Основные операторы. Решение задач с использованием основных операторов.

Тема 3

Технологии программирования

- **Структурное программирование** — это технология создания программ, позволяющая путем соблюдения определенных правил сократить время разработки и уменьшить количество ошибок, а также облегчить возможность модификации программы.
- Идеи структурного программирования получили свое дальнейшее развитие в **объектно-ориентированном программировании** — технологии, позволяющей достичь простоты структуры и управляемости очень больших программных систем.

Выражение

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

```
i++;
bool ok=a>b;
a+=5;
```

- Частным случаем выражения является пустой оператор:

```
;
```

Блок (составной оператор)

Последовательность описаний и операторов, заключенная в фигурные скобки.

```
   int s=0;
do{
   int a=Convert.ToInt32(Console.ReadLine());
   s+=a;
} while(a!=0);
```

Операторы ветвления

- Условный оператор

```
//полная форма
if(условие) оператор_1;
else оператор_2;
```

Операторы ветвления

- Условный оператор

//сокращенная форма

if(условие**)** оператор_1;

Примеры

```
//сокращенная форма
if(a<b&&a<c) min=a;

//полная форма
double x1, x2;
double d = Math.Pow(b, 2) - 4 * a * c;
 if (d < 0)
Console.WriteLine("Решения нет");
 else
 {
      x1 = (-b + Math.Sqrt(d)) / (2 * a);
      x2 = (-b - Math.Sqrt(d)) / (2 * a);
     Console.WriteLine("x1={0}, x2={1}", x1.ToString("f5"),
    x2.ToString("f5"));
  }
```

Оператор выбора

```
switch (выражение)
{
    case константа1 : оператор1 ; break;
    case константа2 : оператор2 ; break;
    . . . . . .
    [default: операторы;]
}
```

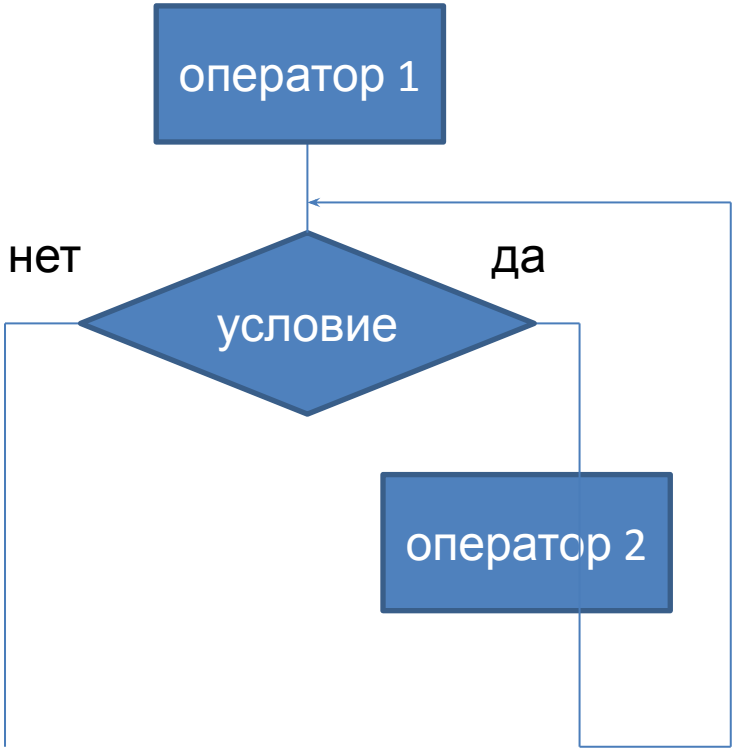
Пример

```
int i;
 Console.WriteLine("Введите целое число");
 i=Int32.Parse(Console.ReadLine());
switch (i)
{
     case 1: Console.WriteLine("\nThe number is one"); break;
     case 2: Console.WriteLine("\n2*2={0}", i * i); break;
     case 3: Console.WriteLine("\n3*3="+ i * i); break;
     case 4: Console.WriteLine("\n" + i + " is very beautiful!");
               break;
    default: Console.WriteLine("\nThe end of work"); break;
          }
```

