикл перебора элементов.
4. Объявляем переменную, которая будет хранить найденное максимальное (минимальное) число.
5. Помещаем в данную переменную первый элемент массива.
6. Описываем цикл с первого элемента, который будет перемещаться по массиву и выбирать числа.
7. Внутри цикла сравниваем выбранное число из цикла с числом в переменной.

Если находится максимум:

Если число в буфере меньше числа в переменной, то перезаписываем переменную этим числом.

Если число в буфере больше числа в переменной, то пропускаем элемент.

Если находится минимум:

Если число в буфере больше числа в переменной, то перезаписываем переменную этим числом.

Если число в буфере меньше числа в переменной, то пропускаем число.

8. Выбираем в цикле следующий элемент массива.

Если элементы есть, то переходим на пункт 6.

Если элементов нет, значит, переменная хранит максимальное (минимальное число).

9. Конец работы алгоритма.

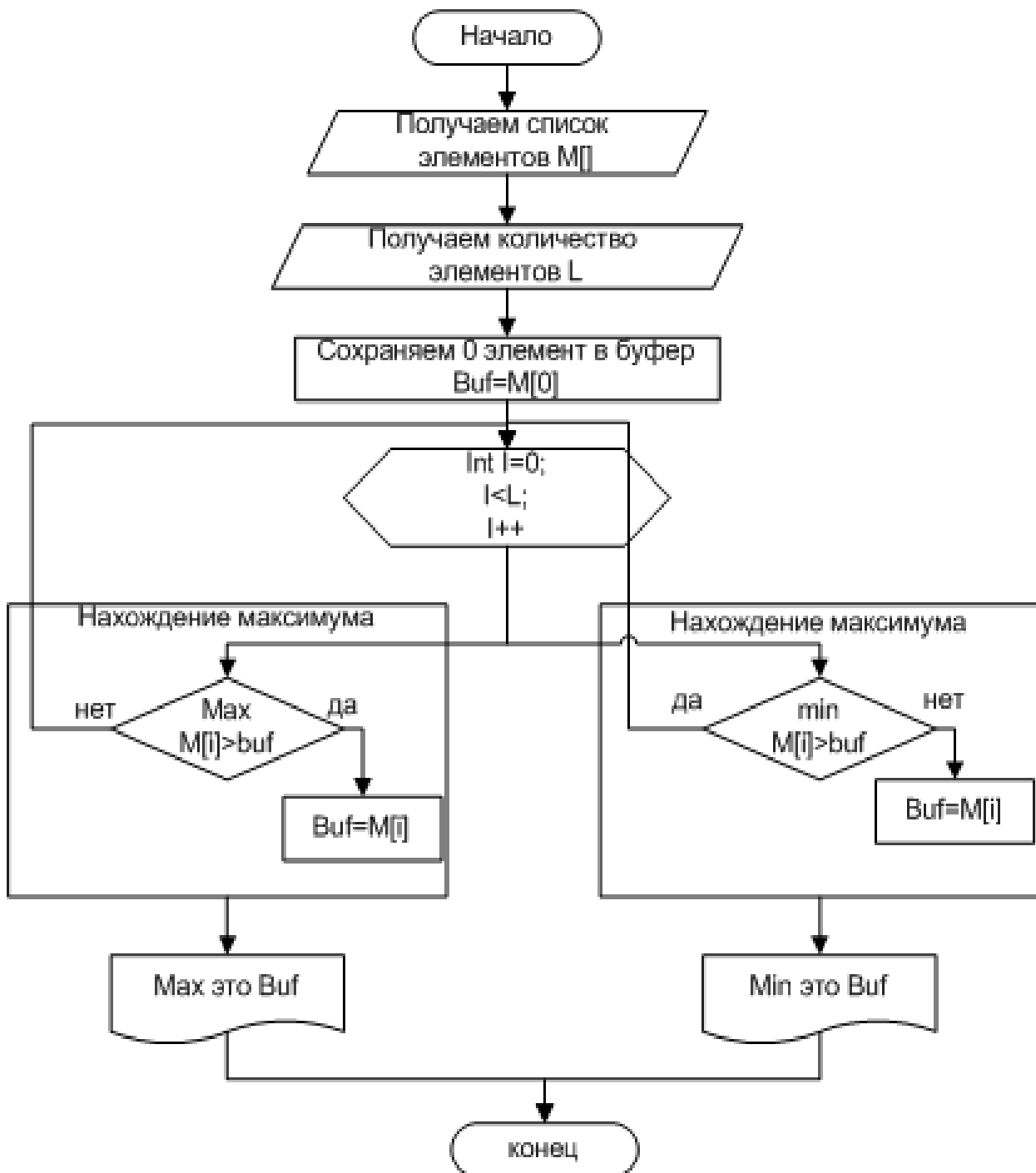


Рисунок 1. Алгоритм нахождения min/max значения

## 2. ОПЕРАТОР ВЫБОРА В SWITCH-CASE

Операторы выбора в C# позволяют выполнять или не выполнять определенные блоки кода в зависимости от наступления определенных условий. Если условие выполняется, выражению присваивается значение true («истина»), и код внутри операторов выбора выполняется, иначе – присваивается значение false («ложь»), и код игнорируется.

В C# самая популярная конструкция выбора – if-else («если-то»). Проводя аналог с человеческим языком, получаем: «если выполняется такое условие, делать вот это, а иначе делать вот это». Проверяемые условия – это операторы сравнения (те, кто сходны с математическими) и логического сравнения («и», «или»). После проверки условия и выполнения того или иного блока кода управление передается следующему за конструкцией выбора коду. Проверка, если она не находится в теле цикла, выполняется однократно.

### Пример:

На форме есть текстовое поле с именем textBox1, в которое пользователь вписывает ответ на заданный вопрос (user\_ans). Правильный ответ (переменная tru\_ans) – «Яблоко»:

```
string tru_ans = «Яблоко»;  
string user_ans = textBox1.Text;  
  
if (tru_ans == user_ans) //если ответ пользователя совпадает с правильным  
{  
    MessageBox.Show(«Ответ верный!»);  
}  
else //а иначе (если ответы не совпадают)  
{  
    MessageBox.Show(«Ответ неверный!»);  
}
```

Конструкцию выбора можно реализовать и по-другому: с помощью операторов **switch-case** («переключение на случай»). Оператор **switch** содержит проверяемое выражение. Внутри есть несколько Case с вариантами кода.

Выполняться будет только один из Case, значение которого совпадет со значением switch. Операторов Case может быть множество, но они должны быть уникальными. После каждого блока Case должен стоять оператор перехода break. Если не выполняется ни один из случаев (Case), управление переходит к оператору Default. Если оператор Default не предусмотрен программистом, осуществляется выход из конструкции выбора и переход к последующему фрагменту.

```
int a = 1;
switch (a)
{
    case 1:
        MessageBox.Show(«Первый случай»);
        break;

    case 2:
        MessageBox.Show(«Второй случай»);
        break;

    default:
        MessageBox.Show(«Ни один из случаев»);
        break;
}
```

Значение первого *Case* здесь равно единице. Это соответствует значению *Switch*, равному единице. Значит, на экран выведется сообщение «Первый случай». Если изменить код, и присвоить переменной *a* значение 5, выведется сообщение «Второй случай». Если переменная *a* станет равна 50, на экране появится сообщение «Ни один из случаев».



### 3. ТИПЫ ДАННЫХ

Типы данных имеют особенное значение в С#, поскольку это строго типизированный язык. Все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, причем недопустимые операции не компилируются. Строгий контроль типов позволяет исключить ошибки и повысить надежность программ. Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определенному типу. Такого понятия, как "бестиповая" переменная, в данном языке программирования вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

В С# имеются две общие категории встроенных типов данных: *типы значений* и *ссылочные типы*. Они отличаются по содержимому переменной. Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение.

Типы значений:

- Целочисленные типы (*byte, sbyte, char, short, ushort, int, uint, long, ulong*)
- Типы с плавающей запятой (*float, double*)
- Тип *decimal*
- Тип *bool*
- Перечисления *enum*
- Структуры (*struct*)

Ссылочные типы:

- Тип *object*
- Тип *string*
- Классы (*class*)
- Интерфейсы (*interface*)
- Делегаты (*delegate*)

Эти типы сохраняются в разных местах памяти: типы значений сохраняются в области, известной как стек, а ссылочные типы — в области, называемой управляемой кучей.

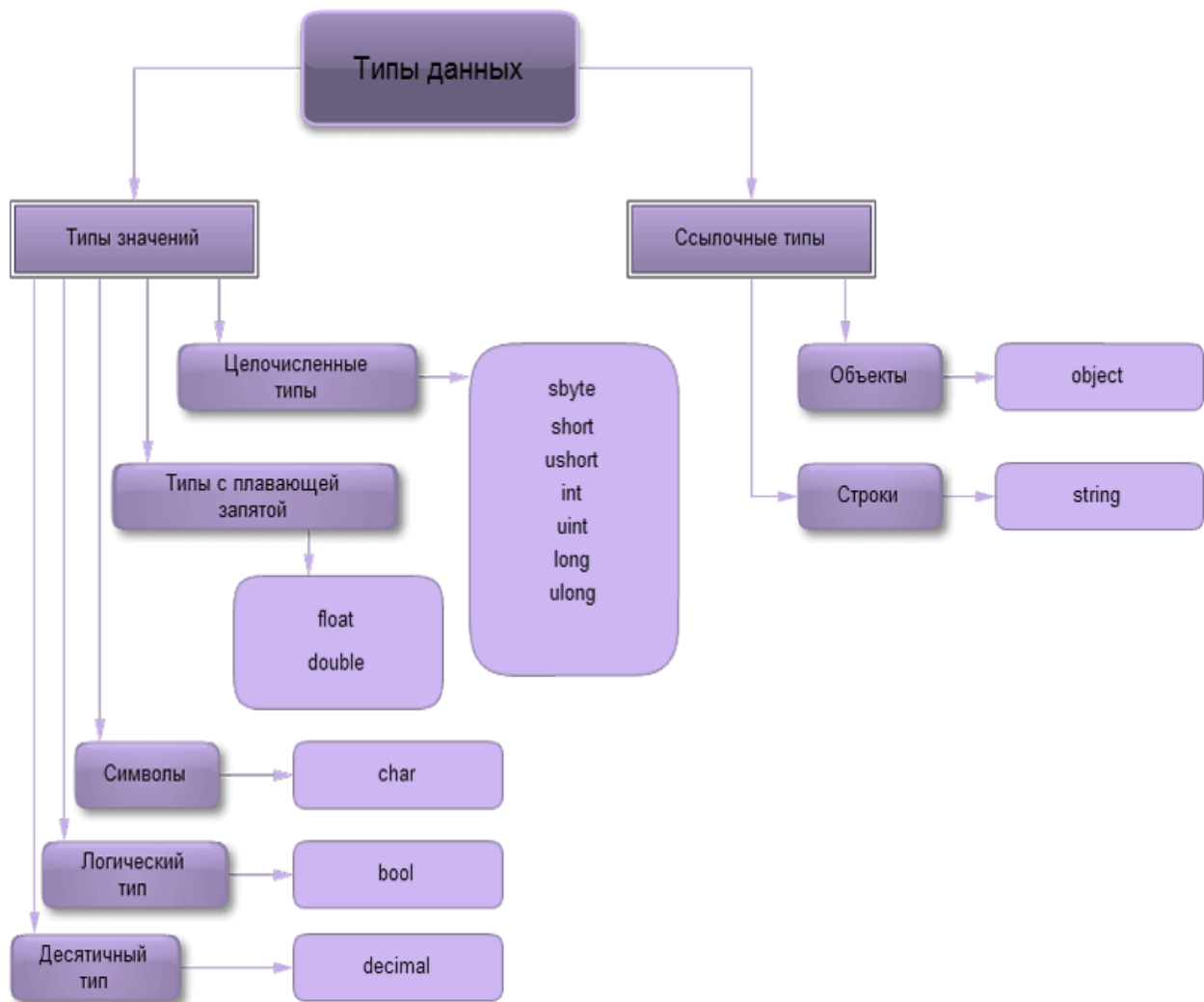


Рисунок 2. Типы данных в C#

## Целочисленные типы

В C# определены девять целочисленных типов: *char*, *byte*, *sbyte*, *short*, *ushort*, *int*, *uint*, *long* и *ulong*. Но тип *char* применяется, главным образом, для представления символов и поэтому рассматривается отдельно. Остальные восемь целочисленных типов предназначены для числовых расчетов. Ниже представлены их диапазон представления чисел и разрядность в битах.

