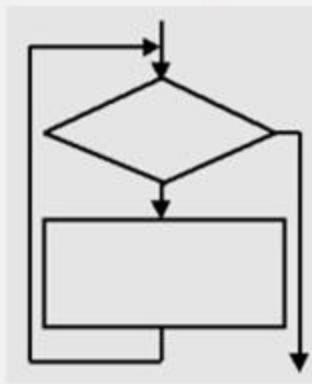
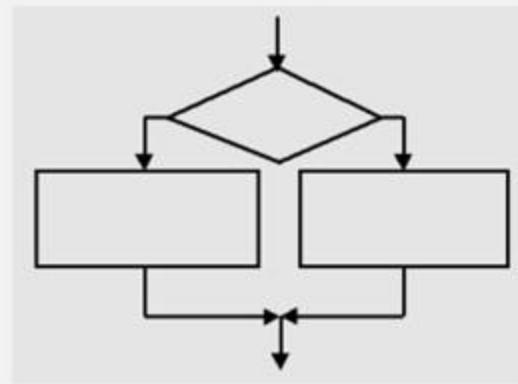
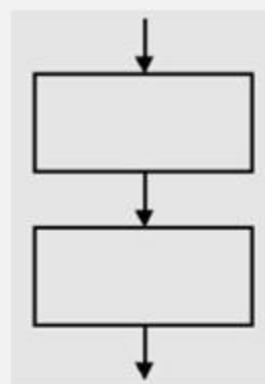


Операторы языка C# вместе с типами данных определяют круг задач, которые можно решать с помощью языка. C# реализует типичный набор операторов языка программирования общего назначения. Операторы являются инструкциями языка, которые представляет собой законченные предложения и определяют законченные этапы выполнения программы и обработки данных. В состав операторов входят ключевые слова, переменные, константы, операции и выражения. Каждый оператор заканчивается символом “;”. В строке программы может быть несколько операторов и один оператор может быть в нескольких строках.

Структурное программирование - технология создания программ, позволяющая путем соблюдения правил сократить время разработки, уменьшить количество ошибок, облегчить возможность модификации программы. В теории структурного программирования программа решения любой задачи может быть реализована в виде одной из 3-х структур (базовых конструкций структурного программирования) или их совместного использования, называемых **следованием**, **ветвлением** и **циклом**.



Следование - конструкция, представляющая собой последовательное выполнение операторов (простых или составных). **Ветвление** - конструкция для выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия. **Цикл** - конструкция для выполнения многократного выполнение оператора.

Есть 4 группы операторов: следования, ветвления, цикла, управления.

Операторы следования выполняются в естественном порядке: начиная с первого до последнего. К операторам следования относятся: 1. **оператор присваивания**, 2. **оператор - выражение**, 3. **составной (пустой) оператор**.

1.1. Простой оператор присваивания: *Переменная = Выражение;*

`x = 3; y = x + 3 * r; s = Math.Sin(x);`

1.2. Множественное присваивание: В операторе последовательно справа налево нескольким переменным присваивается одно и то же значение:

Переменная_1 = Переменная_2 = ... = Переменная_N = Выражение;

`a=b=c=1;` эквивалентно выполнению трёх операторов: `a=1; b=a; c=b;`

1.3. Присваивание с одновременным выполнением операции в общем

виде записывается: *Переменная Знак_операции = Выражение;*

это равносильно: *Переменная = Переменная Знак_операции Выражение;*

`S += 5; //1-й вариант` делает то же, что и `S = S + 5; //2-й вариант`

2. Оператор – выражение Любое выражение, завершающееся точкой с запятой, рассматривается как **оператор**, выполнение которого заключается в вычислении значения выражения или выполнении законченного действия:

`double i = 1, a = 2, b, x, y; //оператор описания`

`b = x = y = 3; //оператор множественного присваивания`

`++i; //оператор инкремента`

`x += y; //оператор сложение с присваиванием`

`Console.WriteLine("x = " + x + " i = " + i); //вызов метода`

`x = Math.Pow(a, b) + a * b; //вычисление сложного выражения`

Частным случаем оператора выражения является пустой оператор:();

3. Оператор составной (блок) - последовательность операторов, заключенных в фигурные скобки { } { *Оператор_1, ..., Оператор_N* }

Блок воспринимается как единичный оператор.

