Курс «C#. Программирование на языке высокого уровня»

Павловская Т.А.

Лекция 4. Простейший ввод-вывод. Управляющие операторы

Основные возможности консольного ввода-вывода (класс Console) и управляющие операторы языка (ветвления, циклы, передача управления).

Ввод-вывод в C#

Вывод на консоль

```
using System;
namespace A
{    class Class1
   {    static void Main()
      {
         int       i = 3;
         double  y = 4.12;
         decimal d = 600m;
         string    s = "Вася";

         Console.Write( " y = {0,5:0.# ' руб. '} \n", y );
      }
   }
}
```

Результат работы программы:
3 y = 4,12
d = 600 s = Вася

Ввод с консоли

```
using System;
namespace A
{    class Class1
   {    static void Main()
        {
            string s = Console.ReadLine();          // ввод строки

            char c = (char)Console.Read();          // ввод символа
            Console.ReadLine();

            string buf;                             // буфер для ввода чисел
            buf = Console.ReadLine();
            int i = Convert.ToInt32( buf );         // преобразование в целое

            buf = Console.ReadLine();
            double x = Convert.ToDouble( buf ); // преобразование в вещ.

            buf = Console.ReadLine();
            double y = double.Parse( buf );         // преобразование в вещ.
        }
   }
}
```

Математические функции: класс Math

Имя	Описание	Результат	Пояснения
`Abs`	Модуль	перегружен	|x| записывается как `Abs(x)`
`Acos`	Арккосинус	`double`	`Acos(double x)`
`Asin`	Арксинус	`double`	`Asin(double x)`
`Atan`	Арктангенс	`double`	`Atan(double x)`
`Atan2`	Арктангенс	`double`	`Atan2(double x, double y)` — угол, тангенс которого есть результат деления `y` на `x`
`BigMul`	Произведение	`long`	`BigMul(int x, int y)`
`Ceiling`	Округление до большего целого	`double`	`Ceiling(double x)`
`Cos`	Косинус	`double`	`Cos(double x)`
`Cosh`	Гиперболический косинус	`double`	`Cosh(double x)`
`DivRem`	Деление и остаток	перегружен	`DivRem(x, y, rem)`
`E`	База натурального логарифма (число *e*)	`double`	2,71828182845905
`Exp`	Экспонента	`double`	e^x записывается как `Exp(x)`

`Floor`	Округление до меньшего целого	`double`	`Floor(double x)`
`IEEERemainder`	Остаток от деления	`double`	`IEEERemainder(double x, double y)`
`Log`	Натуральный логарифм	`double`	$\log_e x$ записывается как `Log(x)`
`Log10`	Десятичный логарифм	`double`	$\log_{10} x$ записывается как `Log10(x)`
`Max`	Максимум из двух чисел	перегружен	`Max(x, y)`
`Min`	Минимум из двух чисел	перегружен	`Min(x, y)`
`PI`	Значение числа π	`double`	3,14159265358979
`Pow`	Возведение в степень	`double`	x^y записывается как `Pow(x, y)`
`Round`	Округление	перегружен	`Round(3.1)` даст в результате 3 `Round (3.8)` даст в результате 4
`Sign`	Знак числа	`int`	аргументы перегружены
`Sin`	Синус	`double`	`Sin(double x)`
`Sinh`	гиперболический синус	`double`	`Sinh(double x)`
`Sqrt`	Квадратный корень	`double`	$\sqrt{x}$ записывается как `Sqrt(x)`
`Tan`	Тангенс	`double`	`Tan(double x)`
`Tanh`	Гиперболический тангенс	`double`	`Tanh(double x)`

Пример: перевод температуры из F в C

$$C = \frac{5}{9}(F - 32)$$

```
using System;
namespace CA1
{    class Class1
   {    static void Main()
     {
         Console.WriteLine( "Введите температуру по Фаренгейту" );
        double fahr = Convert.ToDouble(Console.ReadLine() );

        double cels =5.0 / 9 * (fahr - 32);

        Console.WriteLine( "По Фаренгейту: {0} в градусах Цельсия: {1}",
                        fahr, cels );
     }
   }
}
```