### Целочисленные типы C#

Тип	Тип CTS	Разрядность в битах	Диапазон
byte	System.Byte	8	0:255
sbyte	System.SByte	8	-128:127
short	System.Int16	16	-32768: 32767
ushort	System.UInt16	16	0: 65535
int	System.Int32	32	-2147483648: 2147483647
uint	System.UInt32	32	0: 4294967295
long	System.Int64	64	-9223372036854775808: 9223372036854775807
ulong	System.UInt64	64	0: 18446744073709551615

В C# определены оба варианта различных целочисленных типов: со знаком и без знака. Целочисленные типы со знаком отличаются от аналогичных типов без знака способом интерпретации старшего разряда целого числа. Так, если в программе указано целочисленное значение со знаком, то компилятор C# сгенерирует код, в котором старший разряд целого числа используется в качестве флага знака. Число считается положительным, если флаг знака равен 0, и отрицательным, если он равен 1.

Отрицательные числа практически всегда представляются методом дополнения до двух, в соответствии с которым все двоичные разряды отрицательного числа сначала инвертируются, а затем к этому числу добавляется 1.

Самым распространенным в программировании целочисленным типом является тип *int*. Переменные типа *int* нередко используются для управления циклами, индексирования массивов и математических расчетов общего назначения. Когда же требуется целочисленное значение с большим диапазоном представления чисел, чем у типа *int*, то для этой цели имеется целый ряд других целочисленных типов.

Если значение нужно сохранить без знака, то для него можно выбрать тип *uint*, для больших значений со знаком — тип *long*, а для больших значений без знака — тип *ulong*. Ниже приведена программа, вычисляющая расстояние от Земли до Солнца в сантиметрах. Для хранения столь большого значения в ней используется переменная типа *long*:

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            long result;
            const long km = 149800000; // расстояние в км.

            result = km * 1000 * 100;
            Console.WriteLine(result);
            Console.ReadLine();
        }
    }
}
```

Всем целочисленным переменным значения могут присваиваться в десятичной или шестнадцатеричной системе обозначений. В последнем случае требуется префикс *0x*:

```
long x = 0x12ab;
```

Если возникает какая-то неопределенность относительно того, имеет ли целое значение тип *int*, *uint*, *long* или *ulong*, то по умолчанию принимается *int*. Чтобы явно специфицировать, какой другой целочисленный тип должно иметь значение, к числу можно добавлять следующие символы:

```
uint ui = 1234U;
```

```
long l = 1234L;
```

```
ulong ul = 1234UL;
```

U и L можно также указывать в нижнем регистре, хотя строчную L легко зрительно спутать с цифрой 1 (единица).

### Типы с плавающей точкой

Типы с плавающей точкой позволяют представлять числа с дробной частью. В C# имеются две разновидности типов данных с плавающей точкой: *float* и *double*. Они представляют числовые значения с одинарной и двойной точностью соответственно. Так, разрядность типа *float* составляет 32 бита, что приблизительно соответствует диапазону представления чисел от  $5E-45$  до  $3,4E+38$ . А разрядность типа *double* составляет 64 бита, что приблизительно соответствует диапазону представления чисел от  $5E-324$  до  $1,7E+308$ .

Тип данных *float* предназначен для меньших значений с плавающей точкой, для которых требуется меньшая точность. Тип данных *double* больше, чем *float*, и предлагает более высокую степень точности (15 разрядов).

Если нецелочисленное значение жестко кодируется в исходном тексте (например, 12.3), то обычно компилятор предполагает, что подразумевается значение типа *double*. Если значение необходимо специфицировать как *float*, потребуется добавить к нему символ F (или f):

```
float f = 12.3F
```

## Десятичный тип данных

Ключевое слово `decimal` обозначает 128-разрядный тип данных. По сравнению с типами данных с плавающей запятой, диапазон значений типа `decimal` меньше, а точность выше, благодаря чему этот тип подходит для финансовых расчетов.

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // *** Расчет стоимости капиталовложения с ***
            // *** фиксированной нормой прибыли ***
            decimal money, percent;
            int i;
            const byte years = 15;

            money = 1000.0m;
            percent = 0.045m;

            for (i = 1; i <= years; i++)
            {
                money *= 1 + percent;
            }

            Console.WriteLine("Общий доход за {0} лет: {1} $$",years,money);
            Console.ReadLine();
        }
    }
}
```

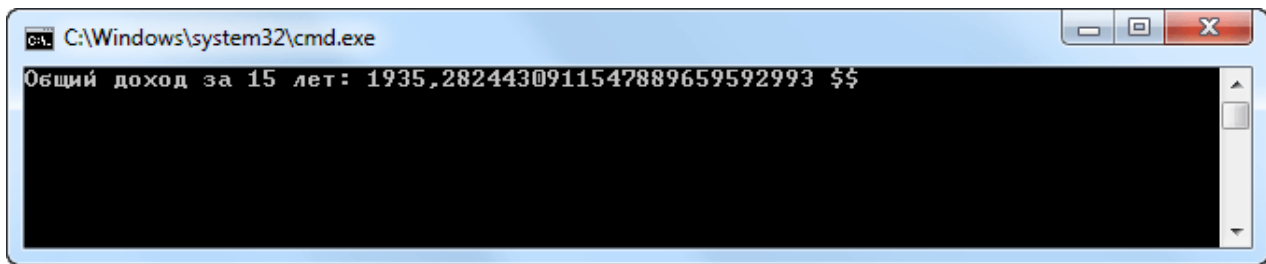


Рисунок 3. Результат работы программы с типом данных decimal

## Символы

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, а 16-разрядным кодом, который называется *юникодом (Unicode)*. В юникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете. Если для многих естественных языков, в том числе английского, французского и немецкого, характерны относительно небольшие алфавиты, то в ряде других языков, например китайском, употребляются довольно обширные наборы символов, которые нельзя представить 8-разрядным кодом. Для преодоления этого ограничения в C# определен тип *char*, представляющий 16-разрядные значения без знака в пределах от 0 до 65 535. При этом стандартный набор символов в 8-разрядном коде ASCII является подмножеством юникода в пределах от 0 до 127. Следовательно, символы в коде ASCII по-прежнему остаются действительными в C#.

Для того чтобы присвоить значение символьной переменной, достаточно заключить это значение (т.е. символ) в *одинарные кавычки*:

```
char ch;  
ch = 'Z';
```

Несмотря на то, что тип *char* определен как целочисленный, его не следует путать со всеми остальными целочисленными типами. Дело в том, что в C# отсутствует автоматическое преобразование символьных значений в целочисленные и обратно. Например, следующий фрагмент кода содержит ошибку:

```
char ch;
```

`ch = 8; // ошибка, правильный вариант ch = '8';`

Наравне с представлением *char* как символьных литералов, их можно представлять как 4-разрядные шестнадцатеричные значения Unicode (например, `\u0041`), целочисленные значения с приведением (например, `(char) 65`) или же шестнадцатеричные значения (например, `\x0041`). Кроме того, они могут быть представлены в виде управляющих последовательностей.

## Логический тип данных

Для работы с логическими величинами используется тип *bool*. Допустимы два значения: `true` (истина) и `false` (ложь). Пример объявления переменной логического типа:

```
bool p;
```

Для данных логического типа допустимы операции проверки на равенство и неравенство.

Следует иметь в виду, что для булевого типа в C# запрещены какие-либо преобразования в целые типы.

## Константы

Константа — это переменная, значение которой не меняется за время ее существования. Предваряя переменную ключевым словом *const* при ее объявлении и инициализации, вы объявляете ее как константу:

```
const int a = 100; // Это значение не может быть изменено
```

Ниже перечислены основные характеристики констант:

Они должны инициализироваться при объявлении, и однажды присвоенные им значения никогда не могут быть изменены.

Значение константы должно быть вычислено во время компиляции. Таким образом, инициализировать константу значением, взятым из другой переменной, нельзя. Если все-таки нужно это сделать, используйте поля только для чтения.



Константы всегда неявно статические. Нельзя включать модификатор `static` в объявление константы.

Использование констант в программах обеспечивает, по крайней мере, три преимущества:

Константы облегчают чтение программ, заменяя "магические" числа и строки читаемыми именами, назначение которых легко понять.

Константы облегчают модификацию программ. Например, предположим, что в программе `C#` имеется константа `SalesTax` (налог с продаж), которой присвоено значение 6 процентов. Если налог с продаж когда-нибудь изменится, вы можете модифицировать все вычисления налога, просто присвоив новое значение этой константе, и не понадобится просматривать код в поисках значений и изменять каждое из них, надеясь, что оно нигде не будет пропущено.

Константы позволяют избежать ошибок в программах. Если попытаться присвоить новое значение константе где-то в другом месте программы, а не там, где она объявлена, компилятор выдаст сообщение об ошибке.

#### 4. ФУНКЦИИ ДЛЯ ВВОДА/ВЫВОДА В ЯЗЫКЕ C#

Функции для консольного ввода и вывода в языке C# существенно отличаются от аналогичных функций языка C++. Таких функций 4, и все они являются членами класса Console. Функции WriteLine и Write обеспечивают вывод на экран и отличаются тем, что функция WriteLine после вывода обеспечивает перевод вывода на новую строку на экране, а функция Write такой перевод не выполняет. Для ввода применяются функции ReadLine и ReadKey. Функция ReadLine обеспечивает ввод строки. Если содержимое строки является числом, то требуется преобразование строки в число. Такое преобразование выполняет функция Parse, которая записывается после точки после имени типа данных. Для обратного перевода в строку текста применяется функция ToString. Функция ReadKey позволяет ввести код нажатой клавиши, и она часто применяется для останова экрана пользователя после вывода на экран. Если эту функцию не записать, то экран быстро исчезнет, не дав возможности посмотреть, что было выведено.