Операторы ветвления предназначены для изменения порядка выполнения операторов в программе: условный оператор **if** (оператор альтернативного выбора) и оператор выбора **switch** (оператор разбора случаев).

Условный оператор if используется для разветвления процесса обработки данных на два направления. Он имеет две формы: сокращенную и полную. Синтаксис объявления полного оператора **if**:

if (Выражение_1) Оператор_1; else Оператор_2;

Синтаксис объявления сокращенного оператора **if**:

if (Выражение_1) Оператор_1;

где **Выражение_1** - логическое выражение;

Оператор_1, Оператор_2 - операторы (простые или составные).

int i1=1, x1=1, y1=1, z1=1, a1=1;

if (a1 > 0) x1 = y1; // Сокращенная форма с простым оператором

if (++i1 > 0) { x1 = y1; y1 = 2 * z1; } // Сокращенная форма с сост. оператор.

int i1=1, j1=1, x1=1, y1=1, z1=1, a1=1, b1=1;

if (a1 > 0 || b1 < 0) x1=y1;

else x1=z1; // Полная форма с простым оператором

if (i1+j1-1 > 0) { x1=0; y1=1; }

else {x=11; y1=0;} // Полная форма с составным оператором

Вложенные if – операторы образуются в том случае, если в качестве элемента **Оператор** (при записи полной формы оператора **if**) используется другой **if**-оператор.

Синтаксис 1 объявления вложенных **if** - операторов:

**if (Выражение_1) Оператор_1;
else if(Выражение_2) Оператор_2;
else if(Выражение_K) Оператор_K;
else Оператор_N;**

else-часть относится к ближайшему if-оператору, которая находится внутри того программного блока, но еще не связана с другой else-частью.

```
Пример использования  
if (i == 10)  
{ if (j < 20) a = b;  
    if (k > 100) c = d;  
    else a = c; //Эта else-часть относится к if(k>100)  
}  
else a = d; // Эта else- часть относится к if(i== 10)
```

Синтаксис 2 объявления вложенных if - операторов:

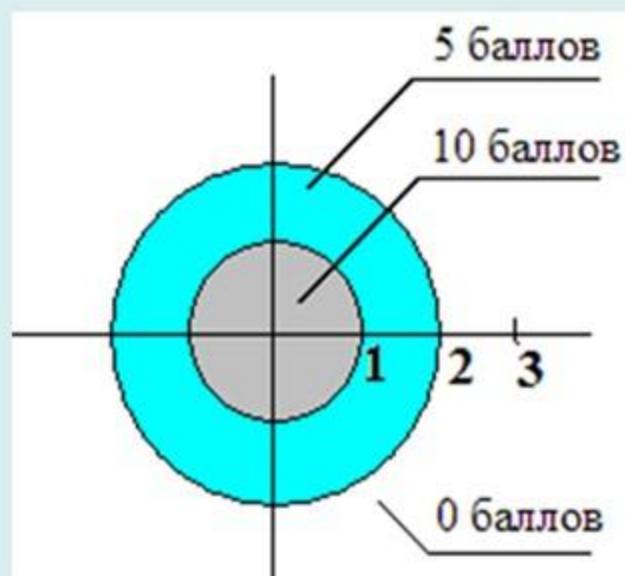
```
if (Выражение_1)
  if(Выражение_2)
    if(Выражение_K)
      if(Выражение_N)
      ...
else Оператор_1
else Оператор_2
else Оператор_K
else Оператор_N
```

```
int v = 3;          Пример использования
if (v > 1)
if (v > 2)
    if (v > 3)
        if (v == 4) Console.WriteLine("v = 4  v = " + v);
        else Console.WriteLine("v < 4  v = " + v);
    else Console.WriteLine("v <= 3  v = " + v);
else Console.WriteLine("v <= 2  v = " + v);
else Console.WriteLine("v <= 1  v = " + v);
```

Результат

$v \leq 3 \quad v = 3$

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;
namespace Оператор_Если
{
    class Program
    {
        static void Main(string[] args)
        {
            int Ball=0;
            float X;
            do
            {
                Console.Write("x=");
                X = float.Parse(Console.F
                Console.Write("y=");
```