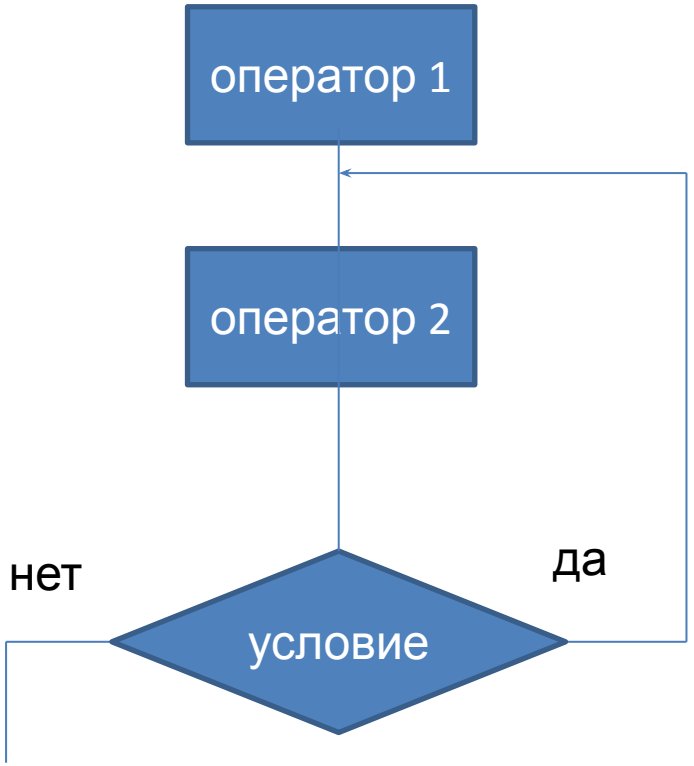
Циклы

- итерационные (известно условие выполнения цикла);
- арифметические (известно количество выполнений цикла).

Итерационные циклы



цикл с предусловием



цикл с постусловием

Цикл с предусловием

оператор_1; //инициализация
while (условие)
оператор_2; //коррекция

Пример:

```
int a=1,s=0; //инициализация
while (a!=0)
{
    a=Int32.Parse(Console.ReadLine()); //коррекция a
    s+=a;
}
```

