

C# Basics

1

Getting Started

д.т.н. Емельянов Виталий Александрович

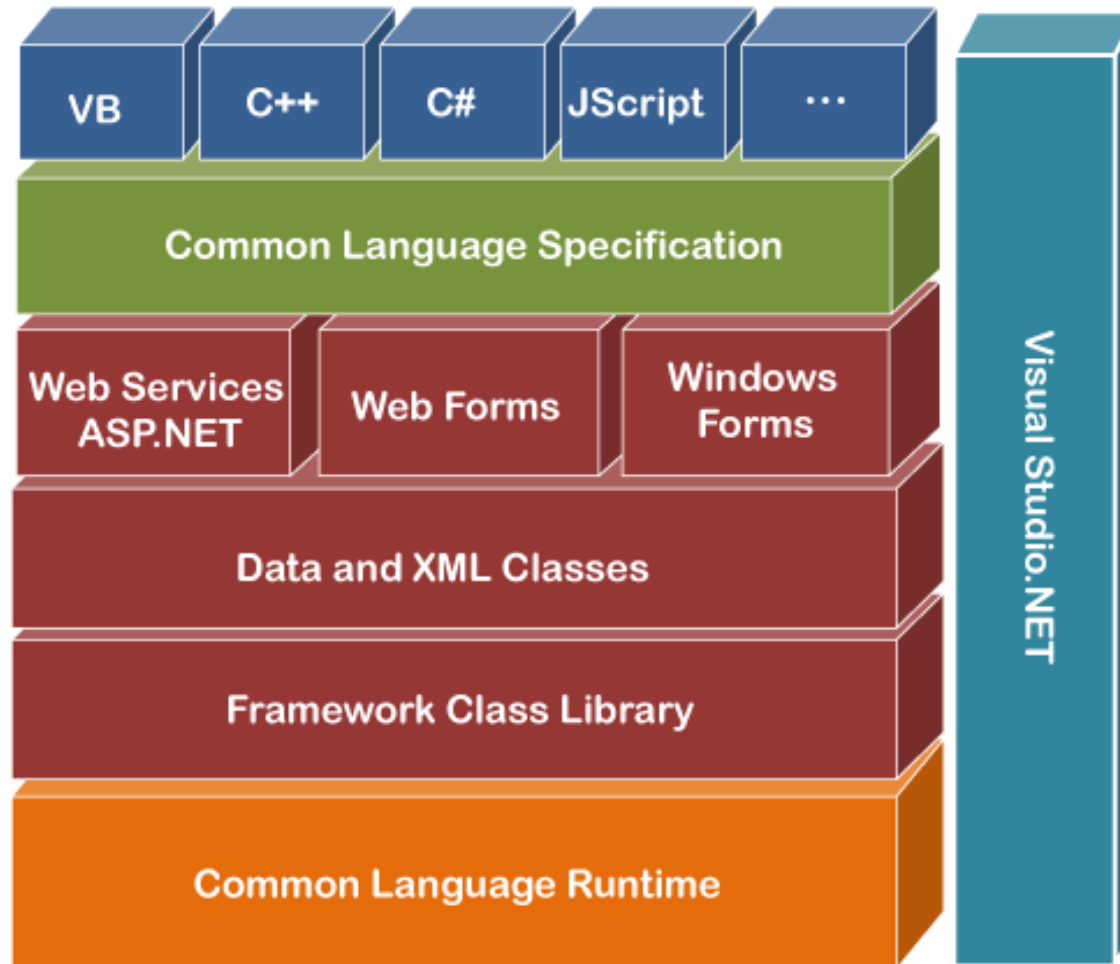
✉: v.yemelyanov@gmail.com



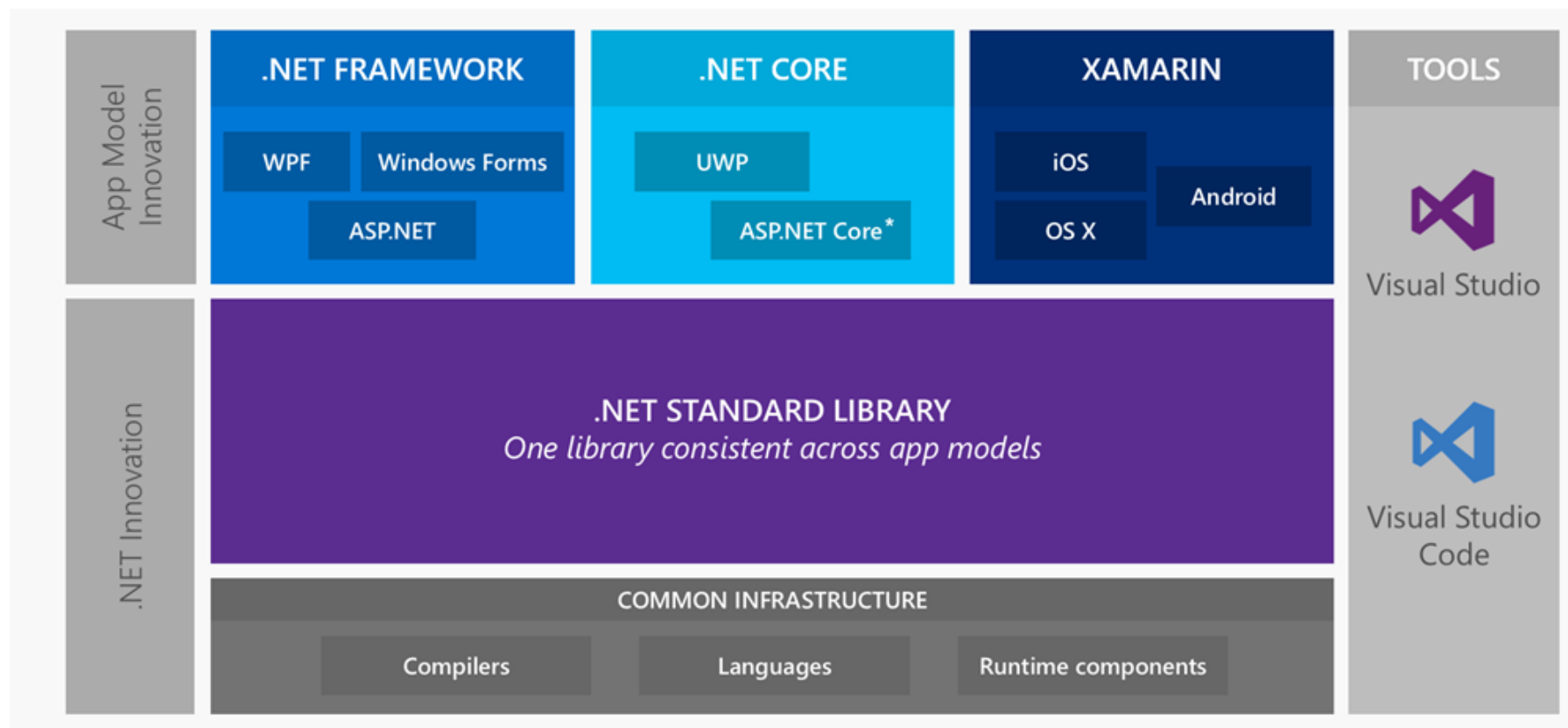
План лекции

- ➔ Платформа .NET
- ➔ Основные управляющие конструкции C#.
- ➔ Работа с массивами на C#.

.NET Framework



Платформа .NET



Особенности .NET

- 1. Поддержка нескольких языков**
- 2. Кроссплатформенность**
- 3. Мощная библиотека классов**
- 4. Разнообразие технологий**