```
float Y = float.Parse(Console.ReadLine());
if (X * X + Y * Y <=1)
    Ball = 10; //окружность с радиусом 1
else
    if (X * X + Y * Y <= 4)
        Ball = 5; //окружность с радиусом 2
    Console.WriteLine("Ball= " + Ball);
} while (X != 1111);
}
}
```

Результат выполнения:

**Координата x=0
Координата y=0,5
Результат - 10 баллов**

**Координата x=1,3
Координата y=1,2
Результат - 5 баллов**

Оператор множественного выбора switch (переключатель) используется для выбора одного из нескольких вариантов процесса вычислений.

Синтаксис объявления оператора switch:

```
switch( <Выражение> )
{
    case <Константное_выражение_1>:
        [<Оператор_1>]; <Оператор перехода>;
    case <Константное_выражение_2>:
        [<Оператор_2>]; <Оператор перехода>;
    ...
    case <Константное_выражение_N>:
        [<Оператор_N>]; <Оператор перехода>;
    [default:
        [<Оператор>]; <Оператор перехода>;
```

Часть **default** оператора **switch** может отсутствовать.

После двоеточия (:) может следовать пустой оператор (использование оператора перехода не обязательно).

Константные выражения в **case** должны иметь тот же тип, что и **switch**-выражение.

case - выражения могут быть только константными выражениями.

Особенности использования оператора switch:

1. Оператор перехода (выхода) из части **case** или необязательной части **default** оператора **switch** должен присутствовать, даже если данная часть последняя в операторе **switch**.

```
int I = 8, J = 0;
switch (I)
{
    case 1: J = J + 1;
//Ошибка
}
```

```
int I = 8, J = 0;
switch (I)
{
    case 1: J = J + 1; break;
    default: J = J + 1; //Ошибка
}
```

2. Нельзя использовать части **case** оператора **switch**, без выхода из них при помощи оператора перехода (выхода).

```
int I = 8, A = 6;
switch (I)
{
    case 5: ++A;
//Ошибка - нет выхода из case
    case 6: --A; break;
}
```

```
int I = 8, A = 6;
switch (I)
{
    case 4:
    case 5: //Ошибка нет
    case 6: --A; break;
}
```

3. При выходе из части **case** оператора **switch** можно использовать оператор **goto**.

```
int A = 6; B = 6;
switch (A)
{
    case 5: ++A; goto default; // переход на часть case
    case 6: --A; break; // выход из switch
    default: B = 0; goto case 6; // переход на часть case
}
```

4. Выражение, передаваемое оператору **switch (switch(а))**, может быть только целого (не **float**, не **double**, не **decimal**) или строкового типа

Если нужно проверить попадание значения в диапазон необходимо использовать оператор **if**

```
int Period = 0, Age = 15; string Status;
if ((Age > 0) && (Age < 7)) Period = 1;
else if ((Age >= 7) && (Age < 17)) Period = 2;
else if ((Age >= 17) && (Age < 22)) Period = 3;
```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Операция_переключатель
{
    class Program
    {
        static void Main(string[] args)
        {
            int X = 0, Y = 0;
            bool ok = true;
            string status_oper = " ";
            do
            {
                Console.WriteLine("\n Введите операцию, закончить - Q или q: ");
                char oper = char.Parse(Console.ReadLine());
                switch (oper)
                {
                    case '+':
                    case '-':
                    case '*':
                    case ':': status_oper = "Вычисление"; break;
                    case 'Q':
                    case 'q': status_oper = "Завершение"; break;
                    default: status_oper = "Ошибка";
                    Console.WriteLine(" Ошибка ввода операции..."); break;
                }
                if (status_oper == "Вычисление")
                {
                    bool ok_Val;
                    do
                    {
                        ok_Val = false;
                        Console.WriteLine("\tВведите X: ");
                        int ok_val;
                        if (int.TryParse(Console.ReadLine(), out ok_val) == false)
                        {
                            ok_Val = true;
                            Console.WriteLine(" Ошибка ввода значения X...");
                        }
                    }
                }
            }
        }
    }
}

```

```
        }
        else
        {
            X = ok_val;
            Console.WriteLine("\tВведите Y: ");
            if (int.TryParse(Console.ReadLine(), out ok_val) == false)
            {
                ok_Val = true;
                Console.WriteLine("Ошибка ввода значения Y..");
            }
            else
                Y = ok_val;
        }
    } while (ok_Val);
    float res = 0;
    switch (oper)
    {
        case '+': res = X + Y; break;
        case '-': res = X - Y; break;
        case '*': res = X * Y; break;
        case '/':
            if (X != 0)
            {
                res = (float)X / Y;
                break;
            }; break;
        default: ok = false; break;
    }
    Console.WriteLine("\tРезультат: {0} {1} {2} = {3}", X, oper, Y, res);
}
else
{
    if (status_oper == "Завершение") ok = false;
}
} while (ok);
}
}
```

Результат выполнения:

Ведите операцию, закончить - Q или q: w

Ошибка ввода операции...