##### **Пример:**

```
double a,c; // Объявление переменных
int b;
Console.WriteLine("Введите число"); // С переходом на новую строку
a = double.Parse(Console.ReadLine());// Ввод с преобразованием
Console.WriteLine("Введите второе число");
b = int.Parse(Console.ReadLine());// Ввод с преобразованием
c = a + b;
Console.WriteLine("Сумма {0} и {1} = {2}", a, b, c);
Console.ReadKey();// Остановка экрана
```

В кавычках в круглых скобках записывается поясняющий текст и задается формат вывода. Для указания формата применяются пары фигурных скобок, в которых записываются целые числа, указывающие на порядок вывода переменных, имена которых записаны после кавычек. Отсчет начинается от нуля.

В примере показано, что числа в фигурных скобках следуют подряд, что указывают на совпадение порядка вывода элементов с порядком записи имен переменных, список которых приведен после кавычек. Принципиально, изменяя порядок следования чисел в фигурных скобках, не изменяя порядка записи переменных в списке, можно изменять порядок вывода переменных на экран.

В списке переменных для вывода можно применять арифметические выражения. Например, строку вывода из примера выше можно записать и так (в примере выше без использования переменной *c*):

```
Console.WriteLine("Сумма {0} и {1} = {2}", a, b, a + b);
```

## 5. ОПЕРАТОР ПРИСВАИВАИВАНИЯ

Оператор присваивания обозначается одиночным знаком равенства (=). В C# оператор присваивания действует таким же образом, как и в других языках программирования. Ниже приведена его общая форма:

*имя\_переменной* = *выражение*

Здесь *имя\_переменной* должно быть совместимо с типом выражения. Оператор присваивания позволяет создавать цепочку операций присваивания. Рассмотрим следующий фрагмент кода:

```
int x, y, z;
```

```
x = y = z = 10; // присвоить значение 10 переменным x, y и z
```

В приведенном выше фрагменте кода одно и то же значение 10 задается для переменных *x*, *y* и *z* с помощью единственного оператора присваивания. Это значение присваивается сначала переменной *z*, затем переменной *y* и, наконец, переменной *x*. Такой способ присваивания "по цепочке" удобен для задания общего значения целой группе переменных.

Составные операторы присваивания

В C# предусмотрены специальные составные операторы присваивания, упрощающие программирование некоторых операций присваивания. Обратимся сначала к простому примеру:

```
x = x + 1;
```

// Можно переписать следующим образом:

```
x += 1;
```

Пара операторов += указывает компилятору на то, что переменной x должно быть присвоено ее первоначальное значение, увеличенное на 1.

Для многих двоичных операций, т.е. операций, требующих наличия двух операндов, существуют отдельные составные операторы присваивания. Общая форма всех этих операторов имеет следующий вид:

*имя переменной op = выражение*

где op — арифметический или логический оператор, применяемый вместе с оператором присваивания. Ниже перечислены составные операторы присваивания для арифметических и логических операций:

<b>Оператор</b>	<b>Аналог (выражение из вышеуказанного примера)</b>
+=	x = x + 1;
-=	x = x - 1;
*=	x = x*1;
/=	x = x/1;
%=	x = x%1;
=	x = x   1;
^=	x = x^1;

Составные операторы присваивания записываются более кратко, чем их несоставные эквиваленты. Поэтому их иногда еще называют *укороченными операторами присваивания*.

У составных операторов присваивания имеются два главных преимущества. Во-первых, они более компактны, чем их "несокращенные" эквиваленты. И во-вторых, они дают более эффективный исполняемый код, поскольку левый операнд этих операторов вычисляется только один раз. Именно по этим причинам составные операторы присваивания чаще всего применяются в программах, профессионально написанных на C#.

## 6. УСЛОВНЫЕ ОПЕРАТОРЫ

Условные операторы позволяют управлять *потоком* выполнения программы, чтобы не выполнялась каждая строка кода, как она следует в программе.

Оператор `if`

Для организации условного ветвления язык C# унаследовал от C и C++ конструкцию `if...else`. Ее синтаксис должен быть интуитивно понятен для любого, кто программировал на процедурных языках:

*if* (*условие*)

*оператор (операторы)*

*else*

*оператор (операторы)*

Если по каждому из условий нужно выполнить более одного оператора, эти операторы должны быть объединены в блок с помощью фигурных скобок `{...}`.

В C# условный оператор `if` может работать только с булевыми выражениями, но не с произвольными значениями вроде `-1` и `0`.

В операторе `if` могут применяться сложные выражения, и он может содержать операторы `else`, обеспечивая выполнение более сложных проверок.

При построении сложных выражений в C# используется вполне ожидаемый набор логических операторов.

### Пример:

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string myStr;
            Console.WriteLine("Введите строку: ");

            myStr = Console.ReadLine();

            if (myStr.Length < 5)
                Console.WriteLine("\nВ данной строке меньше 5 символов");
            else if ((myStr.Length >= 5) && (myStr.Length <= 12))
                Console.WriteLine("\nВ данной строке {0} символов",myStr.Length);
            else Console.WriteLine("\nВ данной строке больше 12 символов");

            Console.ReadLine();
        }
    }
}
```



Рисунок 4. Пример работы условного оператора if-else

Количество `else if`, добавляемых к единственному `if`, не ограничено. Следует отметить касательно `if`: фигурные скобки применять не обязательно, если в условной ветви присутствует только один оператор, как показано в исходном примере.

## 7. ЦИКЛЫ FOR И WHILE

В C# имеются четыре различных вида циклов (`for`, `while`, `do...while` и `foreach`), позволяющие выполнять блок кода повторно до тех пор, пока удовлетворяется определенное условие.

### Цикл `for`

Цикл `for` в C# предоставляет механизм итерации, в котором определенное условие проверяется перед выполнением каждой итерации. Синтаксис этого оператора показан ниже:

*for (инициализатор; условие; итератор)*

*оператор (операторы)*

Здесь:

Инициализатор - это выражение, вычисляемое перед первым выполнением тела цикла (обычно инициализация локальной переменной в качестве счетчика цикла). Инициализация представлена оператором присваивания, задающим первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом;

Условие - это выражение, проверяемое перед каждой новой итерацией цикла (должно возвращать `true`, чтобы была выполнена следующая итерация);

Итератор - выражение, вычисляемое после каждой итерации (обычно приращение значения счетчика цикла).

Эти три основные части оператора цикла `for` должны быть разделены точкой с запятой. Выполнение цикла `for` будет продолжаться до тех пор, пока проверка условия дает истинный результат. Как только эта проверка даст

ложный результат, цикл завершится, а выполнение программы будет продолжено с оператора, следующего после цикла for.

Цикл for отлично подходит для повторного выполнения оператора или блока операторов заранее известное количество раз.

### **Пример применения цикла for:**

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

namespace ConsoleApplication1
{
    class Program
    {
        // Данный метод выводит таблицу умножения
        // размерностью b x b
        static void tab(byte b)
        {
            Console.WriteLine("Таблица умножения {0} x {0}\n", b);
            // Этот цикл проходит по строкам
            for (int i = 1; i <= b; i++)
            {
                // Этот цикл проходит по столбцам
                for (int j = 1; j <= b; j++)
                    Console.Write("{0}\t", j * i);
                Console.WriteLine();
            }
            Console.WriteLine();
        }

        static void Main(string[] args)
        {
            tab(8);

            // Давайте разберем нестандартные возможности цикла for
            // ***** //
        }
    }
}
```



```

// Применение нескольких переменных управления циклом
for (byte i = 0, j = 20; i <= j; i += 5, j -= 5)
    Console.WriteLine("i = {0}, j = {1}",i,j);
Console.WriteLine();

// Использование условного выражения в цикле
bool b = false;
for (byte i = 1, j = 100; !b; i++, j--)
    if (i < Math.Sqrt(j))
        Console.WriteLine("Число {0} меньше квадратного корня из {1}", i, j);
        else b = true;

// Отсутствие части цикла
int k = 0;
for (; k < 10; )
{
    k++;
    Console.Write(k);
}
Console.WriteLine("\n");

// Цикл без тела
int sum = 0;
for (int i = 1; i <= 10; sum += ++i);

Console.WriteLine("Значение суммы: {0}",sum);

Console.ReadLine();
}
}

```

```

file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
Таблица умножения 8 x 8
1      2      3      4      5      6      7      8
2      4      6      8      10     12     14     16
3      6      9      12     15     18     21     24
4      8      12     16     20     24     28     32
5      10     15     20     25     30     35     40
6      12     18     24     30     36     42     48
7      14     21     28     35     42     49     56
8      16     24     32     40     48     56     64

i = 0, j = 20
i = 5, j = 15
i = 10, j = 10
Число 1 меньше квадратного корня из 100
Число 2 меньше квадратного корня из 99
Число 3 меньше квадратного корня из 98
Число 4 меньше квадратного корня из 97
Число 5 меньше квадратного корня из 96
Число 6 меньше квадратного корня из 95
Число 7 меньше квадратного корня из 94
Число 8 меньше квадратного корня из 93
Число 9 меньше квадратного корня из 92
12345678910
Значение суммы: 65
}

```

Рисунок 5. Пример работы цикла for

## Цикл while

Подобно for, while также является циклом с предварительной проверкой. Синтаксис его аналогичен, но циклы while включают только одно выражение: *while(условие)*

*оператор (операторы);*

где *оператор* — это единственный оператор или же блок операторов, а *условие* означает конкретное условие управления циклом и может быть любым логическим выражением. В этом цикле оператор выполняется до тех пор, пока условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла.

Как и в цикле for, в цикле while проверяется условное выражение, указываемое в самом начале цикла. Это означает, что код в теле цикла может вообще не выполняться, а также избавляет от необходимости выполнять отдельную проверку перед самим циклом.

### Пример:

```

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

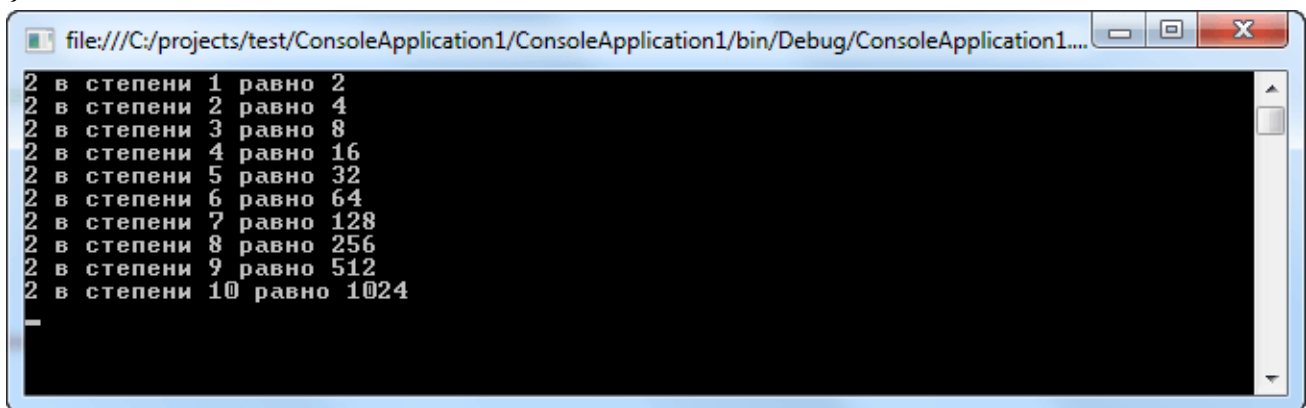
```

```
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Пример возведения числа в несколько степеней
            byte l = 2, i = 0;
            int result = 1;

            while (i < 10)
            {
                i++;
                result *= l;
                Console.WriteLine("{0} в степени {1} равно {2}",l,i,result);
            }

            Console.ReadLine();
        }
    }
}
```



```
file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1....
2 в степени 1 равно 2
2 в степени 2 равно 4
2 в степени 3 равно 8
2 в степени 4 равно 16
2 в степени 5 равно 32
2 в степени 6 равно 64
2 в степени 7 равно 128
2 в степени 8 равно 256
2 в степени 9 равно 512
2 в степени 10 равно 1024
```

Рисунок 6. Возведение числа в несколько степеней

## 8. ЦИКЛЫ DO...WHILE И FOREACH

### Цикл do . . . while

Цикл *do...while* — это версия *while* с постпроверкой условия. Условие цикла проверяется после выполнения тела цикла. Следовательно, циклы *do...while* удобны в тех ситуациях, когда блок операторов должен быть выполнен как минимум однажды. Ниже приведена общая форма оператора цикла *do...while*:

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

При наличии лишь одного оператора фигурные скобки в данной форме записи необязательны. Они зачастую используются для того, чтобы сделать конструкцию *do-while* более удобно читаемой и не путать ее с конструкцией цикла *while*. Цикл *do-while* выполняется до тех пор, пока условное выражение истинно. Цикл *do-while* используется в программе, рассчитывающую факториал числа.