- 5. Производительность**

Алгоритмы

ПЕРВЫМ ВСЕГДА разрабатывается **АЛГОРИТМ** действий



Только после этого алгоритм записывается на языке программирования

***Программный код** – полное, законченное и детальное описание алгоритма на языке программирования.*

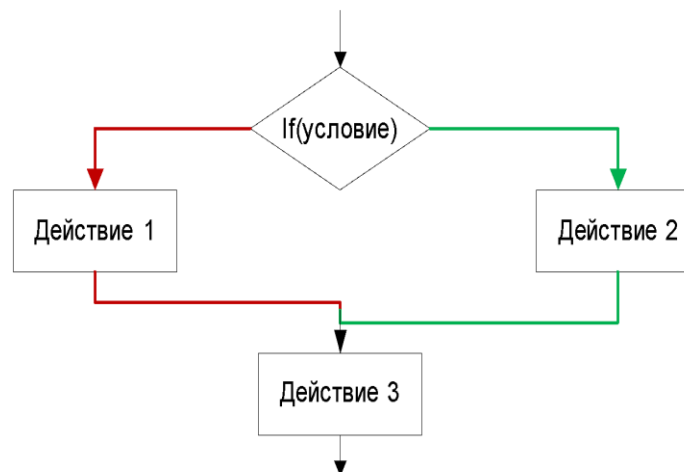
Виды алгоритмов

Алгоритм - это предназначенное для конкретного исполнителя описание последовательности действий, приводящих от исходных данных к требуемому результату