Управляющие операторы языка высокого уровня

Реализуют логику выполнения программы:

- следование
- ветвление
- цикл
- передача управления

Блок (составной оператор)

- *Блок* — это последовательность операторов, заключенная в операторные скобки:
  - begin end
  - { }
- Блок воспринимается компилятором как один оператор и может использоваться **всюду, где синтаксис требует одного оператора, а алгоритм — нескольких**.
- Блок может содержать один оператор или быть пустым.

Оператор «выражение»

- Любое выражение, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения.

```
i++;            // выполняется операция инкремента
a *= b + c;     // выполняется умножение с присваиванием
fun( i, k );    // выполняется вызов функции
```

Пустой оператор

- *пустой оператор* **;** используется, когда по синтаксису оператор требуется, а по смыслу — нет:
- while (true);

Это цикл, состоящий из пустого оператора (бесконечный)

- ;;;

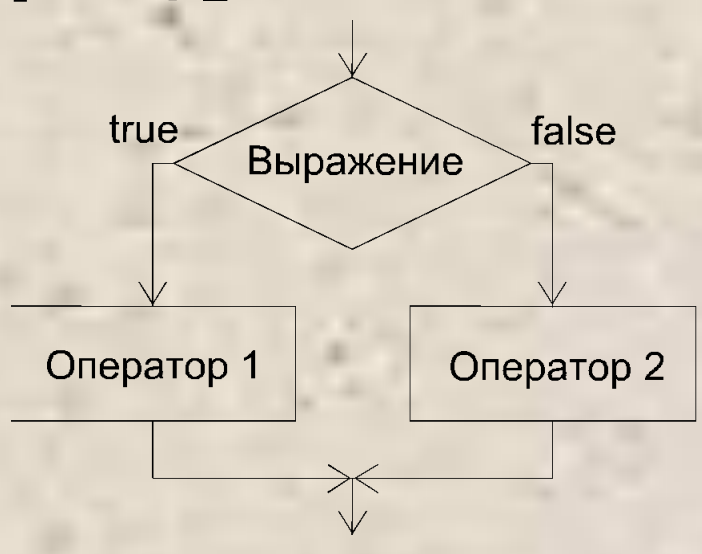
Три пустых оператора

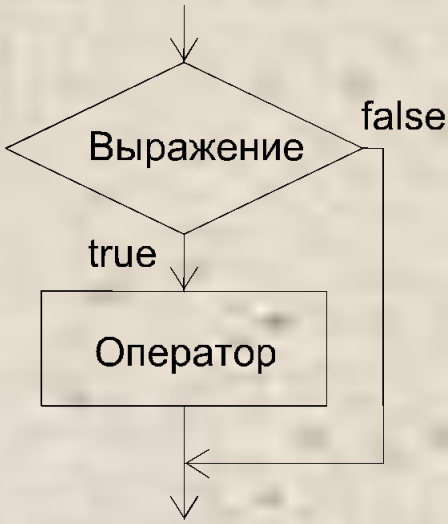
Операторы ветвления:

- развилка (if)
- переключатель (switch)