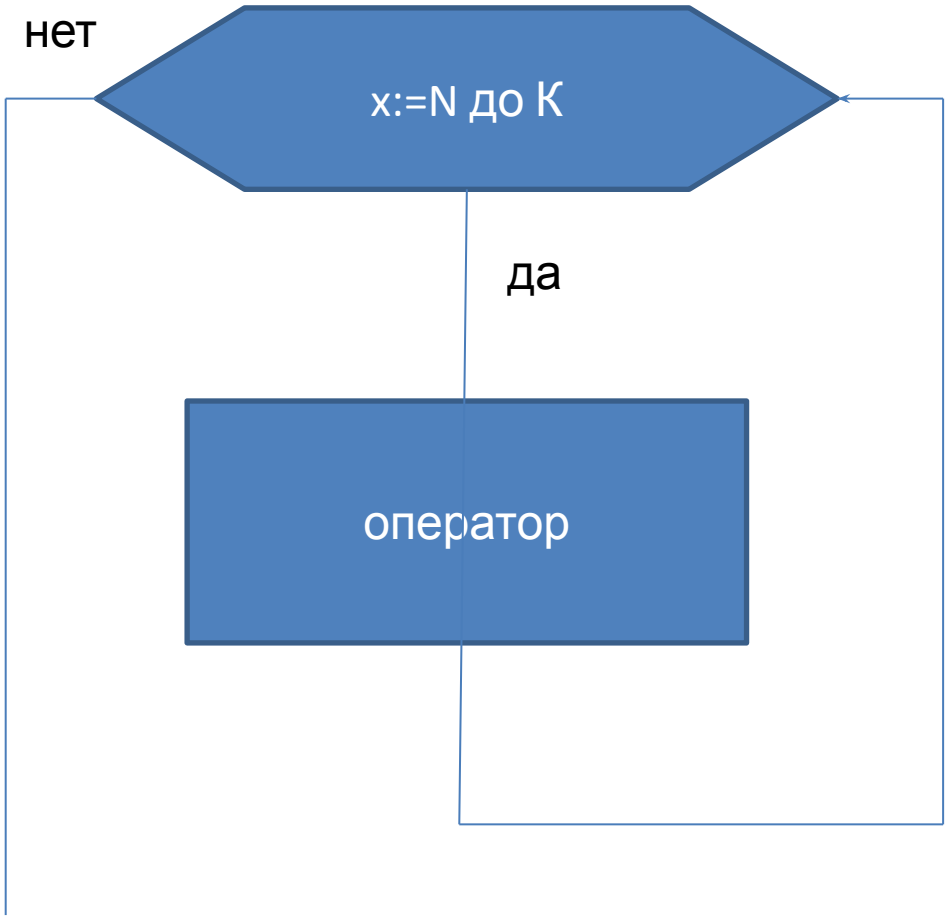
Цикл с постусловием

оператор_1; //инициализация
do
оператор_2;//коррекция
while (условие);

Пример:

```
int a, s=0; //инициализация
do
{
   a=Int32.Parse(Console.ReadLine()); //коррекция a
   s+=a;
} while (a!=0);
```

Арифметический цикл



for(выражение_1; выражение_2; выражение_3)
оператор;

выражение_1 – инициализация
выражение_2 – условие
выражение_3 – коррекция

```
//1 – увеличение параметра
    int a, s=0,i;
    for(i=0; i<10; i++)
    {
       a=Int32.Parse(Console.ReadLine());
       s+=a;
    }
//2 -  уменьшение параметра
        s = 0;
        for(i=n; i>0; i--)
        {
           a=Int32.Parse(Console.ReadLine());
           s+=a;
        }
```

```
//3 изменение шага корректировки
        s=0;
        Console.Write(0);
        for (i = 2; i < 60; i += 13)
        {
           s += i;
           Console.Write("+"+ i );
        }
        Console.WriteLine("=" + s);
//4 проверка условия отличного от того, которое налагается на число
//итераций
        s=0;
        Console.Write(0);
        for ( i=1;i*i<100; i++)
        {
           s += i;
           Console.Write("+" + i);
        }
        Console.WriteLine("=" + s);
```

```
//5  коррекция с помощью умножения
        sd=0;
        Console.Write(0);
        for ( id=10.0; id<15.0; id*=1.1)
        {
           sd += id;
           Console.Write("+" + id);
        }
       Console.WriteLine("=" + sd);
//6 коррекция с помощью арифметического выражения
int x,y=0;
        for (x=1;y<=75;y=5*(x++)+10)
        {
           Console.WriteLine("x={0}, y={1}",x,y);
         }

// 7  использование нескольких корректирующих
//выражений, тело цикла отсутствует
for (x=1, y=0; x<10;x++, y+=x);
```

Операторы перехода

break – оператор выхода из цикла или переключателя .

Пример

```
int summa=0;
        for (int i = 0; i < 10; i++)
        {
            Console.Write(">");
            int number = Int32.Parse(Console.ReadLine());
            if (number == 0) break;
            summa += number;
        }
        Console.WriteLine("Summa=" + summa);
```

continue – переход к следующей итерации цикла.

Пример

```
int summa_pol = 0, kolich_pol = 0;
for (tek = 0, number = 1; number != 0; tek++)
        {
           Console.Write(">");
           number = Int32.Parse(Console.ReadLine());
           if (number <= 0) continue;
           summa_pol += number; kolich_pol++;
        }
Console.WriteLine("summa_pola=" + summa_pol);
Console.WriteLine("kolich_pol=" + kolich_pol);
```

goto – безусловный переход, используется в трех формах:

goto метка:

goto case константа:

goto default:

метка – идентификатор, областью видимости является функция.

Оператор goto используется :

1. необходим принудительный выход из нескольких вложенных циклов;
2. необходим переход из нескольких точек функции в одну точку.

return – оператор возврата из функции, завершает выполнение функции и передает управление в точку вызова.