#### Пример:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace ConsoleApplication1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            try  
            {  
                // Вычисляем факториал числа  
                int i, result = 1, num = 1;
```

```

Console.WriteLine("Введите число:");
i = int.Parse(Console.ReadLine());

Console.Write("\n\nФакториал {0} = ", i);
do
{
    result *= num;
    num++;
} while (num <= i);

Console.Write(result);
}
catch (FormatException ex)
{
    Console.WriteLine("Вы ввели не число. {0}", ex.Message);
}
finally
{
    Console.ReadLine();
}
}
}
}

```

## Цикл foreach

*Цикл foreach* служит для циклического обращения к элементам коллекции, представляющей собой группу объектов. В С# определено несколько видов коллекций, каждая из которых является массивом.

Общая форма оператора цикла foreach:

```

foreach (тип имя_переменной_цикла in коллекция)
    оператор;

```

Здесь *тип имя\_переменной\_цикла* обозначает тип и имя переменной управления циклом, которая получает значение следующего элемента коллекции на каждом шаге выполнения цикла foreach. А коллекция обозначает циклически опрашиваемую коллекцию, которая здесь и далее представляет собой массив. Следовательно, тип переменной цикла должен соответствовать

типу элемента массива. Тип может обозначаться ключевым словом `var`. В этом случае компилятор определяет тип переменной цикла, исходя из типа элемента массива. Это может оказаться полезным для работы с определенными запросами. Как правило, тип указывается явным образом.

Оператор цикла `foreach` действует следующим образом. Когда цикл начинается, первый элемент массива выбирается и присваивается переменной цикла. На каждом последующем шаге итерации выбирается следующий элемент массива, который сохраняется в переменной цикла. Цикл завершается, когда все элементы массива окажутся выбранными.

Цикл `foreach` позволяет проходить по каждому элементу коллекции (объект, представляющий список других объектов). Формально для того, чтобы нечто можно было рассматривать как коллекцию, это нечто должно поддерживать интерфейс `IEnumerable`. Примерами коллекций могут служить массивы `C#`, классы коллекций из пространства имен `System.Collection`, а также пользовательские классы коллекций.

### **Пример:**

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Объявляем два массива
            int[] myArr = new int[5];
            int[,] myTwoArr = new int[5, 6];
            int sum = 0;

            Random ran = new Random();
```

```

// Инициализируем массивы
for (int i = 1; i <= 5; i++)
{
    myArr[i-1] = ran.Next(1, 20);
    for (int j = 1; j <= 6; j++)
        myTwoArr[i - 1, j - 1] = ran.Next(1, 30);
}

// Вычисляем квадрат каждого элемента одномерного массива
foreach (int fVar in myArr)
    Console.WriteLine("{0} в квадрате равно {1}", fVar, fVar*fVar);

Console.WriteLine();

// Вычислим сумму элементов многомерного массива
foreach (int fTwoVar in myTwoArr)
    sum += fTwoVar;

Console.WriteLine("Сумма элементов многомерного массива: {0}", sum);

Console.ReadLine();
}
}
}

```

Если запустить данный пример несколько раз, то можно наглядно увидеть, что элементы массива изменяются каждый раз (с помощью метода `Random.Next`), и соответственно опрашиваются в цикле `foreach`.

```

file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
13 в квадрате равно 169
17 в квадрате равно 289
12 в квадрате равно 144
9 в квадрате равно 81
7 в квадрате равно 49

Сумма элементов многомерного массива: 487

```

Рисунок 7. Пример работы цикла `foreach`

## 9. МЕТОДЫ

В контексте объектно-ориентированного программирования, метод – это подпрограмма. Подпрограмма– это фрагмент программы, который написать нужно один раз, а использовать его можно многократно. Это удобно по нескольким причинам, например, можно реализовать определенный алгоритм один раз, хорошо его отладить, и быть уверенным в правильности его работы применяя его в дальнейшем многократно. Использование подпрограмм значительно сокращает объем исходного кода проекта.

### Объявление методов

В C# определение метода состоит из любых модификаторов (таких как спецификация доступности), типа возвращаемого значения, за которым следует имя метода, затем список аргументов в круглых скобках и далее - тело метода в фигурных скобках:

```
[модификаторы] тип_возврата ИмяМетода([параметры])  
{  
// Тело метода  
}
```

Каждый параметр состоит из имени типа параметра и имени, по которому к нему можно обратиться в теле метода. Если метод возвращает значение, то для указания точки выхода должен использоваться оператор возврата `return` вместе с возвращаемым значением.

Если метод не возвращает ничего, то в качестве типа возврата указывается **void**, поскольку вообще опустить тип возврата невозможно. Если же он не принимает аргументов, то все равно после имени метода должны присутствовать пустые круглые скобки. При этом включать в тело метода оператор возврата не обязательно - метод возвращает управление автоматически по достижении закрывающей фигурной скобки.

### Возврат из метода и возврат значения

Возврат из метода может произойти при двух условиях. Во-первых, когда встречается фигурная скобка, закрывающая тело метода. И во-вторых, когда



выполняется оператор **return**. Имеются две формы оператора return: одна — для методов типа void (*возврат из метода*), т.е. тех методов, которые не возвращают значения, а другая — для методов, возвращающих конкретные значения (*возврат значения*).

### Пример:

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

namespace ConsoleApplication1
{
    class MyMathOperation
    {
        public double r;
        public string s;

        // Возвращает площадь круга
        public double sqrCircle()
        {
            return Math.PI * r * r;
        }

        // Возвращает длину окружности
        public double longCircle()
        {
            return 2 * Math.PI * r;
        }

        public void writeResult()
        {
            Console.WriteLine("Вычислить площадь или длину? s/l:");
            s = Console.ReadLine();
            s = s.ToLower();
            if (s == "s")
            {
                Console.WriteLine("Площадь круга равна {0:#.###}",sqrCircle());
            }
        }
    }
}
```

```

        return;
    }
    else if (s == "l")
    {
        Console.WriteLine("Длина окружности равна {0:#.##}",longCircle());
        return;
    }
    else
    {
        Console.WriteLine("Вы ввели не тот символ");
    }
}
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Введите радиус: ");
        string radius = Console.ReadLine();

        MyMathOperation newOperation = new MyMathOperation { r =
double.Parse(radius) };
        newOperation.writeResult();

        Console.ReadLine();
    }
}
}

```

```

file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
Введите радиус:
5
Вычислить площадь или длину? s/l:
l
Длина окружности равна 31,42

```

Рисунок 8. Вычисление площади/длины окружности

## Использование параметров

При вызове метода ему можно передать одно или несколько значений. Значение, передаваемое методу, называется *аргументом*. А переменная, получающая аргумент, называется формальным параметром, или просто *параметром*. Параметры объявляются в скобках после имени метода. Синтаксис объявления параметров такой же, как и у переменных. А областью действия параметров является тело метода. За исключением особых случаев передачи аргументов методу, параметры действуют так же, как и любые другие переменные.

В общем случае параметры могут передаваться методу либо по значению, либо по ссылке. Когда переменная передается по ссылке, вызываемый метод получает саму переменную, поэтому любые изменения, которым она подвергнется внутри метода, останутся в силе после его завершения. Но если переменная передается по значению, вызываемый метод получает копию этой переменной, а это значит, что все изменения в ней по завершении метода будут утеряны. Для сложных типов данных передача по ссылке более эффективна из-за большого объема данных, который приходится копировать при передаче по значению.

### **Пример:**

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

namespace ConsoleApplication1
{
    class myClass
    {
        public void someMethod(double[] myArr, int i )
        {
            myArr[0] = 12.0;
            i = 12;
        }
    }
}
```

```

}
class Program
{
    static void Main(string[] args)
    {
        double[] arr1 = { 0, 1.5, 3.9, 5.1 };
        int i = 0;
        Console.WriteLine("Массив arr1 до вызова метода: ");
        foreach (double d in arr1)
            Console.Write("{0}\t",d);
        Console.WriteLine("\nПеременная i = {0}\n",i);

        Console.WriteLine("Вызов метода someMethod ...");
        myClass ss = new myClass();
        ss.someMethod(arr1,i);
        Console.WriteLine("Массив arr1 после вызова метода:");
        foreach (double d in arr1)
            Console.Write("{0}\t",d);
        Console.WriteLine("\nПеременная i = {0}\n",i);
        Console.ReadLine();
    }
}

```

```

file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
Массив arr1 до вызова метода:
0      1,5      3,9      5,1
Переменная i = 0

Вызов метода someMethod ...
Массив arr1 после вызова метода:
12     1,5     3,9     5,1
Переменная i = 0

```

Рисунок 9. Вызов метода

Значение *i* осталось неизменным, но измененные значения в *myArg* также изменились в исходном массиве *arr1*, так как массивы являются ссылочными типами.

Поведение строк также отличается. Дело в том, что строки являются неизменными, поэтому строки не демонстрируют поведение, характерное для

ссылочных типов. Любые изменения, проведенные в строке внутри метода, не влияют на исходную строку.

## 10. МАССИВЫ

*Массив* представляет собой совокупность переменных одного типа с общим для обращения к ним именем. В С# массивы могут быть как одномерными, так и многомерными. Массивы служат самым разным целям, поскольку они предоставляют удобные средства для объединения связанных вместе переменных.

Массивами в С# можно пользоваться практически так же, как и в других языках программирования. Тем не менее, у них имеется одна особенность: они реализованы в виде объектов.

Для того чтобы воспользоваться массивом в программе, требуется двухэтапная процедура, поскольку в С# массивы реализованы в виде объектов. Во-первых, необходимо объявить переменную, которая может обращаться к массиву. И во-вторых, нужно создать экземпляр массива, используя оператор `new`.

