Введение в язык программирования C#.

Плюсы и минусы языка программирования C#.

Простейшая программа на языке программирования C#.

C (язык С в 1972 г. Дэннисом Ритчи) **WinAPI 32** (Windows Application Programming Interface)	ручное управление памятью, сложные синтаксические конструкции; тысячи глобальных функций и типов данных ограниченность ООП
C++/MFC (язык С++ в 1979 г, Бьярни Страуструпом, стандартизирован в 1997)	ручное управление памятью, сложные синтаксические конструкции при построении UI (макросы, мастера)
Visual Basic	отсутствие наследования, нет поддержки создания параметризованных классов, средств создания многопоточных приложений
Java (Официально вышел в 1995 г. Разработчики Джеймс Гослинг, Патрик Ноутон из компании Sun Microsystems)	отсутствие межъязыкового взаимодействия

```
#include <windows.h>
LRESULT CALLBACK WndProc(HWND hwnd,
                UINT msg,
                WPARAM wparam,
                LPARAM lparam);
INT WINAPI WinMain(HINSTANCE hInstance,
                           HINSTANCE hPrevInstance,
                          LPSTR cmdline, int cmdshow)
 {
MSG msg;
HWND hwnd;
WNDCLASSEX wndclass = { 0 };
wndclass.cbSize = sizeof(WNDCLASSEX);
wndclass.style = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc = WndProc;
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground =
(HBRUSH)GetStockObject(WHITE_BRUSH);
wndclass.lpszClassName = TEXT(«Window1»);
wndclass.hInstance = hInstance;
wndclass.hIconSm = LoadIcon(NULL,
                                      IDI_APPLICATION);
RegisterClassEx(&wndclass);
hwnd = CreateWindow(TEXT(«Window1»),
                           TEXT(«Hello World»),
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT,
                           0,
                           CW_USEDEFAULT,
                            0,
                           NULL,
                           NULL,
                           hInstance,
                           NULL);
if( !hwnd )
return 0;
ShowWindow(hwnd, SW_SHOWNORMAL);
UpdateWindow(hwnd);
while( GetMessage(&msg, NULL, 0, 0) )
  {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
  }
 return msg.wParam;
}
LRESULT CALLBACK
WndProc(HWND hwnd, UINT msg,
WPARAM wparam, LPARAM lparam) {
switch(msg) {
case WM_DESTROY:
PostQuitMessage(WM_QUIT);
break;
default:
return
DefWindowProc(hwnd, msg, wparam, lparam);
}
return 0;}
```

Программа «Hello, World» на языке Си с использованием WinAPI

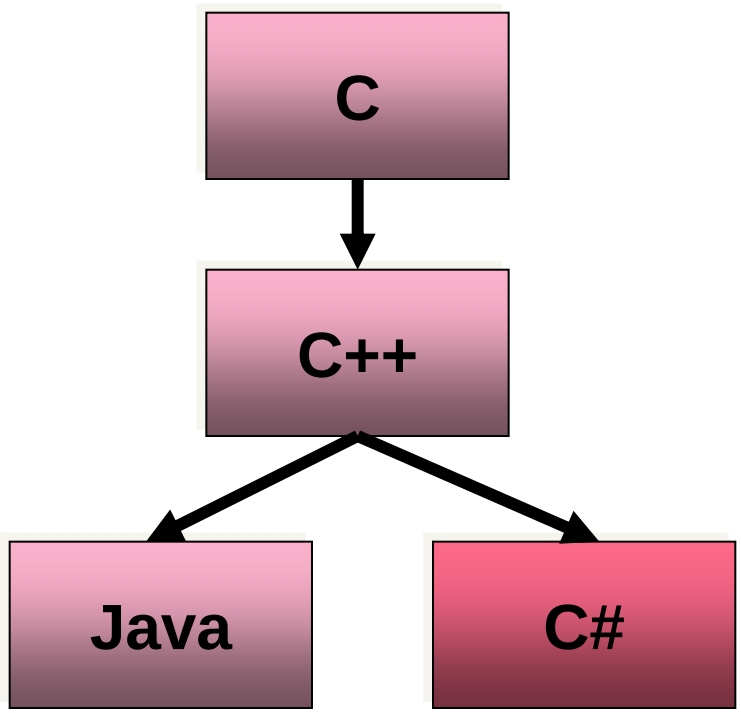
```
import java.awt.event.*;
import java.awt.*;
class simpleFrame extends Frame
{
public static void main(String[] args)
    {
   simpleFrame a= new
              simpleFrame(" Hello, World ");
    }
 simpleFrame(String title)
  {
    setTitle(title);
   show();
 } }
```

Программа «Hello, World» на языке Java

```
using System.Windows.Forms;
using System;
class Program
{
   [STAThread]
static void Main()
   {
        Form myForm = new Form();
        myForm.Text = «Hello World»;
        Application.Run(myForm );
   }
}
```

Программа «Hello, World» на языке C# платформа .NET

Впервые язык C# увидел свет в середине 2000 года. Главным архитектором C# был **Андерс Хейлсберг (Anders Hejlsberg).**



Генеалогическое дерево C#

Достоинства языка C#

- ✓ Не требуется применение указателей.
- ✓ Автоматическое управление памятью через сборку мусора.
- ✓ Наличие перегрузки операций для пользовательских типов без лишних сложностей (по сравнению с C++)
- ✓ Полная поддержка техники программирования, основанной на использовании интерфейсов
- ✓ и ряд других ……

C# 1.0
Управляемый код

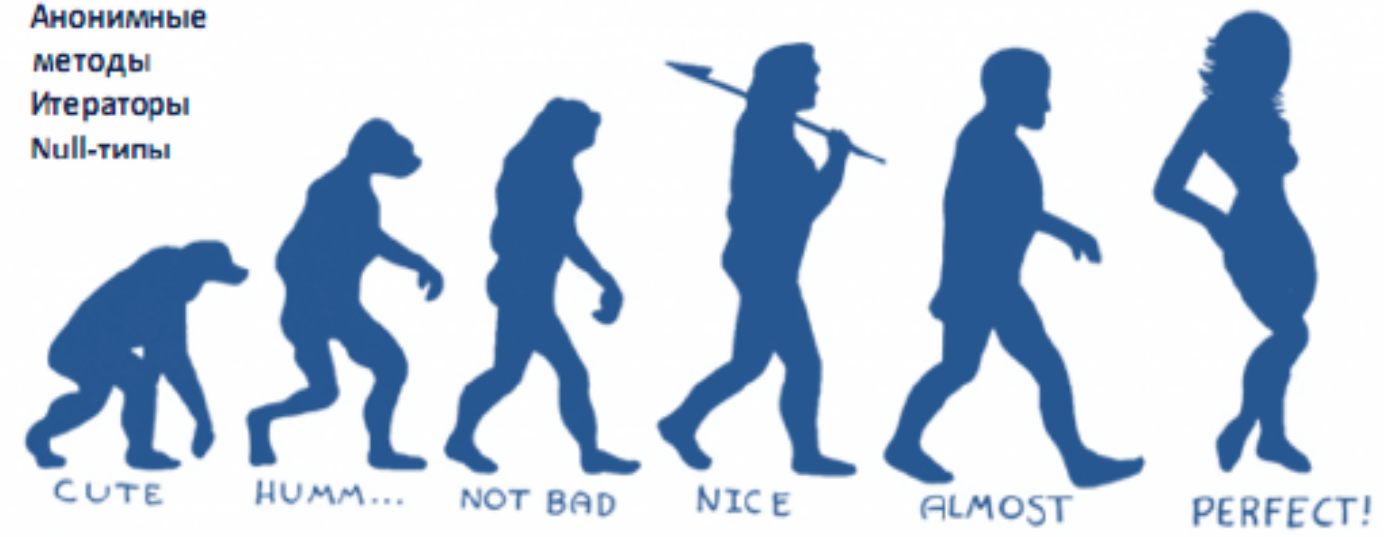
C# 2.0
- Обобщения
- Смешанные типы
- Анонимные методы
- Итераторы
- Null-типы