Условный оператор if

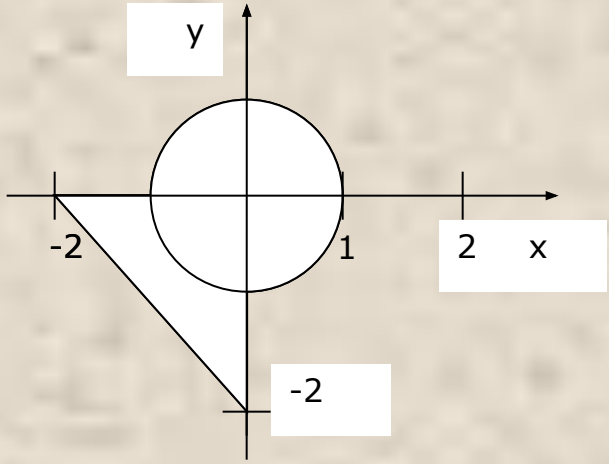
if (выражение) оператор_1;
[else оператор_2;]






```
if ( a < 0 ) b = 1;
if ( a < b && (a > d || a == 0))   ++b;
   else {   b *= a; a = 0; }
   if ( a < b ) if ( a < c ) m = a;
                else         m = c;
   else         if ( b < c ) m = b;
                else         m = c;
```

Пример

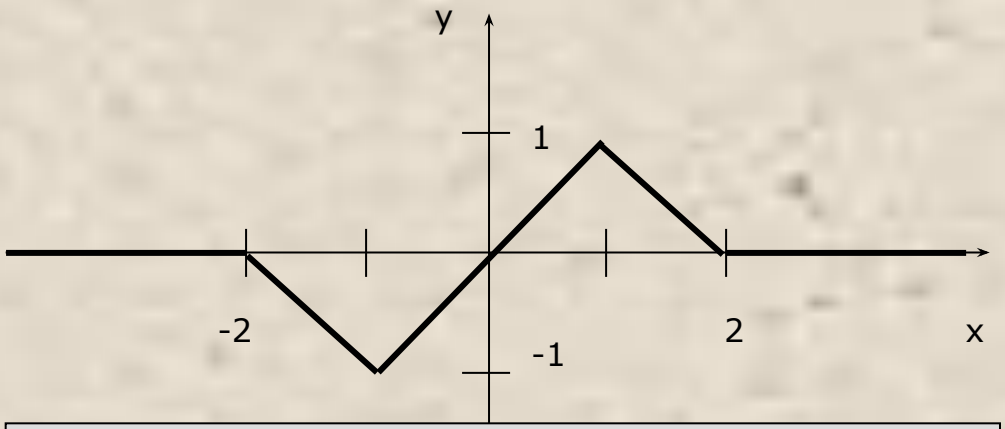



```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {
            Console.WriteLine( "Введите координату x" );
            double x = Convert.ToDouble(Console.ReadLine() );

            Console.WriteLine( "Введите координату y" );
            double y = double.Parse(Console.ReadLine() );

                    if  ( x * x + y * y <= 1 ||
                          x <= 0 && y <= 0 && y >= - x – 2 )
                        Console.WriteLine( " Точка попадает в область " );
                    else
                        Console.WriteLine( " Точка не попадает в область "
);
}}}
```

Пример 2



$$y = \begin{cases} 0, & x < -2 \\ -x - 2, & -2 \le x < -1 \\ x, & -1 \le x < 1 \\ -x + 2, & 1 \le x < 2 \\ 0, & x \ge 2 \end{cases}$$

```
if ( x < -2 )                    y = 0;
if ( x >= -2 && x < -1 ) y = -x - 2;
if ( x >= -1 && x <  1 ) y = x;
if ( x >=  1 && x <  2 ) y = -x + 2;
if ( x >=  2 )                   y = 0;
```

```
if      ( x <= -2 )  y = 0;
else if ( x < -1 )     y = -x - 2;
  else if ( x <  1 )     y = x;
    else if ( x <  2 )     y = -x + 2;
      else                   y = 0;
```

```
y = 0;
if ( x > -2 ) y = -x - 2;
if ( x > -1 ) y = x;
if ( x >  1 ) y = -x + 2;
if ( x >  2 ) y = 0;
```