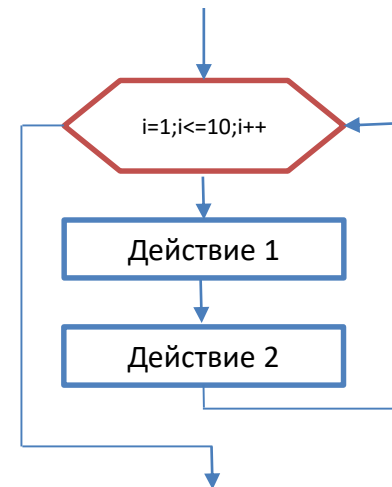
Линейный



Разветвляющийся



Циклический



Способы описания алгоритмов

Естественный язык (сценарии действий)

Пользователь

1. Вводит почтовый адрес

2. Вводит пароль

3. Нажимает кнопку
«Регистрация»

Система

4. Проверяет почтовый адрес

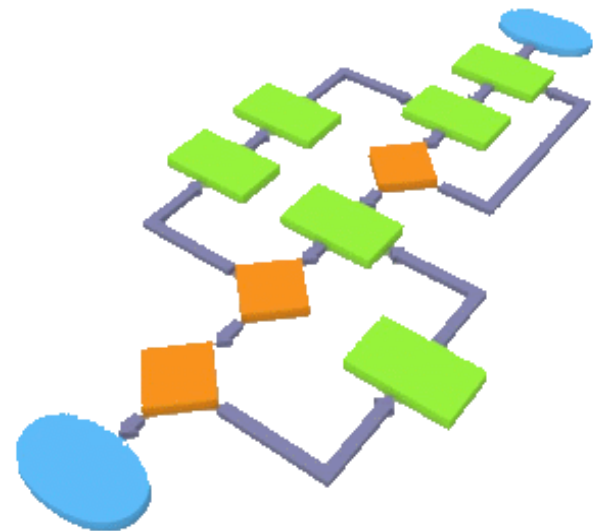
5. Проверяет допустимость пароля

6. Регистрирует пользователя в
системе

7. Отправляет письмо об успешной
регистрации

8. Редирект на страницу аккаунта

Графический (блок-схемы, диаграммы деятельности UML)



Пример алгоритма

- 1) Подойти к терминалу по оплате платежей
- 2) Выбрать оператора связи
- 3) Ввести номер телефона
- 4) Проверить правильность введённого номера
- 5) Вставить денежную купюру в купюроприёмник
- 6) Дождаться сообщения о зачислении денег на счёт
- 7) Получить чек



Выражение, операнды и операции

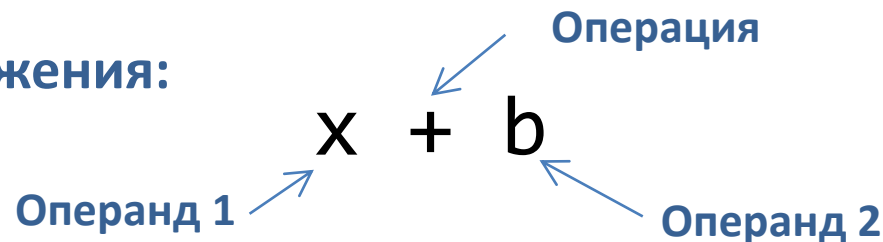
Выражение - это конструкция на языке программирования, результатом которой является определенное значение.

Выражение состоит из **операндов**, объединенных **операциями**.

Операнд - это константа, идентификатор, вызов метода, индексный выражение, выражение выбора элемента.

Операции определяют действия, которые выполняются над операндами. Операции определяются операторами.