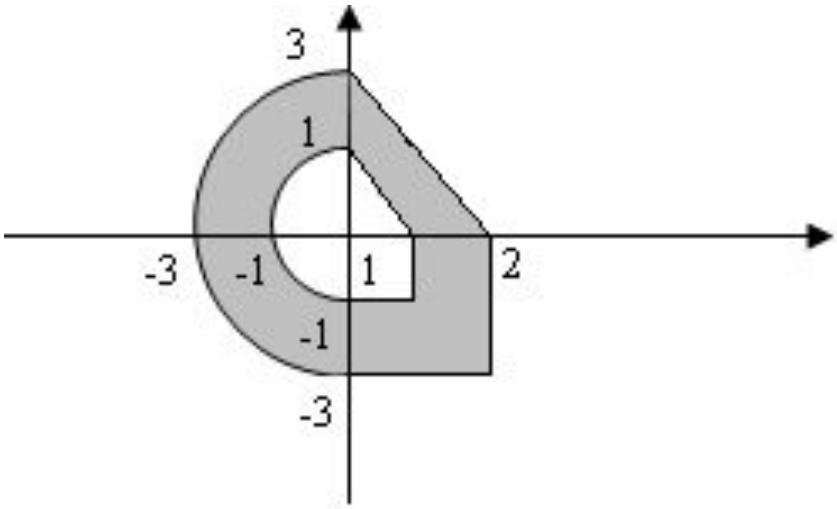
return [выражение];

Пример

```
int  func1()
{
. . . .
return 1;
}
```

Примеры решения задач

- **Задача №1**. Определить, попадет ли точка с координатами (x, y) в заштрихованную область.



Задача 2

- Дана последовательность целых чисел из n элементов. Найти среднее арифметическое этой последовательности.

Задача 3

- Дана последовательность целых чисел, заканчивающаяся нулем. Найти среднее арифметическое этой последовательности.

Задача 4

- Сформировать последовательность чисел Фибоначчи из n элементов.
- Числа Фибоначчи: 1, 1, 2, 3, 5, 8, 13, . . .

[f(i)=f(i-1)+f(i-2)]

Задача 5

- Определить является ли число простым.
- Сформировать n первых простых чисел.

Исключительные ситуации

- **Исключение** – возникновение аварийного события, которое может порождаться некорректным использованием аппаратуры или неправильной работой программы:
  - деление на 0,
  - переполнение.
- Исключения позволяют разделить вычислительный процесс на две части:
  - обнаружение аварийной ситуации;
  - обработка аварийной ситуации.

Исключения могут генерировать:

- среда;
- программист.

Имя стандартного исключения	Описание
ArithmeticException	Ошибка в арифметических операциях
ArrayTypeMismatchException	Попытка сохранения в массиве элемента несовместимого типа
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 при создании объекта
OverFlowException	Переполнение при выполнении арифметических операций
StackOverFlowException	Переполнение стека

Оператор try

- контролируемый блок (try) – содержит потенциально опасные операторы программы.
- обработчик исключения (catch) – содержит операции для обработки ошибки.
- блок завершения (finally) выполняется независимо от того, возникла ошибка в контролируемом блоке или нет.

Синтаксис:

try-блок [блоки catch] [блок finally]

Семантика:

1. Обработка исключения начинается с появления ошибки. Функция или операция, в которой возникла ошибка, генерирует исключение.
2. Выполнение текущего блока прекращается, отыскивается соответствующий обработчик исключения, и ему передается управление.
3. Вне зависимости от возникновения ошибки выполняется блок завершения.
4. Если обработчик не найден, вызывается стандартный обработчик исключения.

- Обработчики исключений должны располагаться непосредственно за блоком try.
- Они начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения.
  - catch(тип имя) { …} - имя параметра используется в теле обработчика для выполнения каких-либо действий, например вывода информации об исключении.
  - catch(тип) {…} - не предполагает использования информации об исключении, играет роль только его тип.
  - catch { …} - применяется для перехвата всех исключений.

```
try
{
Console.Write("Введите количество");
 kolichestvo = Convert.ToInt32(Console.ReadLine());
. . . .
}
catch (FormatException)
 {
 Console.WriteLine("Неправильный формат
    ввода");
}
```