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

```
namespace ConsoleApplication1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            // Объявляем массив  
            int[] myArr = new int[5];
```

```
            // Инициализируем каждый элемент массива вручную
```

```

myArr[0] = 100;
myArr[1] = 23;
myArr[2] = 25;
myArr[3] = 31;
myArr[4] = 1;

foreach (int i in myArr)
    Console.WriteLine(i);

    Console.ReadLine();
}
}
}

```

Следует иметь в виду, что если массив только объявляется, но явно не инициализируется, каждый его элемент будет установлен в значение, принятое по умолчанию для соответствующего типа данных (например, элементы массива типа `bool` будут устанавливаться в `false`, а элементы массива типа `int` — в `0`).

#### Инициализация массива

Помимо заполнения массива элемент за элементом (как показано в предыдущем примере), можно также заполнять его с использованием специального синтаксиса инициализации массивов. Для этого необходимо перечислить включаемые в массив элементы в фигурных скобках `{ }`. Такой синтаксис удобен при создании массива известного размера, когда нужно быстро задать его начальные значения:

*// Синтаксис инициализации массива с использованием*

*// ключевого слова new*

```
int[] myArr = new int[] { 10,20,30,40,50};
```

*// Синтаксис инициализации массива без использования*

*// ключевого слова new*

```
string[] info = { "Фамилия", "Имя", "Отчество" };
```

*// Используем ключевое слово new и желаемый размер*

```
char[] symbol = new char[4] { 'X','Y','Z','M' };
```

В случае применения синтаксиса с фигурными скобками размер массива указывать не требуется (как видно на примере создания переменной myArr), поскольку этот размер автоматически вычисляется на основе количества элементов внутри фигурных скобок. Кроме того, применять ключевое слово *new* не обязательно (как при создании массива info).

### Неявно типизированные массивы

Ключевое слово *var* позволяет определить переменную так, чтобы лежащий в ее основе тип выводился компилятором. Аналогичным образом можно также определять неявно типизированные локальные массивы. С использованием такого подхода можно определить новую переменную массива без указания типа элементов, содержащихся в массиве.

#### **Пример:**

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            var arr1 = new[] { 1, 2, 3 };
            Console.WriteLine("Тип массива arr1 - {0}",arr1.GetType());

            var arr2 = new[] { "One", "Two", "Three" };
            Console.WriteLine("Тип массива arr2 - {0}",arr2.GetType());
        }
    }
}
```

```

        Console.ReadLine();
    }
}

```

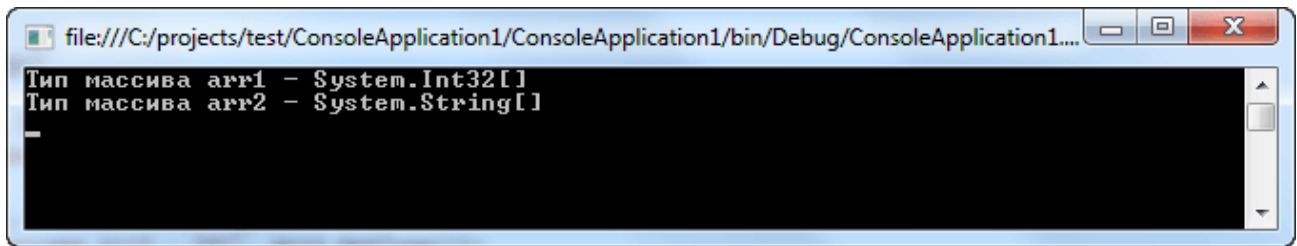


Рисунок 10. Неявно типизированные массивы

При создании массива с использованием явного синтаксиса C#, элементы, указываемые в списке инициализации массива, должны обязательно иметь один и тот же базовый тип (т.е. должны все быть int, string или MyCar).

### Определение массива объектов

В большинстве случаев при определении массива тип элемента, содержащегося в массиве, указывается явно. Хотя на первый взгляд это выглядит довольно понятно, существует одна важная особенность. В основе каждого типа в системе типов .NET (в том числе фундаментальных типов данных) в конечном итоге лежит базовый класс System.Object. В результате получается, что в случае определения массива объектов находящиеся внутри него элементы могут представлять собой что угодно:

```

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)

```



```

{
    // Объявляем и инициализируем массив объектов
    object[] arrByObject = { true, 10, "Привет", 13.7m };

    // Выведем в консоль тип каждого члена массива
    foreach (object me in arrByObject)
        Console.WriteLine("Тип {0} - {1}",me,me.GetType());

    Console.ReadLine();
}
}
}

```

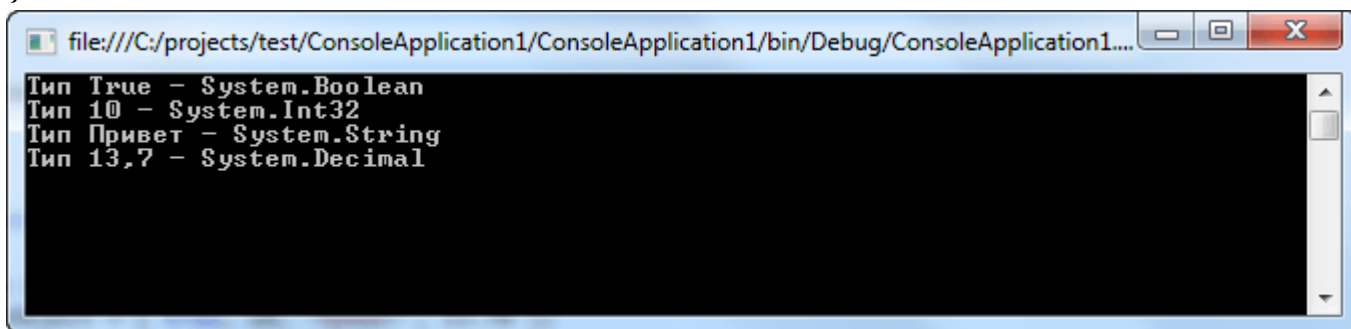


Рисунок 11. Определение массива объектов

## Свойство Length

Реализация в C# массивов в виде объектов дает целый ряд преимуществ. Одно из них заключается в том, что с каждым массивом связано *свойство Length*, содержащее число элементов, из которых может состоять массив. Следовательно, у каждого массива имеется специальное свойство, позволяющее определить его длину.

Когда запрашивается длина многомерного массива, то возвращается общее число элементов, из которых может состоять массив. Благодаря наличию у массивов свойства Length операции с массивами во многих алгоритмах становятся более простыми, а значит, и более надежными.

### Пример:

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] myArr = { 1, 2, 3, 4 };

            for (int i = 0; i < myArr.Length; i++)
                Console.WriteLine(myArr[i]);

            Console.ReadLine();
        }
    }
}

```

## 11. ОПЕРАТОРЫ ПЕРЕХОДА

Язык C# предлагает несколько операторов, позволяющих немедленно перейти на другую строку программы.

### Оператор goto

Имеющийся в C# оператор goto представляет собой оператор безусловного перехода. Когда в программе встречается оператор goto, ее выполнение переходит непосредственно к тому месту, на которое указывает этот оператор. Он уже давно "вышел из употребления" в программировании, поскольку способствует созданию "макаронного" кода. Хотя в некоторых случаях он оказывается удобным и дает определенные преимущества, если используется благоразумно. Главный недостаток оператора goto с точки зрения программирования заключается в том, что он вносит в программу беспорядок и

делает ее практически неудобочитаемой. Но иногда применение оператора goto может, скорее, прояснить, чем запутать ход выполнения программы.

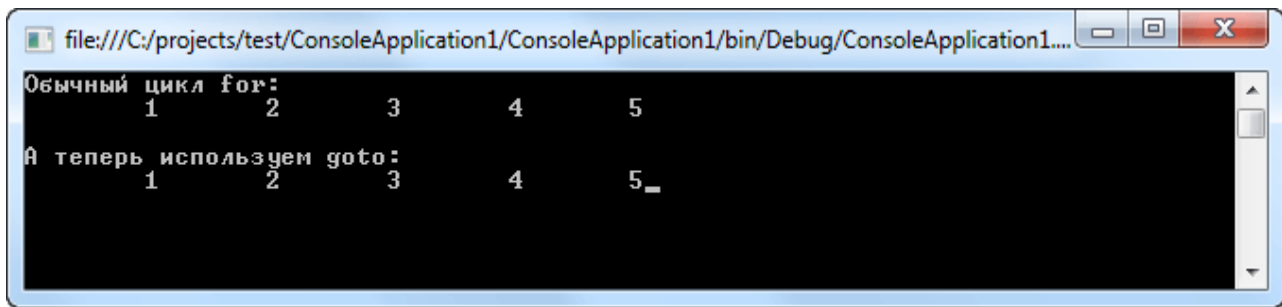
Для выполнения оператора goto требуется *метка* — действительный в C# идентификатор с двоеточием. Метка должна находиться в том же методе, где и оператор goto, а также в пределах той же самой области действия.

### **Пример:**

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Обычный цикл for выводящий числа от 1 до 5
            Console.WriteLine("Обычный цикл for:");
            for (int i = 1; i <= 5; i++)
                Console.Write("\t{0}",i);
            // Реализуем то же самое с помощью оператора goto
            Console.WriteLine("\n\nА теперь используем goto:");
            int j = 1;
            link1:
                Console.Write("\t{0}",j);
                j++;
                if (j <= 5) goto link1;

            Console.ReadLine();
        }
    }
}
```



```
file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1....
Обычный цикл for:
1      2      3      4      5
А теперь используем goto:
1      2      3      4      5_
```

Рисунок 12. Вывод чисел с помощью For и Goto

Репутация оператора goto такова, что в большинстве случаев его применение категорически осуждается. Вообще говоря, он, конечно, не вписывается в рамки хорошей практики объектно-ориентированного программирования.

## Оператор break

С помощью оператора break можно специально организовать немедленный выход из цикла в обход любого кода, оставшегося в теле цикла, а также минуя проверку условия цикла. Когда в теле цикла встречается оператор break, цикл завершается, а выполнение программы возобновляется с оператора, следующего после этого цикла. Оператор break можно применять в любом цикле, предусмотренном в C#.

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

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // В данном цикле выведутся числа от 1 до 5 вместо 100
            for (int i = 1; i < 100; i++)
                if (i <= 5)
                    Console.WriteLine(i);
        }
    }
}
```

```

        else break;

        Console.ReadLine();
    }
}
}

```

Если оператор `break` применяется в целом ряде вложенных циклов, то он прерывает выполнение только самого внутреннего цикла.

В отношении оператора `break` необходимо также иметь в виду следующее. Во-первых, в теле цикла может присутствовать несколько операторов `break`, но применять их следует очень аккуратно, поскольку чрезмерное количество операторов `break` обычно приводит к нарушению нормальной структуры кода. И во-вторых, оператор `break`, выполняющий выход из оператора `switch`, оказывает воздействие только на этот оператор, но не на объемлющие его циклы.

## Оператор `continue`

С помощью оператора `continue` можно организовать преждевременное завершение шага итерации цикла в обход обычной структуры управления циклом. Оператор `continue` осуществляет принудительный переход к следующему шагу цикла, пропуская любой код, оставшийся невыполненным. Таким образом, оператор `continue` служит своего рода дополнением оператора `break`.