Пример выражения:



Операторы

Оператор – законченное выражение, назначением которого является вычисление результата, вызов метода, присвоение значения, проверка условия и т.д.

В C# операторы разделяются знаком ;

Примеры операторов:

C#

```
j++;
```

```
y = x + b;
```

```
Class.Func(par1, par2);
```

Операторы

Составной оператор или **блок** представляет собой несколько операторов, которые объединены фигурными скобками **{ }**:

C#

```
{  
    x1 = y1 + y2;  
    x2 = z1 + z2;    // блок операторов  
}
```

Комментировать код в C# можно 2 способами:

- ➔ если это одна строка то перед ней ставится **//**
- ➔ если строк много – впереди ставится **/*** , а после кода ***/**

Переменные

Переменная – именованная область памяти, в которой хранятся данные определенного типа.

Переменная – величина, которая во время работы программы может изменять свое значение.

Особенности:

- ➔ Перед использованием любая **переменная должна быть описана (объявлена)**.
- ➔ Для каждой переменной задается **имя и тип данных**.
- ➔ Имя переменной служит для обращения к области памяти, в которой хранится значение.
- ➔ Имя переменной должно быть **уникальным** в своей области действия

Типы данных

Тип данных определяет:

- ➡ внутреннее представление данных, т.е. *множество их возможных значений*
- ➡ Допустимые действия над данными => *операции и функции*

Типы данных

Название	Обозначение	Размер	Диапазон	Пример на C#
Строка	string	-	-	string s = "hello";
Символ	char	2 байта	любой символ Unicode	char c = 'a';
Байт	byte	1 байт	0...255	byte b = 0;
Целое число	int	4 байта	от -2147483648 до 2147483647	int i = 10;
Число одиночной точности с плавающей запятой	float	4 байта	Примерно от +1.5E-45 до +3.4E+38	float f = 10.5;
Число двойной точности с плавающей запятой	double	8 байт	Примерно от +5.0E-324 до +1.7E+308	double f = 2.0;
Истина/Ложь (булев тип)	bool	1 байт	true или false	bool b = true;

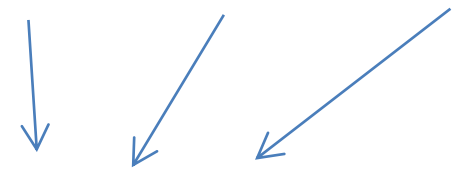
Объявление переменных и констант

С#

тип данных

имя

значение



```
int i = 10; // целочисленная переменная с именем i, которая =10
string s;    // строковая переменная (строка) с именем s
double t = 0.2; // вещественная переменная с именем t

const string s = "строка"; // строковая константа
const int i = 20;          // целочисленная константа
```


Область действия и время жизни

- ➔ Переменные описываются внутри какого-л. блока (класса, метода или блока внутри метода)
 - Переменные, описанные непосредственно внутри класса, называются **полями класса**.
 - Переменные, описанные внутри метода класса, называются **локальными переменными**.
- ➔ **Область действия переменной** - область программы, где можно использовать переменную. Область действия переменной начинается в точке ее описания и длится до конца блока, внутри которого она описана.
- ➔ **Время жизни**: переменные создаются при входе в их область действия (блок) и уничтожаются при выходе.

Операции в С#

Арифметические операции:

- ➔ обозначаются знаками: "+"; "-"; "*"; "/" и "%" (остаток от деления).
- ➔ требуют **числовых** типов данных - целых или вещественных чисел.

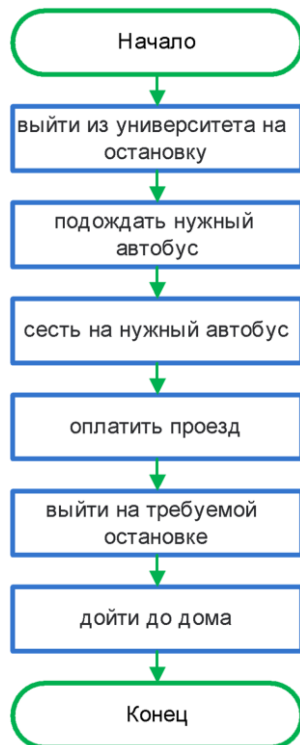
Логические операции:

- ➔ возвращают результат "истина" (true) или "ложь" (false) .