C# 3.0
- Неявно типизируемые локальные переменные
- Инициализаторы объектов и коллекций
- Автоматическая реализация свойств
- Анонимные типы
- Методы расширения
- Запросы
- Лямбда-выражения
- Деревья выражений

C# 4.0
- Динамическое связывание
- Именованные и
- дополнительные аргументы
- Обобщенная ковариантность и
- контрвариантность

C# 5.0
Асинхронные функции

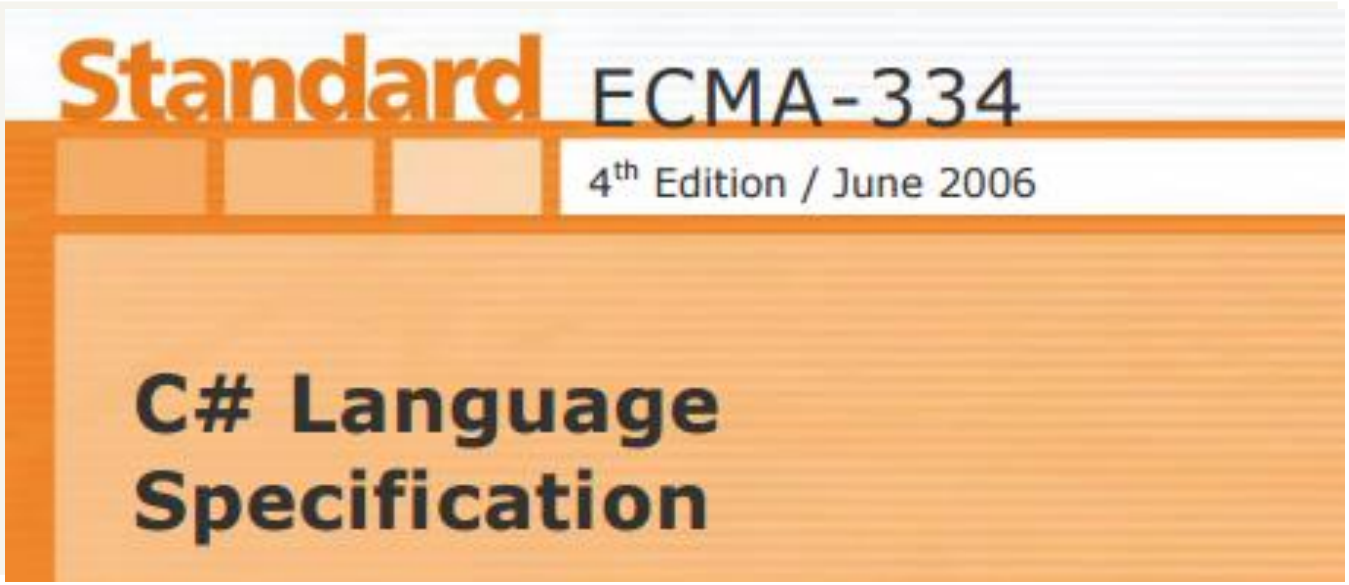
C# 6.0
- Getter-only auto-properties
- Auto-property initializers
- Expression-bodied members
- Null-conditional operators
- Using static members
- Index initializers
- String interpolation
- nameof operator
- Await in catch/finally
- Exception filters
- Extension Add in collection initializers
- Improved overload resolution



C# 7.0
?

Версия C#	Версия CLR	Версия Framework
1.0	1.0	1.0
1.2	1.1	1.1
2.0	2.0	2.0, 3.0
3.0	2.0 (SP1)	3.5
4.0	4.0	4.0
5.0	4.5 (Patched 4.0)	4.5 (including 4.5.1 и 4.5.2)
6.0	4.5 (Patched 4.0)	4.6

http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

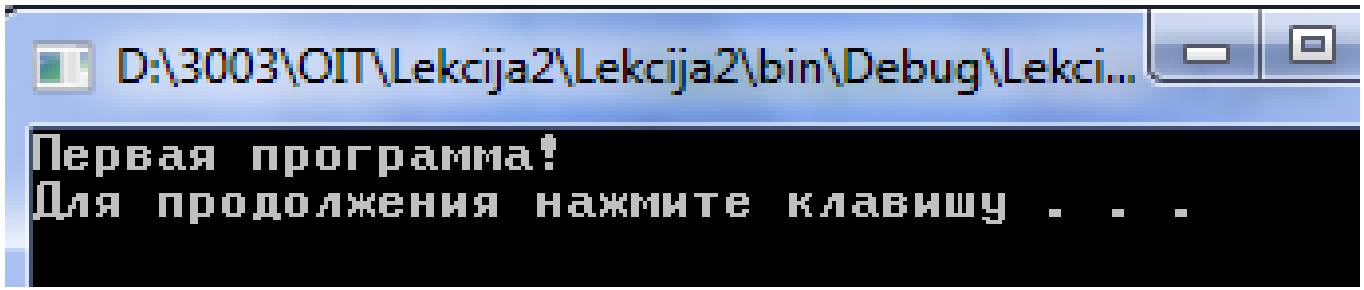


Общая структура программы на языке программирования C# имеет следующий вид:

```
//  1. ПОДКЛЮЧЕНИЕ ДЛЯ ИСПОЛЬЗОВАНИЯ
//     НЕОБХОДИМЫХ ПРОСТРАНСТВ ИМЕН
using System;
//  2. ОПРЕДЕЛЕНИЕ ПРОСТРАНСТВА ИМЕН ТЕКУЩЕГО ПРОЕКТА
namespace Lekcija2
{
//  3. ОБЪЯВЛЕНИЕ КЛАССА Program
    class Program
    {
//  4. ОСНОВНАЯ ФУНКЦИЯ (МЕТОД) ПРОГРАММЫ
        public static void Main()
        {

//  5. МЕСТО,  В КОТОРОМ РЕАЛИЗУЕТСЯ  ОСНОВНАЯ
        //  ЛОГИКА ПРОГРАММЫ
        }
    }
}
```

Пример программы, в которой выводятся сообщения на экран.

```
using System;           // подключение для использования пространства имен
namespace Lekcija2  // определение пространства имен текущего проекта
{
   class Program   // объявление класса Program
   {
      // Основная функция (метод) программы
      public static void Main()
       {
          // Вывод сообщения на экран
          Console.WriteLine("Первая программа!");

          Console.Write("Для продолжения нажмите клавишу . . . ");
          Console.Read();
      }   } }
```



Типы данных.

Целочисленные типы данных.

Типы данных для чисел с плавающей точкой.