Проверка вещественных величин на равенство

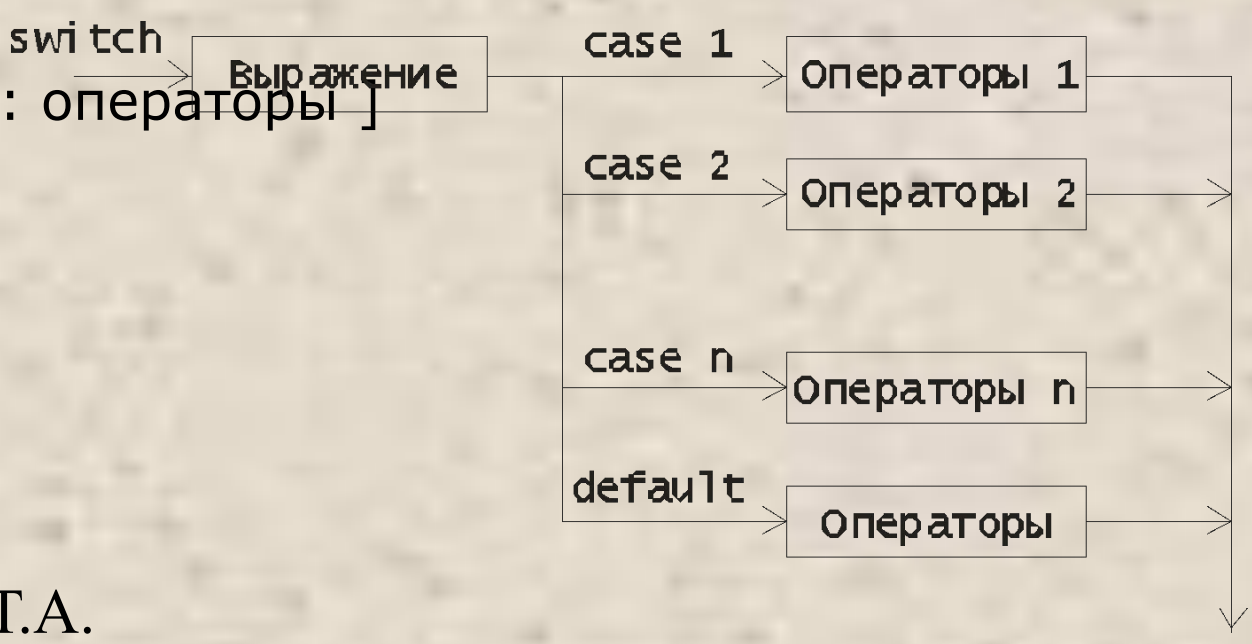
- Из-за погрешности представления вещественных значений в памяти следует ее избегать, вместо этого лучше сравнивать модуль разности с некоторым малым числом.
- float a, b; …
- if (a == b) … // не рекомендуется!
- if (Math.Abs(a - b) < 1e-6) … // надежно!
- Значение величины, с которой сравнивается модуль разности, следует выбирать в зависимости от решаемой задачи и точности участвующих в выражении переменных.
- Снизу эта величина ограничена определенной в классах Single и Double константой Epsilon. Это минимально возможное значение переменной такое, что

  1.0 + Epsilon != 1.0

Оператор выбора switch

```
switch ( выражение ){
    case константное_выражение_1: [ список_операторов_1
   ]
    case константное_выражение_2: [ список_операторов_2
   ]


    case константное_выражение_n: [ список_операторов_n
   ]
    [ default: операторы ]
}
```