Ведите операцию, закончить - Q или q: +

Ведите X: qw

Ошибка ввода значения X...

Ведите X: 12

Ведите Y: sd

Ошибка ввода значения Y...

Ведите X: 12

Ведите Y: 34

Результат: $12 + 34 = 46$

Ведите операцию, закончить - Q или q: *

Ведите X: 345

Ведите Y: 123

Результат: $345 * 123 = 42435$

Операторы цикла Цикл - это последовательность каких-либо действий,

которая может повторяться многократно. Цикл содержит: тело цикла (действия, которые многократно повторяются) и управляющую конструкцию (выражение для определения необходимости повтора цикла).

По расположению управляющей части циклы делятся на циклы с **предусловием** и на циклы с **постусловием**. В **цикле с предусловием** вначале определяется, можно ли выполнять тело цикла, и затем, если можно, тело цикла выполняется. Это происходит до тех пор, пока не станет ложным условие в заголовке цикла. **Цикл с постусловием** всегда выполняется хотя бы один раз. В нём вначале выполняется тело цикла, а затем делается проверка: надо ли снова повторять цикл.

В C# используется четыре оператора цикла: цикл с параметром **for**, цикл с предусловием **while**, цикл с постусловием **do** и цикл перебора **foreach**.

Оператор FOR - цикл с параметром - самый популярный оператор цикла для всех С-подобных языков (объясняется гибкостью и мощью оператора).

Синтаксис объявления оператора for:

for (Начал. действия; Условие; Допол. действия) Оператор; или

for (Начал. действия; Условие; Допол. действия)

{ Операторы тела цикла }

Оператор- тело цикла (обычно телом цикла является блок операторов).

Начальные действия- операторы присваивания, записанные “,”. В секции осуществляется инициализация переменных, необходимых для работы цикла **for**, называемых переменными цикла. Область действия переменных, объявленных в секции - цикл и вложенные блоки. Инициализация выполняется один раз в начале выполнения цикла.

Условие - логическое выражение, задающее условие окончания (выражение при вычислении получает значение **true** или **false**). Истинность выражения проверяется перед каждым выполнением тела цикла (цикл **for** реализован, как цикл с предусловием). В случае отсутствия в секции **Условие** выражения предполагается, что значение выражения **Условие** всегда истинно.

Дополнительные действия - записанные через запятую операторы присваивания и (или) операторы-выражения (инкремент, декремент). В этой секции осуществляется изменение переменной цикла в каждой итерации.

При выполнении оператора for:

1. Вычисляются операторы, составляющие секцию **Начальные действия**.
2. Вычисляется **Условие**. Если оно ложно, то оператор **for** заканчивает свою работу. Если **Условие** истинно, выполняются операторы секции **Операторы тела цикла**, потом - операторы секции **Дополнительных действий**, и затем снова делается проверка истинности **Условия**. И так до тех пор, пока **Условие** остаётся истинным.

Количество операторов в **Начальных** и в **Дополнительных действиях** может быть любым. Любая часть в заголовке (или даже все части) могут отсутствовать: **for (; ;) { Операторы}** - но **(;)** обязательно должны быть.

Различные варианты использования оператора **for**:**1 - стандартный вид цикла**

```
int Sum1, i1;
for (Sum1 = 0, i1 = 0 ; i1 < 5; i1++) Sum1 = Sum1 + i1;
```

2 - отсутствует часть Начальные действия

```
int i2 = 0, Sum2 = 0;
for ( ; i2 < 5; i2++) Sum2 += i2;
```

3 - отсутствует части Начальные действия, Условие

```
int i3 = 0, Sum3 = 0;
for ( ; ; i3++)
{
    if (i3 >= 5) break;
    Sum3 += i3;
}
```