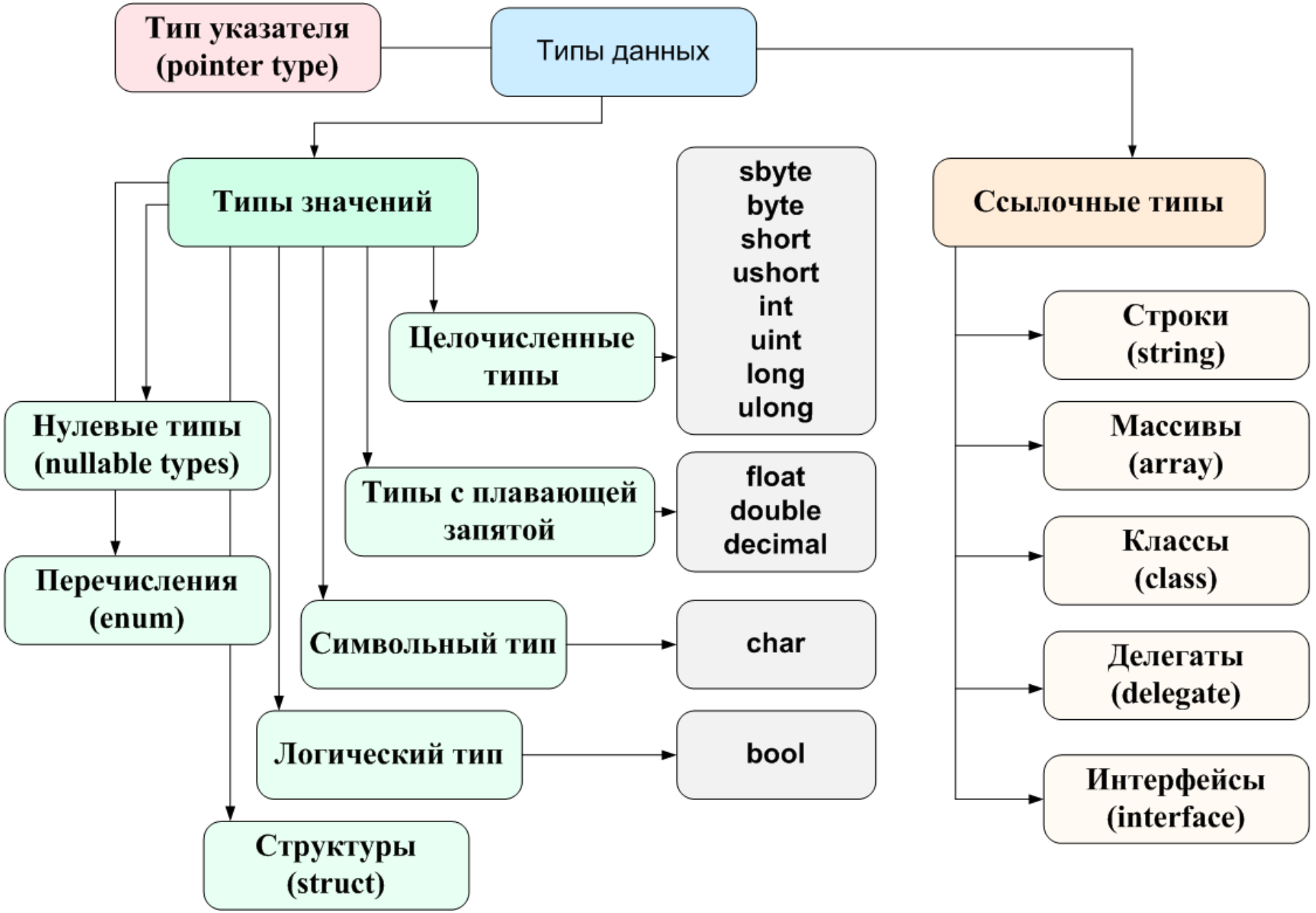
Символьный тип данных.

Другие типы данных.

Тип данных определяют

- множество значений, которые может принимать объект (экземпляр этого типа);
- множество операций, которые допустимо выполнять над ним;
- способ хранения объектов в оперативной памяти.

Под типом понимается ***классы***, ***интерфейсы***, ***структуры, перечисления и делегаты***


Тип указателя (pointer type)
Типы данных
Типы значений
Ссылочные типы
Целочисленные типы
sbyte
byte
short
ushort
int
uint
long
ulong
Нулевые типы (nullable types)
Типы с плавающей запятой
float
double
decimal
Перечисления (enum)
Символьный тип
char
Логический тип
bool
Структуры (struct)
Строки (string)
Массивы (array)
Классы (class)
Делегаты (delegate)
Интерфейсы (interface)

Название	.NET Class	Наличие знака	Размер в битах	Диапазон значений
byte	Byte	-	8	от 0 до 255
sbyte	Sbyte	+	8	от -128 до 127
short	Int16	+	16	от -32 768 до 32 767
ushort	Uint16	-	16	от 0 до 65 535
int	Int32	+	32	от -2 147 483 648 до 2 147 438 648
uint	Uint32	-	32	от 0 до 4 294 967 295
long	Int64	+	64	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
ulong	Uint64	-	64	от 0 до 18 446 744 073 709 551 615

Название	.NET Class	Наличие знака	Размер в битах	Диапазон значений
float	Single	+	32	от 3.402823e38 до 3.402823e38
double	Double	+	64	от -1.797693134862 32e308 до 1.797693134862 32e308
decimal	Decimal	+	128	от ±1.0 × 10e−28 до ±7.9 × 10e28

```
  decimal devidend = 1;
//нижеследующая строка выводит в консоль 1
  Console.WriteLine(devidend);
  decimal devisor = 3;
  devidend = devidend / devisor;
//нижеследующая строка выводит в консоль 0,33333333333333333333333333
  Console.WriteLine(devidend);
//нижеследующая строка выводит в консоль 0,99999999999999999999999999
  Console.WriteLine(devidend * devisor);

//вывод - ошибки округления привели к потере данных

  double doubleDevidend = 1;
//нижеследующая строка выводит в консоль 1
  Console.WriteLine(doubleDevidend);
  double doubleDevisor = 3;
  doubleDevidend = doubleDevidend / doubleDevisor;
//нижеследующая строка выводит в консоль 0,333333333333333
  Console.WriteLine(doubleDevidend);
//нижеследующая строка выводит в консоль 1
  Console.WriteLine(doubleDevidend * doubleDevisor);
```

```
 //определяет является ли символ управляющим
char.IsControl('\t')                    //true

//определяет является ли символ цифрой
char.IsDigit('5')                       //true

//определяет является ли символ бувенным
char.IsLetter('x')                      //true

//определяет находится ли символ в нижнем регистре
char.IsLower('m')                       //true

//определяет находится ли символ в верхнем регистре
char.IsUpper('P')                       //true

//определяет является ли символ знаком пунктуации
char.IsPunctuation(',')                 //true

//определяет является ли символ специальным символом
char.IsSymbol('<')                      //true

//определяет является ли символ пробелом
char.IsWhiteSpace(' ')                  //true

//переводит символ в нижний регистр
char.ToLower('T')                       //t

//переводит символ в верхний регистр
char.ToUpper('t')                       //T
```

Литералы.

Литералами называются постоянные значения, представленные в удобной для восприятия форме.

Литералы – используются для представления значений в исходном коде.

Булевы литералы – true (истина) / false (ложь)

Целые литералы – int
u – uint
l - long

Действительные литералы f или F – float
d или D – double (по умолчанию)
m или M - decimal

Символьные литералы – отдельный символ ‘$’

Строковые литералы – регулярные “Строка”
буквальные @“Строка”

Литерал null

символ	Действие управляющего символа
\a	Звуковой сигнал
\b	Возврат на одну позицию
\f	Переход к началу следующей страницы
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\0	Нуль-символ (символ конца строки)
\’	Одинарная кавычка
\”	Двойная кавычка
\\	Обратная косая черта

Класс String

Представляет текст как последовательность знаков Юникода.

```
string str = "Привет, C#";
```

Метод Object.ToString

Возвращает строковое представление текущего объекта.

```
int a = 100;
string str = a.ToString();
```

```
   string str = "hello :)";
   string anotherString = (string)str.Clone();
   Console.WriteLine(anotherString);
//выводит: "hello :)"
   Console.WriteLine(str.Contains("hello"));
//выводит: true
   Console.WriteLine(str.Insert(6, "world"));
//выводит: "hello world:)"
   Console.WriteLine(str.Remove(5, 1));
//выводит: "hello:)"
   Console.WriteLine(str.Replace(":)", ":("));
//выводит: "hello :("
   Console.WriteLine(str.StartsWith("hell"));
//выводит: "true"
   Console.WriteLine(str.Substring(6));
//выводит: ":)"
   Console.WriteLine(str.ToUpper());
//выводит: "HELLO :)"

   str = "       " + str + "       ";
   str = str.Trim();
   Console.WriteLine(str);
//выводит: "hello :)"
```

Переменные

Понятие переменной.