В циклах `while` и `do-while` оператор `continue` вызывает передачу управления непосредственно условному выражению, после чего продолжается процесс выполнения цикла. А в цикле `for` сначала вычисляется итерационное выражение, затем условное выражение, после чего цикл продолжается:

```

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

```

```

namespace ConsoleApplication1

```

```

{
class Program
{
    static void Main(string[] args)
    {
        // Выводим числа кратные 5
        for (byte i = 1; i <= 100; i++)
        {
            if (i % 5 != 0) continue;
            Console.Write("\t{0}", i);
        }

        Console.ReadLine();
    }
}
}

```

Результат:

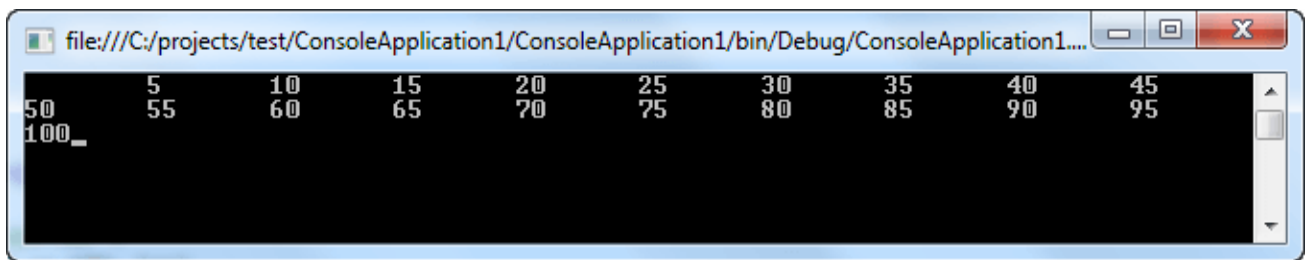


Рисунок 13. Вывод чисел кратных 5

Оператор `continue` редко находит удачное применение, в частности, потому, что в `C#` предоставляется богатый набор операторов цикла, удовлетворяющих большую часть прикладных потребностей. Но в тех особых случаях, когда требуется преждевременное прерывание шага итерации цикла, оператор `continue` предоставляет структурированный способ осуществления такого прерывания.

### Оператор `return`

Оператор `return` организует возврат из метода. Его можно также использовать для возврата значения. Имеются две формы оператора `return`: одна

— для методов типа `void`, т.е. тех методов, которые не возвращают значения, а другая — для методов, возвращающих конкретные значения.

Для немедленного завершения метода типа `void` достаточно воспользоваться следующей формой оператора `return`:

```
return;
```

Когда выполняется этот оператор, управление возвращается вызывающей части программы, а оставшийся в методе код пропускается.

Для возврата значения из метода в вызывающую часть программы служит следующая форма оператора `return`:

```
return значение;
```

### **Пример:**

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

```
namespace ConsoleApplication1
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            int result = Sum(230);
```

```
            Console.WriteLine("Сумма четных чисел от 1 до 230 равна: " + result);
```

```
            Console.ReadLine();
```

```
        }
```

```
// Метод, возвращающий сумму всех четных чисел
```

```
// от 1 до s
```

```
static int Sum(int s)
```

```
{
```

```
    int mySum = 0;
```

```
    for (int i = 1; i <= s; i++)
```

```
        if (i % 2 == 0)
```

```
        mySum += i;  
    return mySum;  
    }  
}  
}
```

Результат работы данной программы:

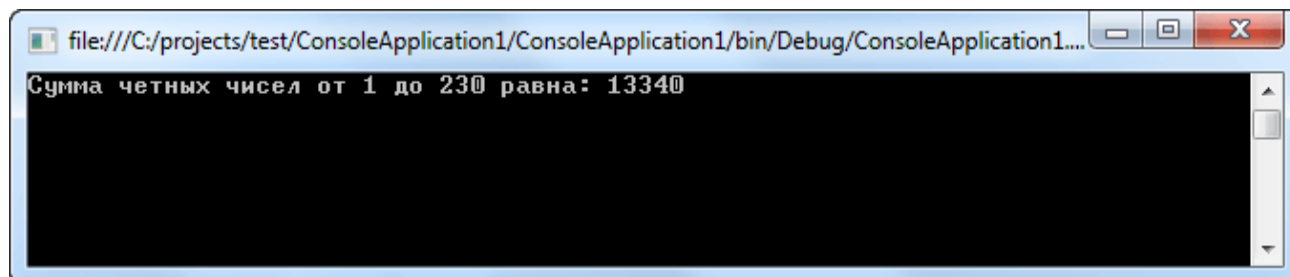


Рисунок 14. Сумма четных чисел от 1 до S



## Примеры решения типовых задач

1. Найти максимальный элемент массива, состоящего из  $n$  элементов.

```
1  using System;
2  namespace MyProgram
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              int n = int.Parse(Console.ReadLine());
9              int [] arr = new int [n];
10             for (int i = 0; i < n; i++)
11                 arr[i] = int.Parse(Console.ReadLine());
12             int max = arr[0];
13             for (int i = 0; i < n; i++)
14                 if (arr[i] > max) max = arr[i];
15             Console.WriteLine(max);
16             Console.ReadKey();
17         }
18     }
19 }
```

2. Дано целое число  $K$ . Вывести строку-описание оценки, соответствующий числу  $K$  (1-плохо,2-неудовлетворительно,3-удовлетворительно,4-хорошо,5-отлично). Если  $K$  не лежит в диапазоне 1-5, то вывести строку "Ошибка".

Оператор выбора (Case)

```
Console.WriteLine("Введите K");
int K = int.Parse(Console.ReadLine());
string s;
switch (K)
{
    case 1: s = "плохо"; break;
    case 2: s = "неудовлетворительно"; break;
    case 3: s = "удовлетворительно"; break;
    case 4: s = "хорошо"; break;
    case 5: s = "отлично"; break;
    default: s = "Ошибка"; break;
}
Console.WriteLine(s);
Console.ReadLine();
```

3. Используя VisualStudio, создайте проект по шаблону ConsoleApplication.

Дано значение числа  $\pi$ , которое равно 3,141592653 и значение числа Эйлера  $e$ , которое равно 2,7182818284590452. Создайте две переменные, присвойте им значения числа  $\pi$  и числа  $e$  и выведите их на экран без потери точности.

```
{
class Program
{
    static void Main()
    {
        double pi = 3.141592653;
        decimal e = 2.7182818284590452m;

        Console.WriteLine(pi);
        Console.WriteLine(e);

        Console.ReadKey();
    }
}
```

4. Используя VisualStudio, создайте проект по шаблону ConsoleApplication. Создайте три строковые переменные и присвойте им значения:

"\nмоя строка 1"

"\tмоя строка 2"

"\амоя строка 3"

Выведите значение каждой переменной на экран.

```
{
class Program
{
    static void Main()
    {
        string firstString = "\nмоя строка 1";
        String secondString = "\tмоя строка 2";
        string thirdString = "\амоя строка 3";

        Console.WriteLine(firstString);
        Console.WriteLine(secondString);
        Console.WriteLine(thirdString);

        Console.ReadKey();
    }
}
```

5. Используя VisualStudio, создайте проект по шаблону ConsoleApplication. Создайте две целочисленные переменные и выведите на экран результаты всех арифметических операций над этими двумя переменными.

```

class Program
{
    static void Main()
    {
        int a = 12, b = 5;
        int sum = 0;
        sum = a + b;

        Console.WriteLine(sum);

        int difference = 0;
        difference = a - b;

        Console.WriteLine(difference);

        int product = 0;
        product = a * b;

        Console.WriteLine(product);

        int quotient = 0, remainder = 0;
        quotient = a / b;

        Console.WriteLine(quotient);

        Console.ReadKey();
    }
}

```

6. Имеется 3 переменные типа *int*:  $x = 10$ ,  $y = 12$ , и  $z = 3$ . Выполните и рассчитайте результат следующих операций для этих переменных:

- i.  $x += y - x++ * z;$
- ii.  $z = --x - y * 5;$
- iii.  $y /= x + 5 \% z;$
- iv.  $z = x++ + y * 5;$
- v.  $x = y - x++ * z;$

```

static void Main()
{
    int x = 10, y = 12, z = 3;
    int operazia1 = 0;
    operazia1 = (x += y - x++ * z);

    Console.WriteLine(operazia1);

    x = 10; y = 12; z = 3;
    int operazia2 = 0;
    operazia2 = (z = --x - y * 5);

    Console.WriteLine(operazia2);

    x = 10; y = 12; z = 3;
    int operazia3 = 0;
    operazia3 = (y /= x + 5 % z);

    Console.WriteLine(operazia3);

    x = 10; y = 12; z = 3;
    int operazia4 = 0;
    operazia4 = (z = x++ + y * 5);

    Console.WriteLine(operazia4);

    x = 10; y = 12; z = 3;
    int operazia5 = 0;
    operazia5 = (x = y - x++ * z);

    Console.WriteLine(operazia5);

    Console.ReadKey();
}
}

```

7. Вычислите среднее арифметическое трех целочисленных значений и выведите его на экран.

```

class Program
{
    static void Main()
    {
        int a = 3, b = 6, c = 5;
        int quotient = 0;
        quotient = (a + b + c) / 3;

        Console.WriteLine(quotient);

        Console.ReadKey();
    }
}

```

8. Создайте константу с именем –  $\pi$  (число  $\pi$ ), создайте переменную радиус с именем –  $r$ . Используя формулу  $\pi R^2$ , вычислите площадь круга и выведите результат на экран.

```
{
class Program
{
    static void Main()
    {
        const double pi = 3.141;
        double r = 4;
        double result = pi * Math.Pow(r, 2);

        Console.WriteLine(result);

        Console.ReadKey();
    }
}
```

9. Напишите программу расчета объема –  $V$  и площади поверхности –  $S$  цилиндра. Объем  $V$  цилиндра радиусом –  $R$  и высотой –  $h$ :  $V = \pi R^2 h$ .  
Площадь  $S$  поверхности цилиндра:  $S = 2\pi R^2 + 2\pi R h = 2\pi R(R+h)$ .  
Результаты расчетов выведите на экран.

```
{
class Program
{
    static void Main()
    {
        const double pi = 3.141;
        double R = 1.5, h = 2;
        double V = pi * Math.Pow(R, 2) * h;
        double S = (2 * pi * Math.Pow(R, 2)) + (2 * pi * R * h);

        Console.WriteLine(V);
        Console.WriteLine(S);

        Console.ReadKey();
    }
}
```

10. Напишите программу определения, попадает ли указанное пользователем число от 0 до 100 в числовой промежуток [0 - 14] [15 - 35] [36 - 50] [51 - 100]. Если да, то укажите, в какой именно промежуток. Если пользователь указывает число не входящее ни в один из имеющихся числовых промежутков, то выводится соответствующее сообщение.

```

class Program
{
    static void Main()
    {
        Console.WriteLine("Введите число от 0 до 100");
        // строковый тип преобразуем в числовой
        int chislo = int.Parse(Console.ReadLine());

        // используем условный оператор if-else
        if (0 <= chislo && chislo <= 14)
        {
            Console.WriteLine("Это число принадлежит промежутку [0-14]"); // Ветвь 1
        }
        else if (15 <= chislo && chislo <= 50)
        {
            Console.WriteLine("Это число принадлежит промежутку [15-50]"); // Ветвь 2
        }
        else if (51 <= chislo && chislo <= 100)
        {
            Console.WriteLine("Это число принадлежит промежутку [51-100]"); // Ветвь 3
        }
        else
        {
            Console.WriteLine("Введите число от 0 до 100!"); // Ветвь 4
        }

        // Delay.
        Console.ReadKey();
    }
}

```

11. Известно, что у чисел, которые являются степенью двойки, только один бит имеет значение 1.