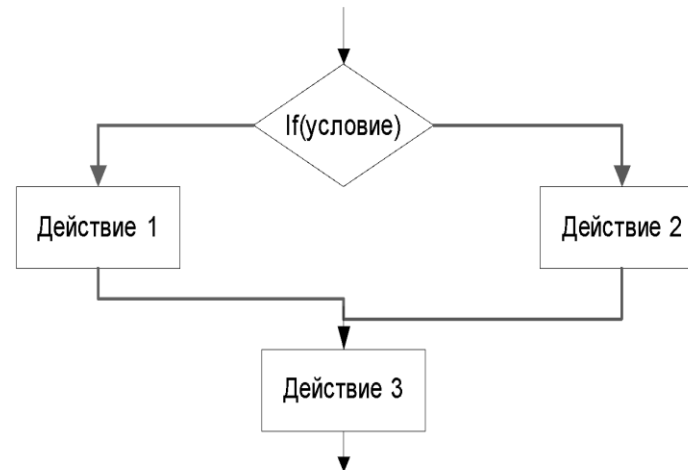
Логическая операция	С#
И (AND) - логическое умножение	&
ИЛИ (OR) - логическое сложение	
Эквивалентность (равенство)	==
Неэквивалентность (неравенство)	!=
Инверсия (отрицание):	!
Сравнение (отношение)	>, <, <=, >=

Виды алгоритмов

Линейный



Разветвляющийся




Циклический



Линейный алгоритм

C#



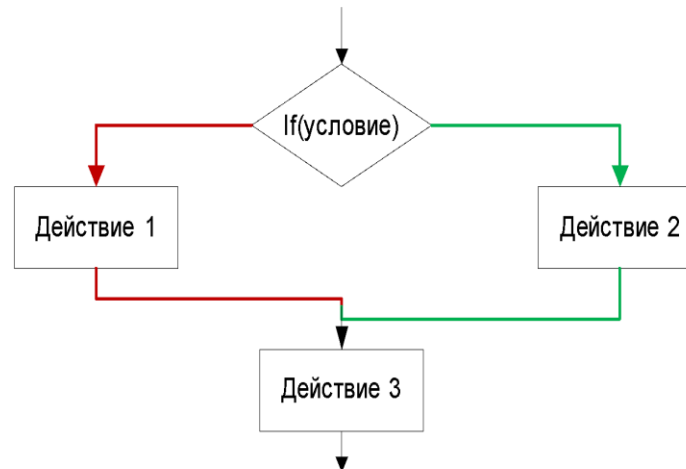
```
{  
    x1 = y1 + y2;  
    x2 = z1 + z2;      // блок операторов  
    j++;  
    y = j*(x1 + b2);  
}
```

Виды алгоритмов

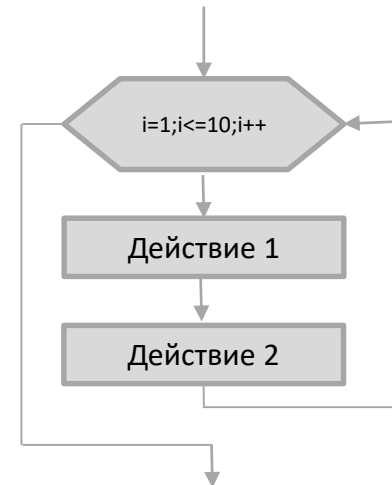
Линейный



Разветвляющийся



Циклический



Ветвления (условия) в С#

Оператор условия `if` предусматривает проверку условия, в зависимости от выполнения которого будет выполняться один блок кода (если условие истинно - `true`) или другой блок кода (если условие ложно - `false`).

Синтаксис:

С#

```
if (условие)
{
    //код для выполнения действий,
    //если условие выполняется
}
else
{
    //код для выполнения действий,
    //если условие не выполняется
}
```

Блок-схема



Пример условия

C#

```
if (a < 10) //проверяется условие: значение переменной a больше 10?
{
    j++;      //блок операторов выполняется,
    b = a + c; //если значение переменной a<10
}
else
    j--;      //оператор выполняется,
              //если значение переменной a>=10
```

Пример условия

C#

```
bool PaymentStatus=false;    //переменная PaymentStatus - состояние
                              //оплаты (оплачен заказ или нет)

•
•

if (PaymentStatus==true)    //проверяем оплату: оплачен заказ - true
                           //не оплачен заказ - false
{
    Console.WriteLine("Оплата поступила. Заказ можно выдавать");
}
else
    Console.WriteLine("Заказ необходимо оплатить");
```