4 - отсутствует части: Начальные действия, Дополнительные действия

```
int i4 = 0, Sum4 = 0;
for ( ; i4 < 5 ; ) Sum4 += i4++;
```

5 - отсутствуют все три части оператора цикла

```
int Sum5 = 0, i5 = 0;
for ( ; ; )
{
    if (i5 >= 5) break;
    Sum5 += i5++;
}
```

6 - отсутствует тело цикла, в этом случае в конце оператора - ";"

```
int i6, Sum6;
for (Sum6 = 0, i6 = 0; i6 < 5; Sum6 += i6++) ;
```

7 - отсутствует тело цикла (суммирование производится по 2 – м пар.)

```
int i7, j7, Sum7;
for (Sum7 = 0, i7 = 0, j7 = 5; i7 < 5 & j7 > 0; Sum7 += i7++, j7--) ;
```

Способы суммирования элементов одномерного массива:

```

int a[5]={1, 22, -3, -4, 5}; // объявление и инициализация массива
1. int Sum=0;
   for ( int i=0; i<5; i++ )
      Sum=Sum+a[i];
2. int i, Sum;
   for ( Sum=0,i=0; i<5; i++ )
      Sum+=a[i];
3. int i, Sum;
   for ( Sum=0, i=0; i<5; Sum+=a[i], i++ ); // отсутствует тело цикла
   for ( Sum= i=0; i<5; Sum+=a[i++]); // отсутствует тело цикла
4. int Sum=0, i=0;
   for ( ; ); // отсутствуют все три части заголовка оператора цикла
   {
      if ( i>=5) break;
      Sum+=a[i];
      i++;
   }

```

Оператор WHILE - цикл с предусловием Цикл выполняется, пока истинно

логическое условие - выражение, имеющее результат логического типа.

Синтаксис объявления оператора while:

while (Условие) Оператор; или **while (Условие) {Операторы тела цикла}**

При выполнении оператора while:

Вычисляются значение **условия** в заголовке оператора. Если условие истинно, то выполняются **операторы тела цикла**, и затем управление снова передаётся оператору цикла. Если условие ложно, то оператор заканчивает работу. После этого будет выполняться оператор, следующий сразу же за оператором цикла. Условие вычисляется перед каждой итерацией цикла.

Если при 1-й проверке условие ложно - **цикл while не выполнится ни разу.**

Использование цикла **while** для вычисления суммы квадратов n чисел.

```
int i, n, s;
Console.WriteLine("n=");
n = int.Parse(Console.ReadLine());
s = 0; i = 1;
while (i <= n)
{ s += i * i; i++; }
Console.WriteLine("s={0}", s);
```

Оператор DO - цикл с постусловием всегда выполняется хотя бы один раз перед первой проверкой логического условия (выражения).

Синтаксис объявления оператора do:

do { Операторы тела цикла } while (Условие)

При выполнении оператора do: Выполняются **операторы тела цикла**. Если **условие** истинно, тело цикла выполняется еще раз и проверка повторяется. Если **условие** (выражение) ложно, или в теле цикла будет выполнен какой-либо оператор передачи управления, оператор цикла заканчивает работу.

Использование цикла do для вычисления суммы квадратов первых n натуральных чисел (пример устранения побочного эффекта).

```
int i, n, s;
do
{ Console.WriteLine("Введите n>0:");
  n = int.Parse(Console.ReadLine()); }
while(n < 1);
s = 0; i = 1;
do
{ s += i * i; i++; }
while(i <= n);
Console.WriteLine("s={0}", s);
```

Оператор FOREACH - цикл перебора - используется для просмотра всех объектов специальным образом организованных данных (*массив, список или другой контейнер*). Удобство цикла заключается в том, что не требуется определять количество элементов и выполнять их перебор по индексу - просто указывается необходимость перебора всех элементов группы.

Синтаксис объявления оператора foreach:

foreach(Тип Переменная_Имя in Коллекция)

{ Операторы тела цикла }

Тип – определяет тип данных, **in** - ключевое слово. **Переменная_Имя** – задает локальную по отношению к циклу переменную цикла, которая будет по очереди принимать все значения из массива **Коллекция** (имя группы данных). В теле цикла выполняются действия с переменной цикла.

При выполнении оператора foreach:

Для каждого элемента коллекции в теле цикла выполняются требуемые действия. Текущий элемент коллекции, с которым выполняются действия при очередном выполнении тела цикла, хранится во временной переменной **Переменная_Имя** (которая имеет тип данных элементов коллекции).