Используя VisualStudio, создайте проект по шаблону ConsoleApplication.

Напишите программу, которая будет выполнять проверку – является ли указанное число степенью двойки или нет.

```

class Program
{
    static void Main()
    {
        stепен:
        Console.WriteLine("Введите число");
        // Принимаем значение a
        int a = int.Parse(Console.ReadLine());
        // Вводим новую переменную b
        int b = 0;
        // обнуляет крайний справа единичный бит.
        b = a & (a - 1);
        if (b == 0) // Если результат равен 0 - значит, b имеет вид 100...00, т.е. представляет собой степень 2
        {
            Console.WriteLine("Данное число - степень двойки");
        }
        else
        {
            Console.WriteLine("Данное число - не степень двойки");
        }

        goto stепен;
        Console.ReadKey();
    }
}

```

12. Используя теорему Де Моргана, преобразуйте исходное выражение  $A \mid B$ , в эквивалентное выражение.

```
class Program
{
    static void Main()
    {
        bool A = true;
        bool B = false;

        // Условие до применения теоремы Де Моргана.
        if (A || B)
            Console.WriteLine("A || B = {0}", A || B);
        else
            Console.WriteLine("A || B = {0}", A || B);

        // Условие после применения теоремы Де Моргана.
        if (!(!A && !B))
            Console.WriteLine("!(!A && !B) = {0}", !(!A && !B));
        else
            Console.WriteLine("!(!A && !B) = {0}", !(!A && !B));

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

13. Известно, что у четных чисел младший бит имеет значение 0. Используя VisualStudio, создайте проект по шаблону ConsoleApplication. Напишите программу, которая будет выполнять проверку чисел на четность.

```
class Program
{
    static void Main()
    {
        Console.WriteLine("Введите число");
        byte x = byte.Parse(Console.ReadLine());

        if (x % 2 == 1)
            Console.WriteLine("Данное число - нечетное");
        else
            Console.WriteLine("Данное число - четное");

        Console.ReadKey();
    }
}
```

14. Имеется 3 переменные типа *int*:  $x = 5$ ,  $y = 10$ , и  $z = 15$ . Выполните и рассчитайте результат следующих операций для этих переменных:

- $x += y >> x++ * z$ ;
- $z = ++x \& y * 5$ ;
- $y /= x + 5 | z$ ;

- $z = x++ \ \& \ y * 5;$
- $x = y \ll x++ \ ^ \ z;$

```
class Program
{
    static void Main()
    {
        int x = 5, y = 10, z = 15;
        int operazia1 = 0;
        operazia1 = (x += y >> x++ * z);
        Console.WriteLine(operazia1);

        x = 5; y = 10; z = 15;
        int operazia2 = 0;
        operazia2 = (z = ++x & y * 5);
        Console.WriteLine(operazia2);

        x = 5; y = 10; z = 15;
        int operazia3 = 0;
        operazia3 = (y /= x + 5 | z);
        Console.WriteLine(operazia3);

        x = 5; y = 10; z = 15;
        int operazia4 = 0;
        operazia4 = (z = x++ & y * 5);
        Console.WriteLine(operazia4);

        x = 5; y = 10; z = 15;
        int operazia5 = 0;
        operazia5 = (x = y << x++ ^ z);
        Console.WriteLine(operazia5);

        Console.ReadKey();
    }
}
```

15. Используя VisualStudio, создайте проект по шаблону ConsoleApplication. Напишите программу расчета начисления премий сотрудникам. Премии рассчитываются согласно выслуге лет.

Если выслуга до 5 лет, премия составляет 10% от заработной платы.

Если выслуга от 5 лет (включительно) до 10 лет, премия составляет 15% от заработной платы.

Если выслуга от 10 лет (включительно) до 15 лет, премия составляет 25% от заработной платы.

Если выслуга от 15 лет (включительно) до 20 лет, премия составляет 35% от заработной платы.

Если выслуга от 20 лет (включительно) до 25 лет, премия составляет 45% от заработной платы.

Если выслуга от 25 лет (включительно) и более, премия составляет 50% от заработной платы.

Результаты расчета, выведите на экран.



```

class Program
{
    static void Main()
    {
        Console.WriteLine("Введите сумму зарплаты");
        int zarplata = int.Parse(Console.ReadLine());
        Console.WriteLine("Введите срок выслуги");
        int vuslyga = int.Parse(Console.ReadLine());

        if (0 < vuslyga && vuslyga < 5)
        {
            Console.WriteLine("Ваша премия составит: {0}", zarplata * 0.10);
        }
        else if (5 <= vuslyga && vuslyga < 10)
        {
            Console.WriteLine("Ваша премия составит: {0}", zarplata * 0.15);
        }
        else if (10 <= vuslyga && vuslyga < 15)
        {
            Console.WriteLine("Ваша премия составит: {0}", zarplata * 0.25);
        }
        else if (15 <= vuslyga && vuslyga < 20)
        {
            Console.WriteLine("Ваша премия составит: {0}", zarplata * 0.35);
        }
        else if (20 <= vuslyga && vuslyga < 25)
        {
            Console.WriteLine("Ваша премия составит: {0}", zarplata * 0.45);
        }
        else if (25 <= vuslyga)
        {
            Console.WriteLine("Ваша премия составит: {0}", zarplata * 0.50);
        }
        else
        {
            Console.WriteLine("Вы ввели что-то неправильно");
        }

        // Delay.
        Console.ReadKey();
    }
}

```

16. Создайте две целочисленные переменные и задайте им некоторые значения. Применяя технику вложенных циклов, нарисуйте прямоугольник из звездочек. Используйте значения ранее созданных переменных для указания высоты и ширины прямоугольника.

```

class Program
{
    static void Main()
    {
        for (int a = 0; a < 10; a++)
        {
            for (int b = 0; b < 30; b++)
            {
                Console.Write("*");
            }

            Console.WriteLine();
        }

        // Delay.
        Console.ReadKey();
    }
}

```

17. Дано два числа  $A$  и  $B$  ( $A < B$ ) выведите сумму всех чисел расположенных между данными числами на экран.

```

class Program
{
    static void Main()
    {
        Console.WriteLine("Введите число A");
        int a = int.Parse(Console.ReadLine());
        Console.WriteLine("Введите число B");
        int b = int.Parse(Console.ReadLine());
        int s = 0;
        // Оператор инкремента (++) увеличивает свой операнд на 1
        if (a <= b)
        {
            for (a = 3; a <= b; a++)
            {
                b--;
                if (a < b)
                    s += (a + b);
                else s += a;
            }
            Console.WriteLine("Сумма чисел равна {0}", s);
        }

        else
        {
            Console.WriteLine("Вы ввели неверно");
        }

        Console.ReadKey();
    }
}

```

18. Дано два числа  $A$  и  $B$  ( $A < B$ ) выведите все нечетные значения, расположенные между данными числами.

```
class Program
{
    static void Main()
    {
        Console.WriteLine("Введите число A");
        int a = int.Parse(Console.ReadLine());
        Console.WriteLine("Введите число B");
        int b = int.Parse(Console.ReadLine());
        if (a <= b)
        {
            if ((a & 1) == 0)
                a++;
            for (int i = a; i < b; i += 2)
                Console.WriteLine("Число {0} - нечетное", i);
        }

        else
        {
            Console.WriteLine("Вы ввели неверно");
        }

        Console.ReadKey();
    }
}
```

19. Имеется  $N$  клиентов, которым компания производитель должна доставить товар. Сколько существует возможных маршрутов доставки товара, с учетом того, что товар будет доставлять одна машина? Используя VisualStudio, создайте проект по шаблону ConsoleApplication. Напишите программу, которая будет рассчитывать, и выводить на экран количество возможных вариантов доставки товара.

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Введите количество клиентов");
        int n = int.Parse(Console.ReadLine());
        int v = 1;
        do
        {
            v *= n;
            n--;
        } while (n > 1);

        Console.WriteLine("Количество вариантов равно {0}", v);
        Console.ReadKey();
    }
}
```

20. Создайте метод с именем Calculate, который принимает в качестве параметров три целочисленных аргумента и выводит на экран среднее арифметическое значений аргументов.

```
class Program
{
    // Создайте метод с именем Calculate, найдите среднее арифметическое трех целочисленных
    static int Calculate(int a, int b, int c)
    {
        return (a + b + c) / 3;
    }

    static void Main()
    {
        int a = 17, b = 55, c = 49;
        int s = Calculate(a, b, c);

        Console.WriteLine("{0} + {1} + {2})/3 = {3}", a, b, c, s);

        Console.ReadKey();
    }
}
```

21. Напишите программу, которая будет выполнять конвертирование валют.

Пользователь вводит:

- сумму денег в определенной валюте
- курс для конвертации в другую валюту

```
class Program
{
    // Сигнатура метода
    static int Convert(int summa, int kurs)
    {
        return summa / kurs;
    }

    static void Main()
    {
        Console.WriteLine("Введите сумму денег");
        int summa = int.Parse(Console.ReadLine());
        Console.WriteLine("Введите курс валюты");
        int kurs = int.Parse(Console.ReadLine());

        int c = Convert(summa, kurs);
        Console.WriteLine("{0} / {1} = {2}", summa, kurs, c);

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

22. Напишите программу - консольный калькулятор. Создайте две переменные с именами operand1 и operand2. Задайте переменным некоторые произвольные значения. Предложите пользователю ввести знак

арифметической операции. Примите значение введенное пользователем и поместите его в строковую переменную sign.

Для организации выбора алгоритма вычислительного процесса, используйте переключатель switch. Выведите на экран результат выполнения арифметической операции.

В случае использования операции деления, организуйте проверку попытки деления на ноль. И если таковая имеется, то отмените выполнение арифметической операции и уведомите об ошибке пользователя.

```
static void Main()
{
    Console.WriteLine("Введите первое число");
    int operand1 = int.Parse(Console.ReadLine());
    Console.WriteLine("Введите второе число");
    int operand2 = int.Parse(Console.ReadLine());

    // Предлагаем пользователю выбрать арифметический знак.
    Console.WriteLine("Возможные арифметические операции: суммирование(+), вычитание(-), умножение(*),");
    Console.WriteLine("Пожалуйста, введите знак арифметической операции: ");

    // Принимаем ввод от пользователя.
    string sign = Console.ReadLine();

    switch (sign)
    {
        case "+":
            Console.WriteLine("Сумма равна {0}", operand1 + operand2);
            break;
        case "-":
            Console.WriteLine("Сумма равна {0}", operand1 - operand2);
            break;
        case "*":
            Console.WriteLine("Сумма равна {0}", operand1 * operand2);
            break;
        case "/":
            if (operand2 != 0)
                Console.WriteLine("Сумма равна {0}", operand1 / operand2);
            else
                Console.WriteLine("Деление на ноль запрещено");
            break;

        default:
            Console.WriteLine("Неверный выбор. Пожалуйста выберите +, -, * или /.");
            break;
    }

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

23. Создайте массив размерностью в 10 элементов, выведите на экран все элементы массива в обратном порядке.

```

class Program
{
    static void Main(string[] args)
    {
        int n = 15;
        int[] A = new int[n];

        for (int N = 0; N < 15; N++)
        {
            if (n >= 0)
            {
                n--;
                A[n] = n;
                Console.WriteLine(n);
            }
        }
        Console.ReadKey();
    }
}

```

24. Создать метод, который будет выполнять увеличение длины массива переданного в качестве аргумента, на 1 элемент. Элементы массива, должны сохранить свое значение и порядок индексов.