Правила именования переменных.

Область видимости переменных.

Переменная – это именованный объект, хранящий значение некоторого типа данных.

Объявление переменных

В общем случае, переменная на C# объявляется так:

тип _переменной имя_переменной;

Пример объявления переменных различных типов

```
int A;  double B;    char C;   string Str;     bool X;
```

В одном объявлении можно с помощью запятой указать несколько переменных, при этом все объявленные переменные будут иметь один и тот же тип.

```
int A1, A2, A;    double B, F, D;
```

Инициализация переменных

```
int  A; double B, F, D; char C;  string Str;   bool X;
A = 29;  B = 0.5;  C='@';  Str = "Язык C#";
```

Присвоить значение переменным можно также при их объявлении.

```
int  A=29;  double B=0.5, F=-12.34, D=125.6;    char C='@';
string Str="Язык C#";
bool X=true;
```

Ввод, вывод в консольном приложении.

```
class Program
{
    static void Main()
    {
        int KM=57;
        int M=20;
        Console.WriteLine("Дальность до цели - " + KM + ", " + M);
        Console.WriteLine("Дальность до цели - {0}, {1}", KM, M);
        Console.Read();
    }
```

```
Дальность до цели - 57, 20
Дальность до цели - 57, 20
```

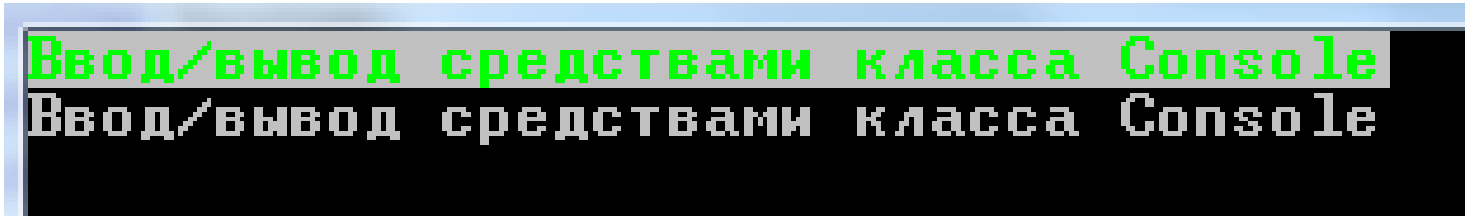
Console.WriteLine("{argNum,width:tmt}", C);

```
Console.WriteLine( "(C) Currency: . . . . . {0:C}\n"+
                   "(D) Decimal:. . . . . {0:D}\n" +
                   "(E) Scientific: . . . . . {1:E}\n" +
                   "(F) Fixed point:. . . . . {1:F}\n" +
                   "(G) General:. . . . . {0:G}\n" +
                   "(N) Number: . . . . . {0:N}\n" +
                   "(P) Percent:. . . . . {1:P}\n" +
                   "(R) Round-trip: . . . . . {1:R}\n" +
                   "(X) Hexadecimal:. . . . . {0:X}\n",
 -123, -123.45f);
```

```
(C) Currency: . . . . . ($123.00)
(D) Decimal:. . . . . -123
(E) Scientific: . . . . . -1.234500E+002
(F) Fixed point:. . . . . -123.45
(G) General:. . . . . -123
(N) Number: . . . . . -123.00
(P) Percent:. . . . . -12,345.00 %
(R) Round-trip: . . . . . -123.45
(X) Hexadecimal:. . . . . FFFFFF85
```

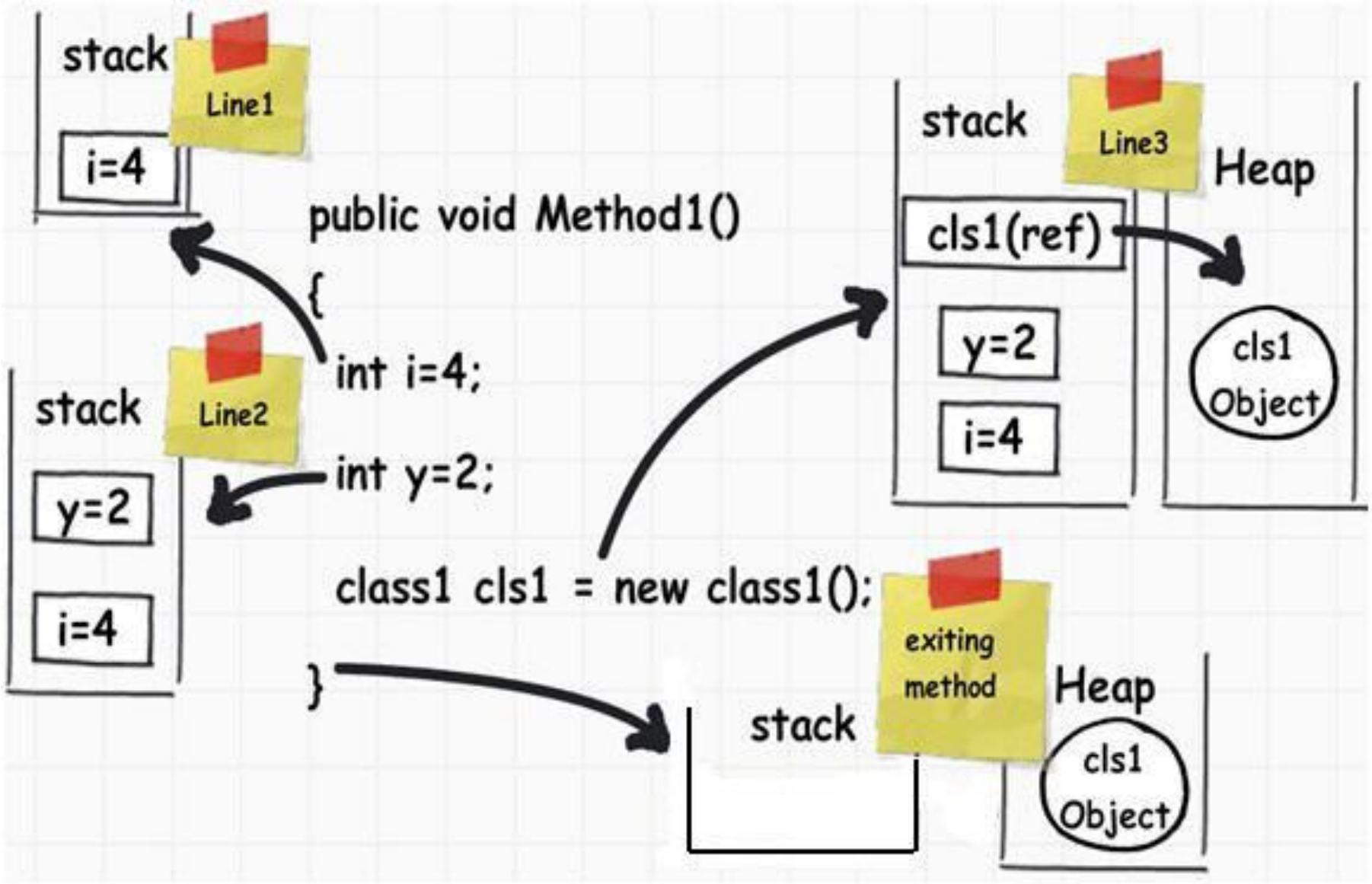
http://msdn.microsoft.com/ru-ru/library/vstudio/aakt1eab(v=vs.100).aspx

```
    // Title  возвращает (устанавливает) текст заголовка окна консоли.
Console.Title = "Ввод/вывод средствами класса Console";
    // BackgroundColor – возвращает или устанавливает фоновый цвет
    // выводимого в консоль текста
Console.BackgroundColor=ConsoleColor.Gray;
    //CursorVisible  –  возвращает  или  устанавливает  значение
    //индикатора  видимости  курсора
Console.CursorVisible = false;
    // ForegroundColor – возвращает или устанавливает фоновый цвет
    // выводимого в консоль текста
Console.ForegroundColor=ConsoleColor.Green;
Console.WriteLine("Ввод/вывод средствами класса Console");
   // ResetColor – сбрасывает значение цвета текста и фона
Console.ResetColor();
Console.WriteLine("Ввод/вывод средствами класса Console");
```