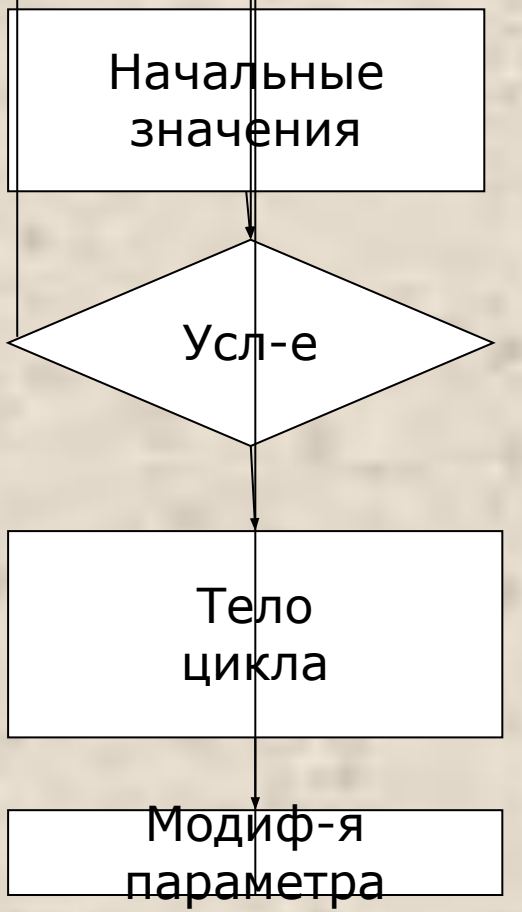


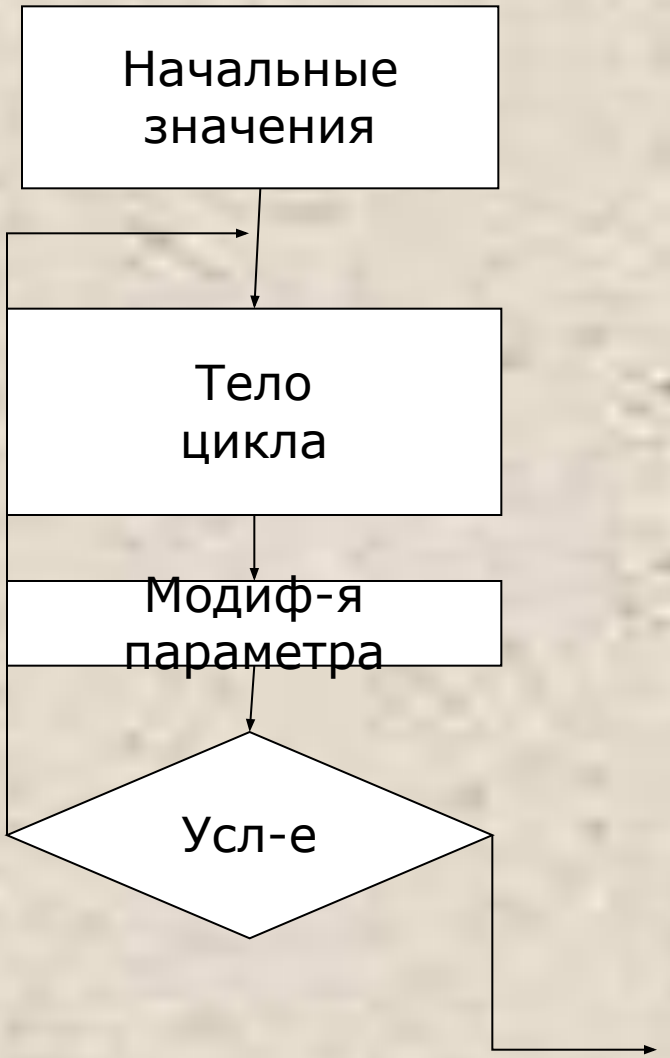
Пример: Калькулятор на четыре действия

```
using System; namespace ConsoleApplication1
{   class Class1    {   static void Main()       {
       string buf;  double a, b, res;
       Console.WriteLine( "Введите 1й операнд:" );
        buf = Console.ReadLine(); a = double.Parse( buf);
        Console.WriteLine( "Введите знак" );
        char op = (char)Console.Read(); Console.ReadLine();
        Console.WriteLine( "Введите 2й операнд:" );
        buf = Console.ReadLine(); b = double.Parse( buf);
       bool ok = true;
       switch (op)
          {
             case '+' : res = a + b; break;
             case '-'  : res = a - b;  break;
             case '*' : res = a * b; break;
             case '/' : res = a / b; break;
             default  : res = double.NaN; ok = false; break;
          }
       if (ok) Console.WriteLine( "Результат: " + res );
       else    Console.WriteLine( "Недопустимая операция" );
    }}}
```