Вывод одномерного массива на экран с помощью оператора foreach:

**foreach (int x in a) //x- локальная переменная цикла, a - одномерный массив
Console.WriteLine(x);**

Рекомендации по выбору оператора цикла

- Оператор **do while** обычно используют, когда цикл требуется обязательно выполнить хотя бы раз, например, если в цикле производится ввод данных.
- Оператором **while** удобнее пользоваться в тех случаях, когда либо число итераций заранее неизвестно, либо очевидных параметров цикла нет, либо модификацию параметров удобнее записывать не в конце тела цикла.
- Оператор **for** предпочтительнее в большинстве остальных случаев - для организации циклов со счетчиками, то есть с целочисленными переменными, которые изменяют свое значение при каждом проходе цикла регулярным образом (например, увеличиваются на 1).

Операторы управления – перехода - позволяют прервать естественный

порядок выполнения операторов. В C# используются:

- оператор безусловного перехода **goto**;
- оператор выхода из цикла **break**;
- оператор перехода к следующей итерации цикла **continue**;
- оператор возврата из функции **return**;
- оператор генерации исключения **throw**.

Операторы могут передать управление в пределах блока, и за его пределы.

Передавать управление внутрь другого блока **запрещается**.

Оператор goto - оператор безусловного перехода. При его выполнении управление программой передается инструкции, указанной с помощью метки.

Синтаксис объявления оператора goto:

goto [метка|case константное_выражение|default];

Метка - идентификатор, за которым следует двоеточие. Метка должна находиться в одном методе с инструкцией **goto**, которая ссылается на эту метку. Все операторы языка C# могут иметь метку.

```
int x = 1;
loop1: if (x < 100) goto loop1;
```

Оператор выхода из цикла BREAK - может стоять в теле цикла или завершать case-ветвь в операторе switch. При выполнении оператора **break** в теле цикла завершается выполнение самого внутреннего цикла. В теле цикла (чаще всего) оператор **break** помещается в одну из ветвей оператора **if**, проверяющего условие преждевременного завершения цикла:

```
int i = 1, j = 1;
for (i = 1; i < 100; i++)
{
    for (j = 1; j < 10; j++)
    {
        if (j >= 3) break;
    }
    Console.WriteLine("Выход из цикла: i = {0} , j = {1}", i, j);
    if (i >= 3) break;
}
Console.WriteLine("Выход из цикла i при i= {0}", i);
```

Оператор перехода к следующей итерации цикла **CONTINUE** можно использовать в теле цикла. Он пропускает все операторы, оставшиеся до конца цикла, и передает управление на начало следующей итерации цикла.

```
for (int i1 = 0; i1 <= 100; i1++) // Вывод четных чисел в диапазоне от 0 - 100
{
    if ((i1 % 2) != 0)      continue; // Переход на следующую итерацию
    Console.WriteLine(i1);
}
```

Оператор возврата из функции **RETURN** - предназначен для завершения выполнения метода. Синтаксис объявления: **return [выражение];**
В общем случае существует два варианта возвращения из метода.

1. обнаружение закрывающей фигурной скобки (обозначает конец метода);
2. выполнение оператора **return**.

Две формы оператора **return**: 1) -предназначена для **void**-методов (методы, которые не возвращают значений) 2) - для возврата из методов значений.

Для немедленного завершения **void** - метода используется конструкция:
return; В методе может быть несколько операторов **return**.

```
public void m1(int i)
{
    if (i > 5)      {      Console.WriteLine(i);      }
    else
    {      Console.WriteLine("Возврат");      return;      }
}
```

Для возвращения из метода значения конструкция: **return значение;**
значение - представляет значение, возвращаемое методом. Для функций использование **return** и аргумента обязательны для возврата значений.

```
public string m2(int i)
{
    if ((i % 2) == 0)  return i + " - четное число";
    else              return i + " - нечетное число";
}
```

Операторы обработки исключений При работе методов возможно возникновение непредвиденных ситуаций (исключений), которые могут привести к аварийному завершению программы (деление на 0, недостаток оперативной памяти, отсутствие требуемых ресурсов - файлов, баз данных). Если в некотором методе предполагается возможность появления таких исключений, то нужно предусмотреть их обработку. Для этого используются **try**-блоки. Перед такими блоками стоит ключевое слово **try**. Вслед за этим блоком следуют один или несколько блоков, обрабатывающих исключения, – **catch**-блоков. Каждый **catch**-блок имеет формальный параметр класса **Exception** (из библиотеки FCL) или одного из его потомков.