Создайте метод, который принимает два аргумента, первый аргумент - типа `int [] array`, второй аргумент - типа `int value`. В теле метода реализуйте возможность добавления второго аргумента метода в массив по индексу – 0, при этом длина нового массива, должна увеличиться на 1 элемент, а элементы получаемого массива в качестве первого аргумента должны скопироваться в новый массив начиная с индекса - 1.

```

class Program
{
    static void MyArray(int b, int[] mas1)
    {
        int[] mas2 = new int[mas1.Length + 1];
        int c = 1;
        while (c < mas2.Length)
        {
            mas2[0] = b;
            mas2[c] = mas1[c - 1];
            c++;
        }
        int n = 0;
        while (n < mas2.Length)
        {
            Console.WriteLine(mas2[n]);
            n++;
        }
    }
    static void Main(string[] args)
    {
        Console.WriteLine("Введите первую цифру которую хотите добавить в массив");
        int a = Convert.ToInt32(Console.ReadLine());
        int[] mas = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
        MyArray(a, mas);
        Console.ReadKey();
    }
}

```

25. Используя циклы и метод: `Console.Write("*")` выведите на экран прямоугольник.

```

class Program
{
    static void Main()
    {
        for (int a = 0; a < 5; a++)
        {
            for (int b = 0; b < 10; b++)
            {
                Console.Write("*");
            }

            Console.WriteLine();
        }

        Console.ReadKey();
    }
}

```

26. Цикл с параметром `For`. Даны два целых числа  $A$  и  $B$  ( $A < B$ ). Вывести в порядке возрастания все целые числа, расположенные между  $A$  и  $B$  (включая сами числа  $A$  и  $B$ ), а также количество  $N$  этих чисел.

```

int A = 5, B = 13, k = 0;
for (int i = A; i <= B; i++)
{
    Console.WriteLine(i.ToString());
    k++;
}
Console.WriteLine("Количество: " + k.ToString());
Console.ReadLine();

```

27. Цикл с условием WHILE. Даны положительные числа  $A$  и  $B$  ( $A > B$ ). На отрезке длины  $A$  размещено максимально возможное количество отрезков длины  $B$  (без наложений). Не используя операций умножения и деления, найти количество отрезков  $B$ , размещённых на отрезке  $A$ .

```

int A = 22, B = 5, k = 0;
while (A > B)
{
    A -= B;
    k++;
}
Console.WriteLine("Количество: " + k.ToString());
Console.ReadLine();

```

28. Цикл с постусловием Repeat/Do-WHILE. Дано целое число  $N(>1)$ . Если оно является простым, т.е. не имеет положительных делителей, кроме 1 и само на себя, то вывести TRUE, иначе вывести FALSE.

```

        int N = 29, k = 1;
bool f = false;
do
{
    if (N % ++k == 0)
        f = true;
} while (k <= Math.Sqrt(N));
Console.WriteLine(f.ToString().ToUpper());
Console.ReadLine();

```

29. Одномерный массив (Array). Дано целое число  $N(>0)$ . Сформировать и вывести целочисленный массив размера  $N$ , содержащий степени двойки от первой до  $N$ -й :2,4,8,16,...

```

Console.WriteLine("Введите N");
int N = int.Parse(Console.ReadLine());
int[] Array = new int[N];
int j=1;
for (int i = 0; i < N; i++)
{
    j <<= 1;
    Array[i] = j;
}
for (int i = 0; i < N; i++)
    Console.WriteLine(Array[i].ToString());
Console.ReadLine();

```



30. Определить максимум двух введенных чисел.

```
1   using System;
2   namespace MyProgram
3   {
4       class Program
5       {
6           static void Main(string[] args)
7           {
8               int a = int.Parse(Console.ReadLine());
9               int b = int.Parse(Console.ReadLine());
10              if (a > b)
11                  Console.WriteLine(a);
12              else
13                  Console.WriteLine(b);
14              Console.ReadKey();
15          }
16      }
17  }
```

31. Напишите программу сдачи зачета в зависимости от балла.

```
1   using System;
2   namespace MyProgram
3   {
4       class Program
5       {
6           static void Main(string[] args)
7           {
8               int value = int.Parse(Console.ReadLine());
9               if(value==3 || value == 4 || value == 5)
10                  Console.WriteLine("зачет");
11              else
12                  Console.WriteLine("незачет");
13              Console.ReadKey();
14          }
15      }
16  }
```

32. Описать процедуру PowerA3(AA, BB), вычисляющую третью степень числа AA и возвращающую ее в переменной BB (AA - входной, BB - выходной параметр; оба параметра являются вещественными). С помощью этой процедуры найти третьи степени пяти данных чисел.

```

1 using System.IO;
2 using System;
3
4 class Program
5 {
6     static void Main(string[] args)
7     {
8         float A,B;
9         for (int i = 1; i <= 5; i++){
10            Console.Write("A: ");
11            A=float.Parse(Console.ReadLine());
12            PowerA3(A,out B);
13            Console.Write("B: ");
14            Console.WriteLine(B);
15        }
16    }
17
18    static void PowerA3(float A, out float B)
19    {
20        B = (float)Math.Pow(A,3);
21    }
22 }

```

33. Вернуть массив из функции.

```

1 /*функции принимает три параметра,
2 а возвращает массив*/
3 public double[] myFunction(int m, int n, int p){
4     double[] x=new double[2]; // объявляем массив на два элемента
5     x[0]=...; // заполняем элементы
6     x[1]=...; // теперь второй
7     return x;
8 }

```

34. Напишите программу для класса с указанием пространства имён (по имени проекта):

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace UserForm
7  {
8      class User
9      {
10         // Field
11         public string name;
12
13         // Конструктор принимающий один аргумент
14         public User()
15         {
16             name = "unknown";
17         }
18
19         // Конструктор не принимающий аргументов
20         public User(string nm)
21         {
22             name = nm;
23         }
24
25         // Метод изменяющий имя (пример метода)
26         public void SetName(string newName)
27         {
28             name = newName;
29         }
30     }
31 }

```

35. Напишите программу, которая находит и выводит на экран сумму элементов массива:

```

static void Main(string[] args)
{
    int[] numbers = { 4, 7, 1, 23, 43 };
    int s = 0;
    for (int i = 0; i < numbers.Length; i++)
    {
        s += numbers[i];
    }
    Console.WriteLine(s);
    Console.ReadKey();
}

```

36. Из любого цикла можно досрочно выйти, используя оператор `break`. Использовать данный оператор есть смысл после удовлетворения некоторого условия, иначе цикл завершится на первой итерации. Пример программы, которая проверяет, есть ли в массиве число кратное 13-ти. Найдя такое число, нет смысла дальше проверять остальные элементы массива, и здесь мы используем оператор `break`:

```

static void Main(string[] args)
{
int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
bool b = false;
for (int i = 0; i < numbers.Length; i++)
{
if (numbers[i] % 13 == 0)
{
b = true;
break;
}
}
Console.WriteLine(b ? "В массиве есть число кратное 13" : "В массиве нет числа кратного 13");
Console.ReadKey();
}

```

37. Напишите программу, которая находит сумму нечетных элементов массива:

```

static void Main(string[] args)
{
int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
int s = 0;
for (int i = 0; i < numbers.Length; i++)
{
if (numbers[i] % 2 == 0)
continue; //переход к следующей итерации
s += numbers[i];
}
Console.WriteLine(s);
Console.ReadKey();
}

```

38. Напишите программу, в которой находится сумма элементов массива с использованием цикла foreach:

```

static void Main(string[] args)
{
int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
int s = 0;

foreach (int el in numbers)
{
s += el;
}
Console.WriteLine(s);
Console.ReadKey();
}

```

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

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

40. Напишите программу с использованием `switch`, которая выводит на экран название дня недели соответственно вводимому порядковому номеру дня:

```
static void Main(string[] args)
{
int a;
Console.WriteLine("Введите порядковый номер дня недели:");
a = Convert.ToInt32(Console.ReadLine());
switch (a)
{
case 1:
Console.WriteLine("Понедельник");
break;
case 2:
Console.WriteLine("Вторник");
break;
case 3:
Console.WriteLine("Среда");
break;
case 4:
Console.WriteLine("Четверг");
break;
case 5:
Console.WriteLine("Пятница");
break;
case 6:
Console.WriteLine("Суббота");
break;
case 7:
Console.WriteLine("Воскресенье");
break;
default :
Console.WriteLine("Ошибка");
break;
}
Console.ReadKey();
}
```

## Список литературы

1. Абрамян, М. Visual C# на примерах / М. Абрамян. - М.: БХВ-Петербург, 2012. - 496 с.
2. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2015. - 472 с.
3. Васильев, А. C#. Объектно-ориентированное программирование / А. Васильев. - М.: Питер, 2012. - 320 с.
4. Зиборов, В. В. Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2013. - 480 с.
5. Культин, Н. Microsoft Visual C# в задачах и примерах / Н. Культин. - М.: БХВ-Петербург, 2012. - 314 с.
6. Подбельский, В. В. Язык C#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, 2013. - 408 с.
7. Рендольф, Н. Visual Studio 2010 для профессионалов / Н. Рендольф и др. - М.: Диалектика, 2011. - 584 с.
8. Скит, Джон C# для профессионалов. Тонкости программирования / Джон Скит. - М.: Вильямс, 2014. - 608 с.
9. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен. - М.: Вильямс, 2015. - 633 с.
10. Фленов, М. Библия C# (+ CD-ROM) / М. Фленов. - М.: БХВ-Петербург, 2013. - 560 с.
11. Фримен, А. LINQ. Язык интегрированных запросов в C# 2010 для профессионалов / А. Фримен, Д. Раттц-мл.. - М.: Вильямс, 2013. - 656 с.

Гареева Г.А.  
Григорьева Д.Р.

Основы программирования на языке С#

Учебно-методическое пособие

Подписано в печать 27.01.2020.  
Формат 60x84/16. Печать ризографическая.  
Бумага офсетная. Гарнитура «Times New Roman».  
Усл.п.л. 4,5 Уч.-изд. л. 4,5  
Тираж 50 экз. Заказ № 1270

Издательско-полиграфический центр  
Набережночелнинского института  
Казанского (Приволжского) федерального университета

---

423810, г. Набережные Челны, Новый город, пр.Мира, 68/19  
тел./факс (8552) 39-65-99 e-mail: [ic-nchi-kpfu@mail.ru](mailto:ic-nchi-kpfu@mail.ru)