Оператор выбора switch

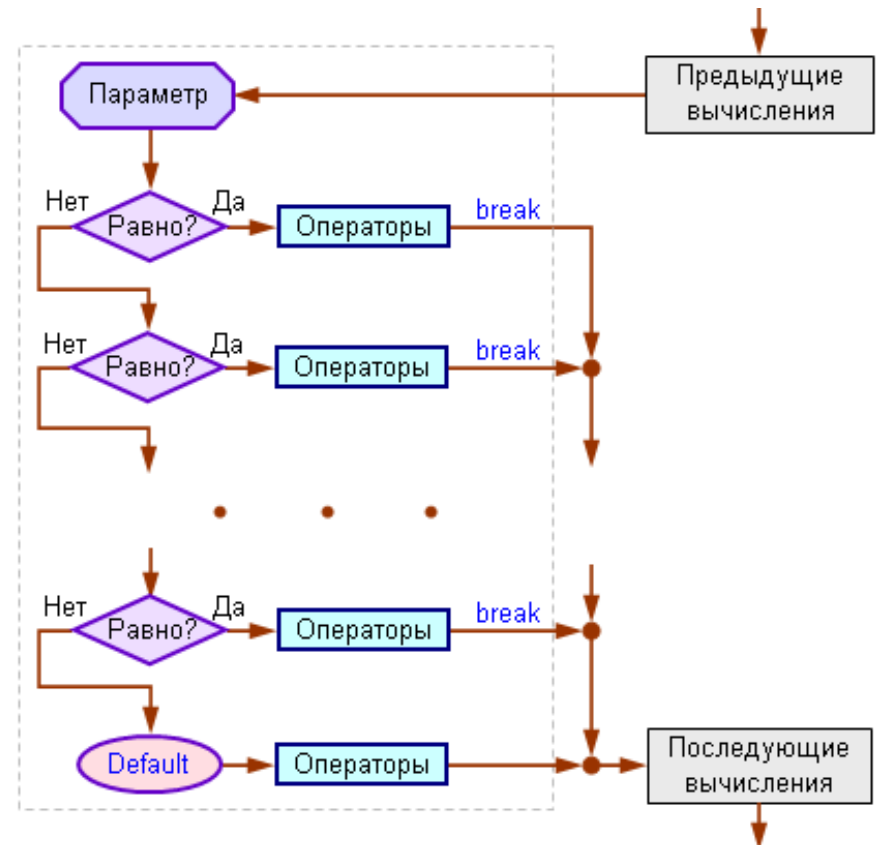
Оператор выбора (**switch**) – оператор, который позволяет выбрать нужное действие (вариант) из множества предлагаемых.

Синтаксис:

С#

```
switch (выражение)
{
    case константное_выражение_1:
        [операторы_1]
        break
    ...
    case константное_выражение_K:
        [операторы_K]
        break
    [default: операторы_N break]
}
```

Блок-схема



Пример switch

C#

```
int Delivery=0; //переменная Delivery - вариант доставки заказа
. . .
switch (Delivery)
{
    case 0:          //сравниваем переменную Delivery со значением 0
        Console.WriteLine(" Доставка курьером в день заказа ");
        break;

    case 1:          //сравниваем переменную Delivery со значением 1
        Console.WriteLine(" Доставка в течение 3-х дней ");
        break;

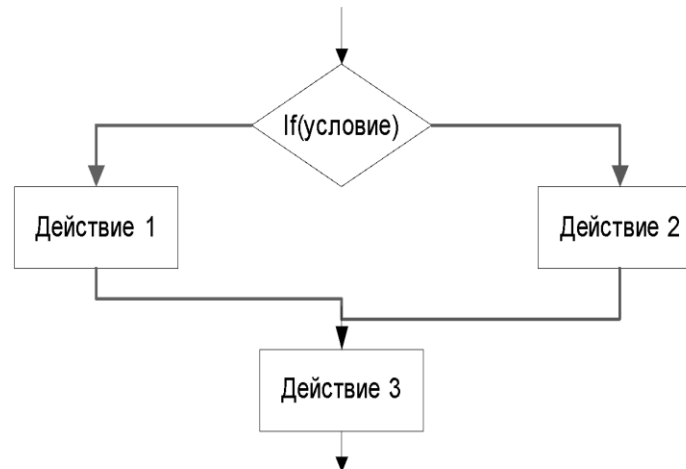
    default:
        Console.WriteLine("Самовывоз. Доставка не требуется");
        break;
}
```