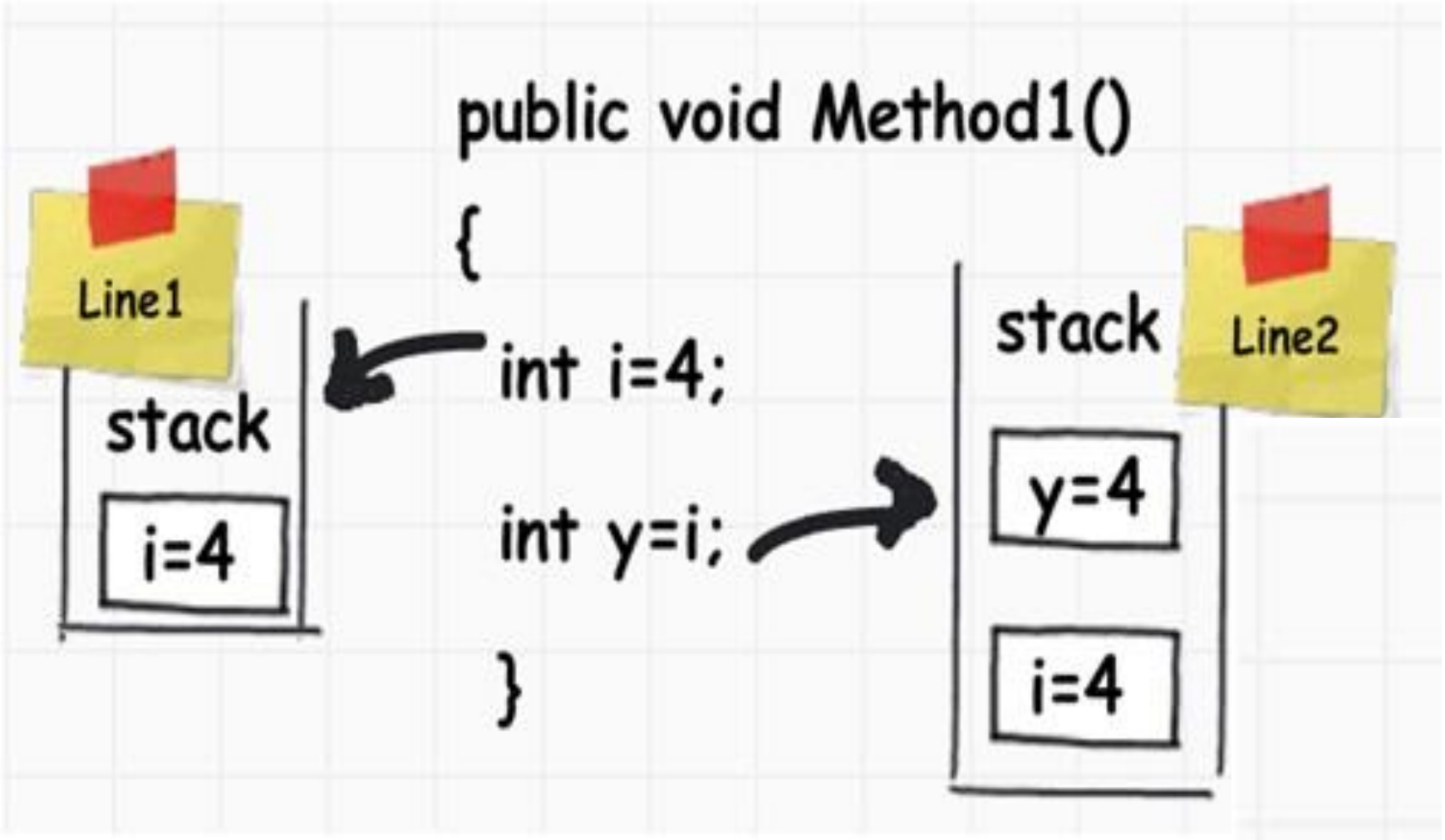


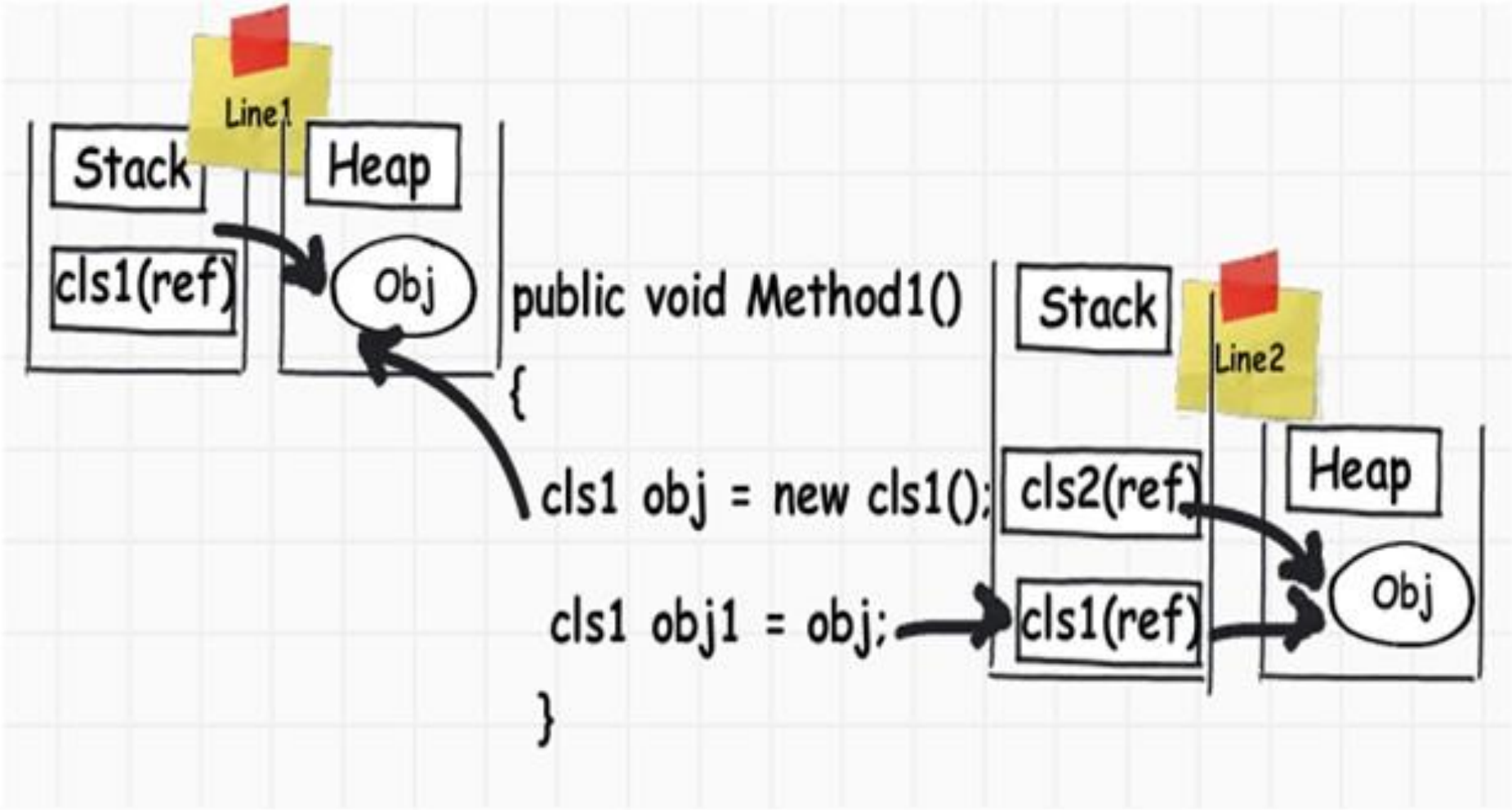
http://msdn.microsoft.com/ru-ru/library/vstudio/System.Console(v=vs.100).aspx