Операторы цикла

Структура оператора цикла

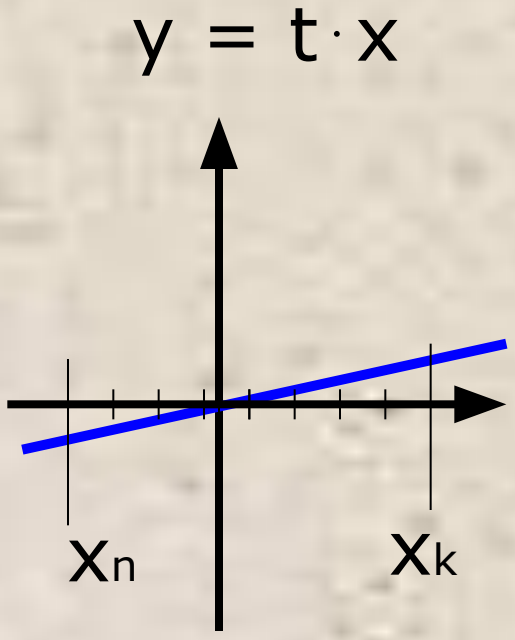




Цикл с предусловием

while (выражение) оператор

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {
            double Xn = -2, Xk = 12, dX = 2, t = 2, y;
            Console.WriteLine( "|     x     |     y     |" );
            double x = Xn;
            while ( x <= Xk )
            {
                y = t * x;
                Console.WriteLine( "| {0,9} | {1,9} |", x, y );
                x += dX;
            }
        }
    }
}
```



Цикл с постусловием

do оператор while выражение;

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {
            char answer;
            do
            {
                Console.WriteLine( "Купи слоника, а?" );
                answer = (char) Console.Read();
                Console.ReadLine();
            } while ( answer != 'y' );
        }
    }
}
```

Цикл с параметром

for (инициализация; выражение; модификации) оператор;

```
int s = 0;
for ( int i = 1; i <= 100; i++ ) s += i;
```

Пример цикла с параметром

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {
            double Xn = -2, Xk = 12, dX = 2, t = 2, y;
            Console.WriteLine( "|     x     |     y     |";
            for ( double x = Xn; x <= Xk; x += dX )
            {
                y = t * x;
                Console.WriteLine( "| {0,9} | {1,9} |", x, y );
            }
        }
    }
}
```

Рекомендации по написанию циклов

- не забывать о том, что если в теле циклов while и for требуется выполнить более одного оператора, нужно заключать их в блок;
- убедиться, что всем переменным, встречающимся в правой части операторов присваивания в теле цикла, до этого присвоены значения, а также возможно ли выполнение других операторов;
- проверить, изменяется ли в теле цикла хотя бы одна переменная, входящая в условие продолжения цикла;
- предусматривать аварийный выход из итеративного цикла по достижению некоторого предельно допустимого количества итераций.

Передача управления

Передача управления

- оператор break — завершает выполнение цикла, внутри которого записан;
- оператор continue — выполняет переход к следующей итерации цикла;
- оператор return — выполняет выход из функции, внутри которой он записан;
- оператор throw — генерирует исключительную ситуацию;
- оператор goto — выполняет безусловную передачу управления

Пример: вычисление суммы ряда

Написать программу вычисления значения функции sin с помощью степенного ряда с точностью по формуле:

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$|C_n| \le \varepsilon$$

$$C_n = (-1)^n \frac{x^{2n-1}}{(2n-1)!}$$

$$C_{n+1} = -C_n \frac{x^2}{2n(2n+1)}$$

Пример: вычисление суммы ряда

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {  double e = 1e-6;            const int iterLimit = 500;
           Console.WriteLine( "Введите аргумент:" );
           double x = Convert.ToDouble(Console.ReadLine());

           bool error = false;         // признак ошибки
           double c = x, y = c;        // член ряда и сумма ряда
           for ( int n = 1; Math.Abs(c) > e; n++ )
           {    c *= - x * x / ((2 * n ) * ( 2 * n + 1 ));
                y += c;
                if ( n > iterLimit ) { error = true; break; }
           }
           if ( error ) Console.WriteLine( "Ряд расходится" );
           else         Console.WriteLine( "Сумма ряда - " + y );
        }}} end.
```