try {...}	Класс Tk должен принадлежать классам исключений .
catch (T1 e1)	Блок можно сделать охраняемым, используя try -блок.
{...}	Вслед за try -блоком могут следовать catch -блоки.
...	
catch (Tk eK)	Завершает эту последовательность finally -блок – блок финализации (который может отсутствовать).
{...}	
finally {...}	Конструкция может быть вложенной – в состав try -блока может входить конструкция try-catch-finally .

Исключения могут генерироваться средой CLR, платформой .NET Framework, внешними библиотеками или кодом приложения. Исключения в коде приложения создаются при помощи оператора **throw** [выражение]. Выражение **throw** задает объект класса, наследником класса **Exception**. Обычно это выражение **new**, создающее новый объект. Если оно отсутствует, то повторно возникает текущее исключение.

Исключения, определенные в пространстве имен **System**

Исключение	Тип	Исключение
ArithmetiException	метод	Основной класс исключений, происходящих при выполнении арифметических операций, таких как DivideByZeroException и OverflowException

Исключение	Тип	Исключение
TypeInitializationException	метод	Создается, когда статический конструктор создает исключение, и не существует ни одного совместимого предложения catch для обработки исключения
ArrayTypeMismatchException	метод	Тип сохраняемого значения несовместим с типом массива
DivideByZeroException	метод	Попытка деления на нуль
IndexOutOfRangeException	метод	Индекс массива оказался вне диапазона
invalidCastException	метод	Неверно выполнено динамическое приведение типов
OutOfMemoryException	метод	Обращение к оператору new оказалось неудачным из-за недостаточного объема свободной памяти
overflowException	метод	Имеет место арифметическое переполнение
NullReferenceException	метод	Была сделана попытка использовать нулевую ссылку, т.е. ссылку, которая не указывает ни на какой объект
stackOverflowException	метод	Переполнение стека
Message	свойство	Содержит строку, которая описывает причину ошибки
StackTrace	свойство	Содержит имя класса и метода, вызвавшего исключение
TargetSite	свойство	Содержит имя метода, из которого было вызвано исключение
Source	свойство	Содержит имя программы, вызвавшей исключение
HelpLink	свойство	Строка с любой дополнительной информацией
Метод ToString()	свойство	Метод возвращает строку с описанием исключения

Использование свойств и методов класса System.Exception:

```
try
{
    double d = double.Parse(Console.ReadLine());
}
catch (Exception e)
```

```

    Console.WriteLine("Полное описание: " + e); //вызов ToString
    Console.WriteLine("Сообщение об ошибке: " + e.Message);
    Console.WriteLine("Имя класса и метода: " + e.StackTrace);
    Console.WriteLine("Метод: " + e.TargetSite);
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace _Исключения
{
    class Program
    {
        public static void genException(int what)
        {
            int t;           int[] nums = new int[2];
            Console.WriteLine("Получаем " + what);
            try
            {
                switch(what)
                {
                    case 0:
                        t = 10 / what; // Генерируем ошибку деления на нуль
                        break;
                    case 1:
                        nums[4] = 4 ; // Генерируем ошибку индексирования массива
                        break;
                    case 2:
                        return; // Возврат из try-блока
                }
            }
            catch(DivideByZeroException)
            { // Перехватываем исключение
                Console.WriteLine("На ноль делить нельзя!");
                return; // Возврат из catch-блока
            }
            catch(IndexOutOfRangeException)

```

```

// Перехватываем исключение
    Console.WriteLine("Нет соответствующего элемента");
}
finally
{
    Console.WriteLine("Действия после окончания try-блока");
}
}

class FinallyDemo
{
    static void Main(string[] args)
    {
        for (int i = 0; i < 3; i++)
            Program.genException(i);
        Console.ReadKey(true);
    }
}
}


```

Результат выполнения:**Получаем 0****На ноль делить нельзя!****Действия после окончания try-блока****Получаем 1****Нет соответствующего элемента****Действия после окончания try-блока****Получаем 2****Действия после окончания try-блока**