Значимые и ссылочные типы.



!!!


public void Method1()
{
int i=4;
int y=i;
}
Line1
stack
i=4
stack
Line2
y=4
i=4


Line1
Stack
Heap
cls1(ref)
Obj
public void Method1()
{
cls1 obj = new cls1();
cls1 obj1 = obj;
}
Stack
Line2
Heap
cls2(ref)
cls1(ref)
Obj

```
  int x = 10; int y = x;
  x = x - 5;
  Console.WriteLine("X:\t" + x.ToString());
//выводит в консоль: X: 5
  Console.WriteLine("Y:\t" + y.ToString());
//выводит в консоль: Y: 10
```

```
  Point a = new Point(); // Point – класс (ссылочный тип)
  a.X = 10;
  Point b = a;
  b.X = b.X - 5;
  Console.WriteLine("a.X:\t" + a.X.ToString());
//выводит в консоль: a.X: 5
   Console.WriteLine("b.X:\t" + b.X.ToString());
//выводит в консоль: b.X: 5
```

```
   Point a = new Point();      Point b = new Point();
   b.X = a.X = 10;    b.X = b.X - 5;
   Console.WriteLine("a.X:\t" + a.X.ToString());
//выводит в консоль: a.X: 10
   Console.WriteLine("b.X:\t" + b.X.ToString());
//выводит в консоль: b.X: 5
   Console.ReadLine();
```

Преобразование типов

Явное преобразование.

Неявное преобразование.

Класс Convert

Неявно допустимо приводить другу к другу следующие типы данных:

Из типа	К типу
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double

Приведение - это команда компилятору преобразовать результат вычисления выражения в указанный тип.

(целевой_тип) выражение

```
double X= 23.7;
double Y = 17.5;
int R = (int)(X / Y); // 1
```

```
double X= 23.7;
double Y = 17.5;
double R = (int)(X / Y); // 1
```

Допустимо явное приведение друг к другу следующих базовых типов данных:

Из типа	К типу
byte	Sbyte или char
sbyte	byte, ushort, uint, ulong, char
short	sbyte, byte, ushort, uint, ulong, char
ushort	sbyte, byte, short, char
int	sbyte, byte, short, ushort, uint, ulong, char
uint	sbyte, byte, short, ushort, int, char
long	sbyte, byte, short, ushort, int, uint, ulong, char
ulong	sbyte, byte, short, ushort, int, uint, long, char
char	sbyte, byte, short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double

Преобразование типов в выражениях

```
double X = 23;
float Y = 17.5f;
? R = (X / Y);
```

ЕСЛИ один операнд имеет тип **decimal**, ТО и второй операнд продвигается к типу **decimal** (но если второй операнд имеет тип **float** или **double**, результат будет ошибочным).
ЕСЛИ один операнд имеет тип **double**, ТО и второй операнд продвигается к типу **double**.
ЕСЛИ один операнд имеет тип **float**, ТО и второй операнд продвигается к типу **float**.
ЕСЛИ один операнд имеет тип **ulong**, ТО и второй операнд продвигается к типу **ulong** (но если второй операнд имеет тип **sbyte**, **short**, **int** или **long**, результат будет ошибочным).
ЕСЛИ один операнд имеет тип **long**, ТО и второй операнд продвигается к типу **long**.
ЕСЛИ один операнд имеет тип **uint**, а второй — тип **sbyte**, **short** или **int**, ТО оба операнда продвигаются к типу **long**.
ЕСЛИ один операнд имеет тип **uint**, ТО и второй операнд продвигается к типу **uint**.
ИНАЧЕ оба операнда продвигаются к типу **int**.

Класс Convert

```
//выводим пользователю сообщение о том,
//что необходимо ввести целое число в консоль
  Console.Write("Введите целое число: ");
//получаем строку из консоли в строковую переменную
  string numberString = Console.ReadLine();

//конвертируем строковое значение в числовое (тип int)
  int number = Convert.ToInt32(numberString);
  int number = Int32.Parse(numberString);   // 2-ой способ

//конвертируем строковое значение в числовое (тип double)
  double numberD = Convert.ToDouble(numberString);
//выводим результат
  Console.WriteLine(«Успешно отконвертирована в тип данных int!");
  Console.WriteLine("Число = " + number);
```

http://msdn.microsoft.com/ru-ru/library/system.convert%28v=vs.110%29.aspx

Операторы

Арифметические операторы.

Операторы отношений.

Логические операторы.

Оператор присваивания.

Приоритет операторов.

Арифметические операторы

Группа		Оператор	Выполняемая операция
Бинарные	Мультипликативные	*	Умножение
		/	Деление
		%	Остаток от деления
	Аддитивные	+	Сложение
		-	Вычитание
Унарные		+	Унарный плюс
		-	Унарный минус (Отрицание)
		++	Инкремент
		--	Декремент

Операторы отношений

Логические операторы

Оператор	Действие	Оператор	Действие
= =	**Равно**	&	**И**
!=	**Не равно**	|	**ИЛИ**
>	**Больше**	^	**Исключающее ИЛИ**
<	**Меньше**	&&	**Сокращенное И**
>=	**Больше или равно**	||	**Сокращенное ИЛИ**
<=	**Меньше или равно**	!	**НЕ**

|	Побитовый оператор ИЛИ сравнивает каждый бит его первого операнда к соответствующему биту его второго операнда. **Если один из битов равен 1, соответствующий бит результата устанавливается в 1. В противном случае соответствующий бит результата устанавливается в 0.**
&	Оператор побитового и сравнивает каждый бит первого операнда его к соответствующему биту его второго операнда. **Если оба бита равны 1, соответствующий бит результата устанавливается в 1. В противном случае соответствующий бит результата устанавливается в 0.**
^	Побитовое-исключающее ИЛИ - оператор сравнивает каждый бит его первого операнда к соответствующему биту его второго операнда. **Если один бит равен 0, а другой бит равен 1, соответствующий бит результата устанавливается в 1. В противном случае соответствующий бит результата устанавливается в 0.**

```
int a = 10;
result = a | 5;                    Операция |  // 10 - 1010
Console.WriteLine(result);                      // 5  - 0101
                                                // 15 - 1111  Ответ

 result = a & 3;                    Операция &  // 10 - 1010
 Console.WriteLine(result);                     // 3  - 0011
                                                // 2  - 0010  Ответ

result = a ^ 6;                     Операция ^  // 10 - 1010
 Console.WriteLine(result);                     // 6  - 0110
                                                // 12 - 1100  Ответ
```

```
  int a = 10;
  int b = 1;
  int result = a >> b;
//деление на 2 в степени второго
//операнда, в данном случае в степени 1, то есть просто на 2
  Console.WriteLine(result); // 10/2=5

  result = a << b;
// умножение 2 в степени второго
//операнда, в данном случае в степени 1, то есть просто на 2
 Console.WriteLine(result); //10*2=20
```