Оператор return

завершает выполнение функции и передает управление в точку ее вызова:

return [выражение];

Оператор goto

goto метка;

В теле той же функции должна присутствовать ровно одна конструкция вида:

метка: оператор;

goto case константное_выражение;

goto default;

Обработка исключений

Исключительная ситуация, или исключение — это возникновение непредвиденного или аварийного события, которое может порождаться некорректным использованием аппаратуры.

Например, это деление на ноль или обращение по несуществующему адресу памяти.

Исключения позволяют логически разделить вычислительный процесс на две части — обнаружение аварийной ситуации и ее обработка.

Возможные действия при ошибке

- прервать выполнение программы;
- возвратить значение, означающее «ошибка»;
- вывести сообщение об ошибке и вернуть вызывающей программе некоторое приемлемое значение, которое позволит ей продолжать работу;
- выбросить исключение

Исключения генерирует либо система выполнения, либо программист с помощью оператора throw.

Некоторые стандартные исключения

Имя	Пояснение
`ArithmeticException`	Ошибка в арифметических операциях или преобразованиях (является предком `DivideBeZeroException` и `OverFlowException`)
`DivideByZeroException`	Попытка деления на ноль
`FormatException`	Попытка передать в метод аргумент неверного формата
`IndexOutOfRangeException`	Индекс массива выходит за границы диапазона
`InvalidCastException`	Ошибка преобразования типа
`OutOfMemoryException`	Недостаточно памяти для создания нового объекта
`OverFlowException`	Переполнение при выполнении арифметических операций
`StackOverFlowException`	Переполнение стека

Оператор try

Служит для обнаружения и обработки исключений.

Оператор содержит три части:

- *контролируемый блок* — составной оператор, предваряемый ключевым словом try. В контролируемый блок включаются потенциально опасные операторы программы. Все функции, прямо или косвенно вызываемые из блока, также считаются ему принадлежащими;
- один или несколько *обработчиков исключений* — блоков catch, в которых описывается, как обрабатываются ошибки различных типов;
- *блок завершения* finally, выполняемый независимо от того, возникла ли ошибка в контролируемом блоке.

Синтаксис оператора try:

try блок [catch-блоки] [finally-блок]

Механизм обработки исключений

- Обработка исключения начинается с появления ошибки. Функция или операция, в которой возникла ошибка, генерируют исключение;
- Выполнение текущего блока прекращается, отыскивается соответствующий обработчик исключения, и ему передается управление.
- В любом случае (была ошибка или нет) выполняется блок finally, если он присутствует.
- Если обработчик не найден, вызывается стандартный обработчик исключения.

Пример 1:

```
try {

        // Контролируемый блок
}
catch ( OverflowException e ) {

        // Обработка переполнения
}
catch ( DivideByZeroException ) {

        // Обработка деления на 0
}
catch {
    // Обработка всех остальных исключений
}
```

Пример 2: проверка ввода

```
static void Main()
    {
        try
        {
            Console.WriteLine( "Введите напряжение:" );
            double u = double.Parse( Console.ReadLine() );
            Console.WriteLine( "Введите сопротивление:" );
            double r = double.Parse(Console.ReadLine() );
            double i = u / r;
            Console.WriteLine( "Сила тока - " + i );
        }
        catch ( FormatException )
        {
            Console.WriteLine( "Неверный формат ввода!" );
        }
        catch                                   // общий случай
        {
            Console.WriteLine( "Неопознанное исключение" );
        }
    }
```