Приоритет операторов

```
int a = 10; int b = 1;
int result = a + b * 2;
Console.WriteLine(result); //12
result = (a + b) * 2;
Console.WriteLine(result); //22
result = a + b - 4 * 2;
Console.WriteLine(result); //3
result = (a + (b - 4)) * 2;
Console.WriteLine(result); //14
```

Условия

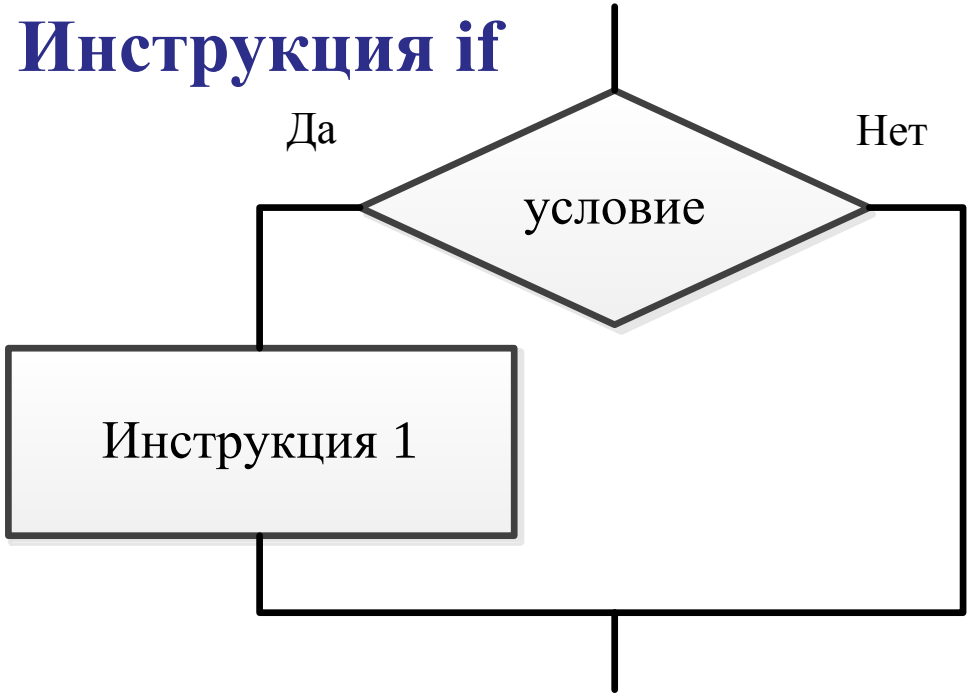
Условный оператор if.

Условный оператор if else.

Условный оператор switch.

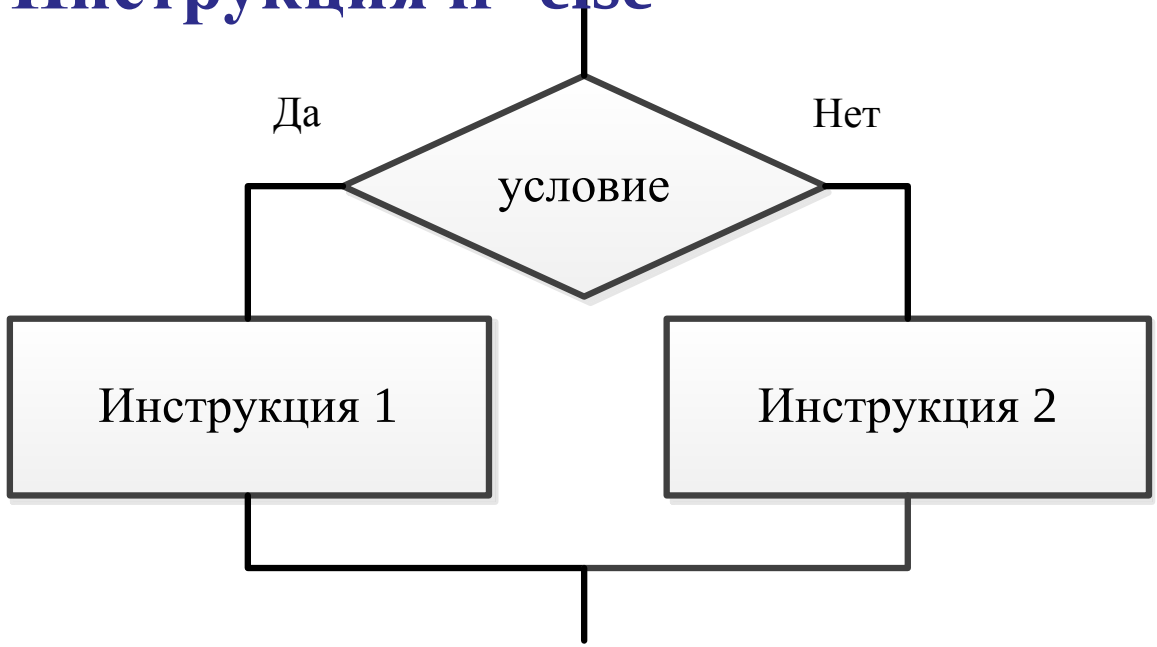
Оператор ?:.

Инструкция if




```
if (условие)
{     инструкция 1 ;
}
```

Инструкция if -else




```
if (условие)
{      инструкция  1;
}
else
{      инструкция   2;
}
```