Оператор throw

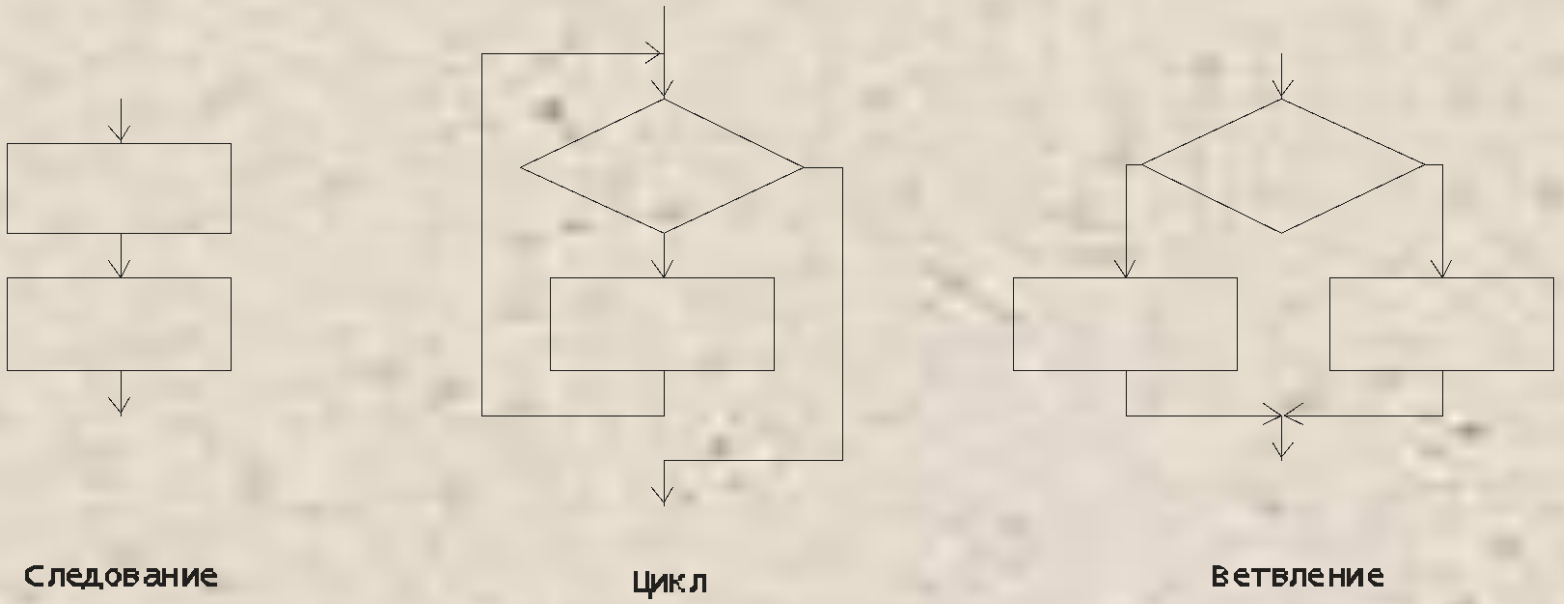
- **throw [выражение];**

Параметр должен быть объектом, порожденным от стандартного класса System.Exception. Этот объект используется для передачи информации об исключении его обработчику.

Пример:

```
throw new DivideByZeroException();
```

Базовые конструкции структурного программирования



- Целью использования базовых конструкций является получение программы простой структуры. Такую программу легко читать, отлаживать и при необходимости вносить в нее изменения.
- Особенностью базовых конструкций является то, что любая из них имеет только один вход и один выход, поэтому конструкции могут вкладываться друг в друга