Особенности оператора switch

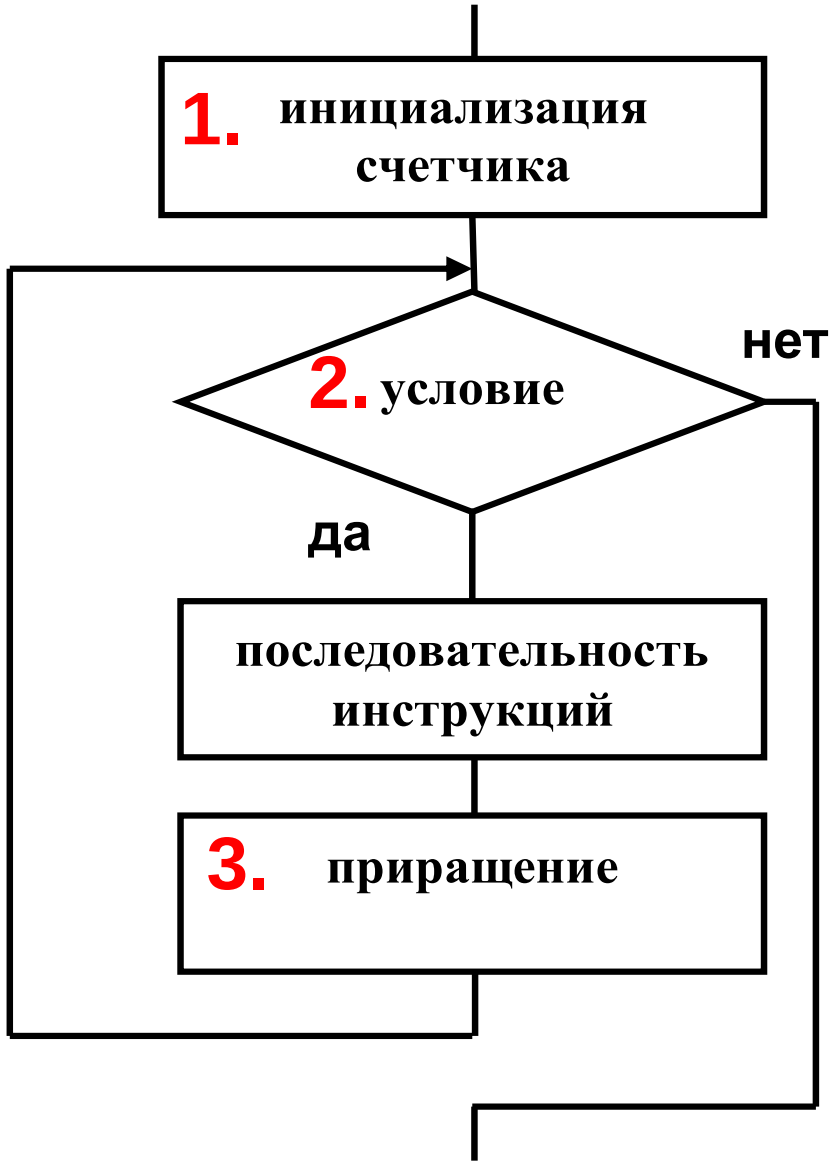
- Для управления оператором switch может быть использовано выражение любого целочисленного, перечислимого или строкового тип.
- В C# должно соблюдаться правило недопущения "**провалов**" в передаче управления ходом выполнения программы (наличие в каждой ветви break).

Тернарная операция

```
int max = 5 > 4 ? 5 : 4;
```

Циклы

- Цикл for.
- Цикл while.
- Цикл do while.
- Цикл foreach.
- Инструкция break.
- Инструкция continue.
- Инструкция goto.



1.Элемент ***«инициализация»*** устанавливает **управляющую переменную цикла** равной **начальному значению**. Эта переменная действует в качестве **счетчика**, который **управляет работой цикла**.

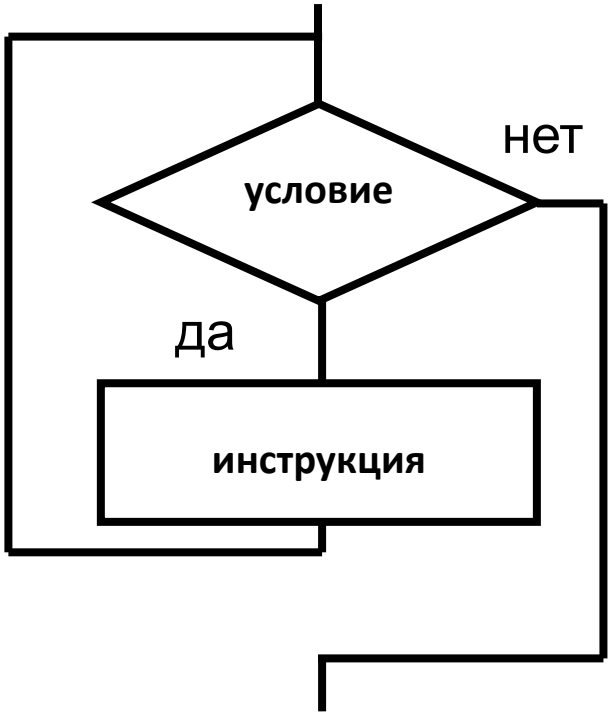
2. В элементе «***условие***» проверяется значение управляющей переменной цикла. **Результат этой проверки определяет, выполнится цикл for еще раз или нет.**

3. Элемент **«приращение»** определяет, как **изменяется значение управляющей переменной цикла** после одного прохода цикла.

Цикл while

Общая форма цикла while имеет такой вид:

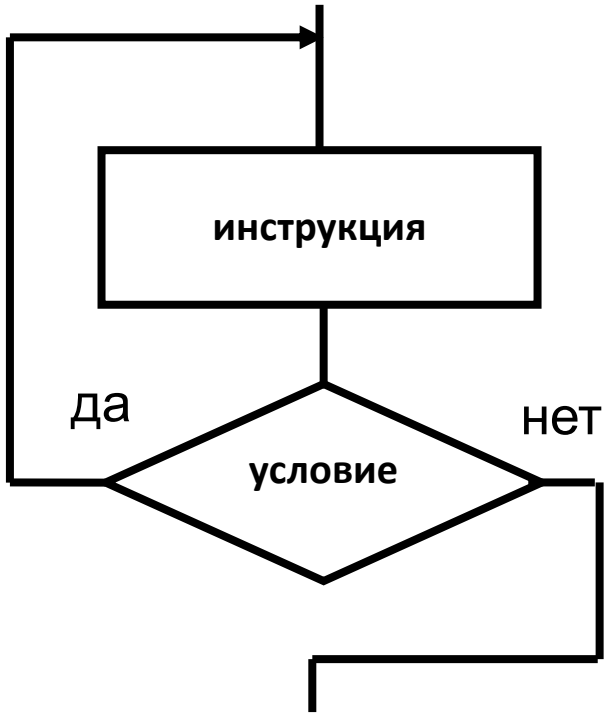
```
while (условие продолжения цикла)
{ инструкция;}

while (условие продолжения цикла)
{
   инструкция 1;
   инструкция 2;
   ….
   инструкция N;
}
```



Цикл do-while

Общий формат имеет такой вид:

```
do
{
   инструкция;
}
while {условие);
```




```
do
{
   инструкция 1;
   инструкция 2;
   ….
   инструкция N;
}
while {условие);
```

Цикл **do-while** всегда выполняется **хотя бы один раз**.

```
class Program
{
    const int counter = 1024 * 1024;
    static void Main(string[] args)
    {
        for (int i = 0; i < counter; ++i)
        {
            Console.WriteLine(i);
        }

        for (int i = 0; i < counter; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

```
using System;
using System.Diagnostics;
class Program
{
    static void Main(string[] args)
    {
        var sw = new Stopwatch();
        sw.Start();
        for (int i = 0; i < 2000000000; i++) { }
        //int i = 0;
        //while (i < 2000000000){++i;}
        Console.WriteLine(sw.ElapsedMilliseconds);
}}
```

for with i++: 1307 for with ++i: 1314
while with i++ : 1261 while with ++i : 1276

```
IL_0000:  ldc.i4.0
IL_0001:  stloc.0
// Start of first loop
IL_0002:  ldc.i4.0
IL_0003:  stloc.0
IL_0004:  br.s       IL_0010
IL_0006:  ldloc.0
IL_0007:  call       void
[mscorlib]System.Console::WriteLine(int32)
IL_000c:  ldloc.0
IL_000d:  ldc.i4.1
IL_000e:  add
IL_000f:  stloc.0
IL_0010:  ldloc.0
IL_0011:  ldc.i4     0x100000
IL_0016:  blt.s      IL_0006
// Start of second loop
IL_0018:  ldc.i4.0
IL_0019:  stloc.0
IL_001a:  br.s       IL_0026
IL_001c:  ldloc.0
IL_001d:  call       void
[mscorlib]System.Console::WriteLine(int32)
IL_0022:  ldloc.0
IL_0023:  ldc.i4.1
IL_0024:  add
IL_0025:  stloc.0
IL_0026:  ldloc.0
IL_0027:  ldc.i4     0x100000
IL_002c:  blt.s      IL_001c
IL_002e:  ret
```

Отсутствуют отличия. IL одинаковый