Виды алгоритмов

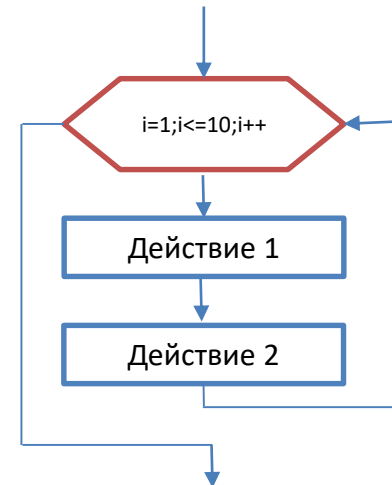
Линейный



Разветвляющийся



Циклический



Циклы

Циклы позволяют многократно выполнять одни и те же действия (тело цикла) определенное количество раз. Тело цикла заключается в фигурные скобки { }.

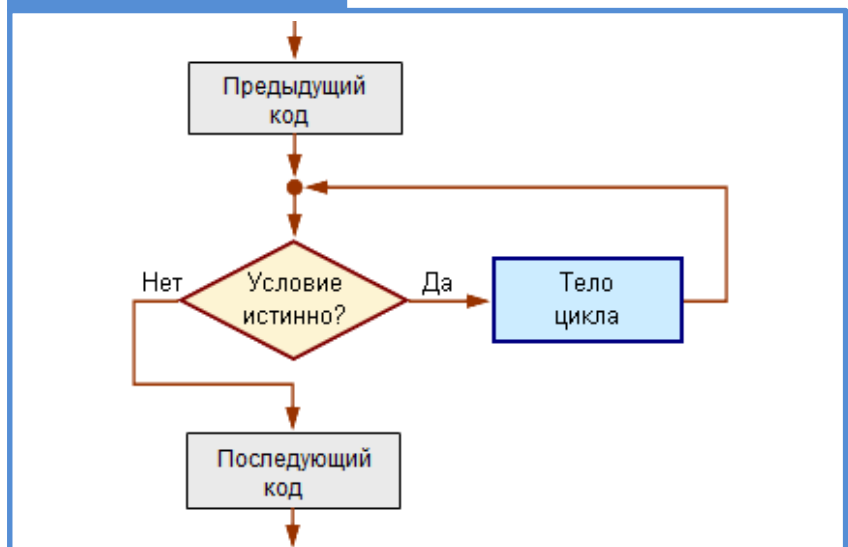
Цикл с предусловием – **while**. Позволяет повторять участок кода (тело цикла) до тех пор, пока условие истинно – **true**.

Синтаксис **while**:

С#

```
while (условие)
{
    //тело цикла
}
```

Блок-схема

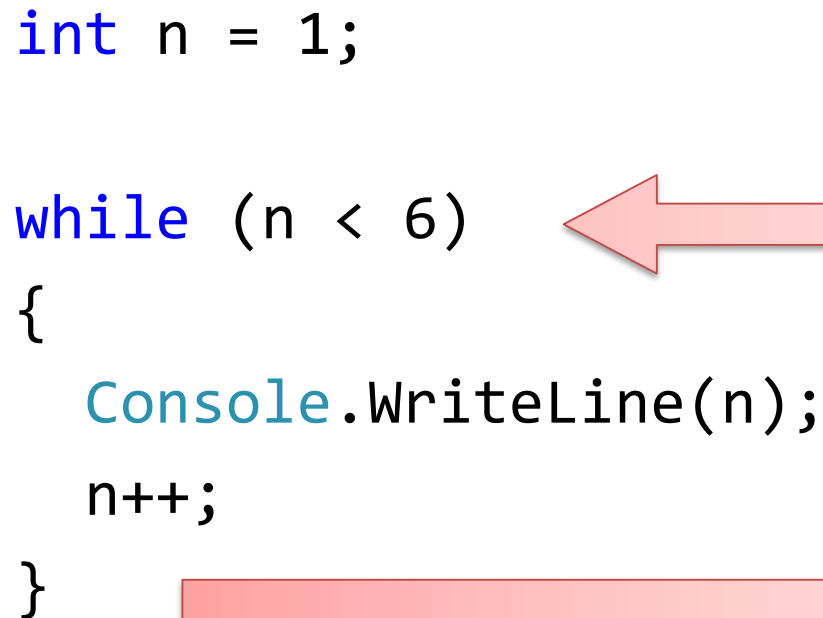


Циклы: пример while

C#

```
int n = 1;

while (n < 6)
{
    Console.WriteLine(n);
    n++;
}
```



Циклы: пример while

C#

```
int Count;           //счетчик обработанных клиентов
int ClientCount = 10; //общее количество клиентов, оформивших заказы
Count = 1;
. . .
while (Count <= ClientCount)
{
    if (PaymentStatus == true)
    {
        Console.WriteLine("Оплата поступила от клиента "+Count);
    }
    else
        Console.WriteLine("Заказ клиента "+Count+ " не оплачен");

    Count++;
}
```

Циклы

Цикл с пост условием `do ... while`

- ➔ Цикл, который проверяет условие завершения работы цикла в конце.
- ➔ Тело такого цикла выполняется, по меньшей мере, один раз.

Синтаксис:

C#

```
do
{
    // тело цикла
}
while (условие);
```

Циклы: пример do while

C#

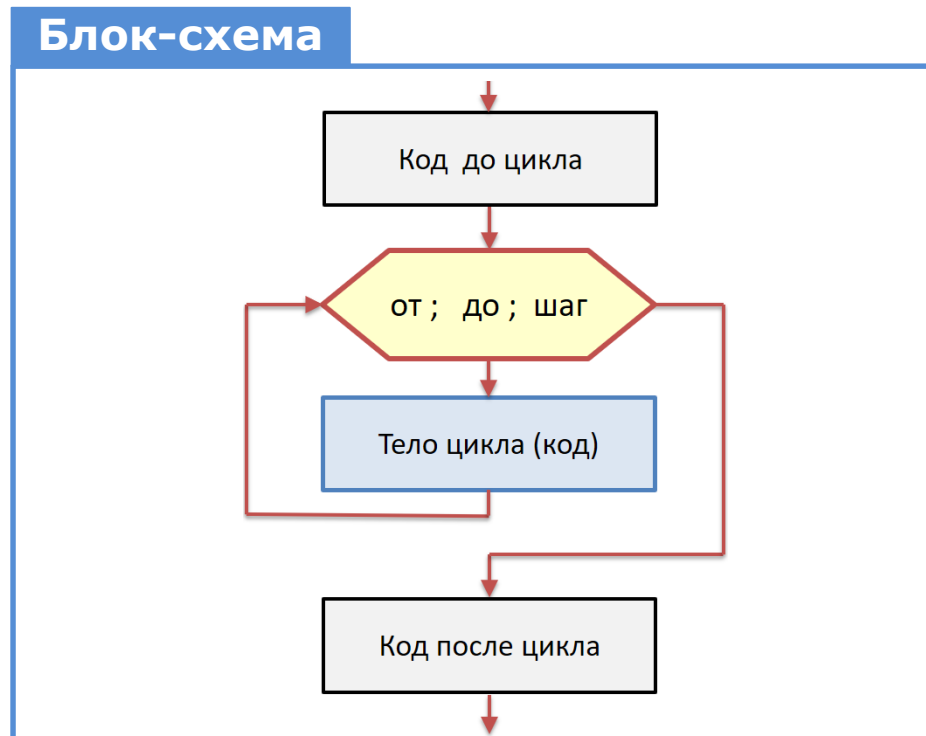
```
int x = 0;

do
{
    Console.WriteLine(x);
    x++;
}
while (x < 5);
```


Циклы

Цикл for:

Цикл **for** позволяет многократно повторять одну и ту же последовательность команд (тело цикла) и прерывать действие цикла при выполнении некоторого условия.



Циклы

Синтаксис:

C#

```
for (инициализатор; условие; оператор изменения инициализатора)
{
    //тело цикла;
}
```

Пример:

C#

```
      от      до      шаг
      └──┬──┘ └──┬──┘ └──┬──┘
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine(i);
}
```

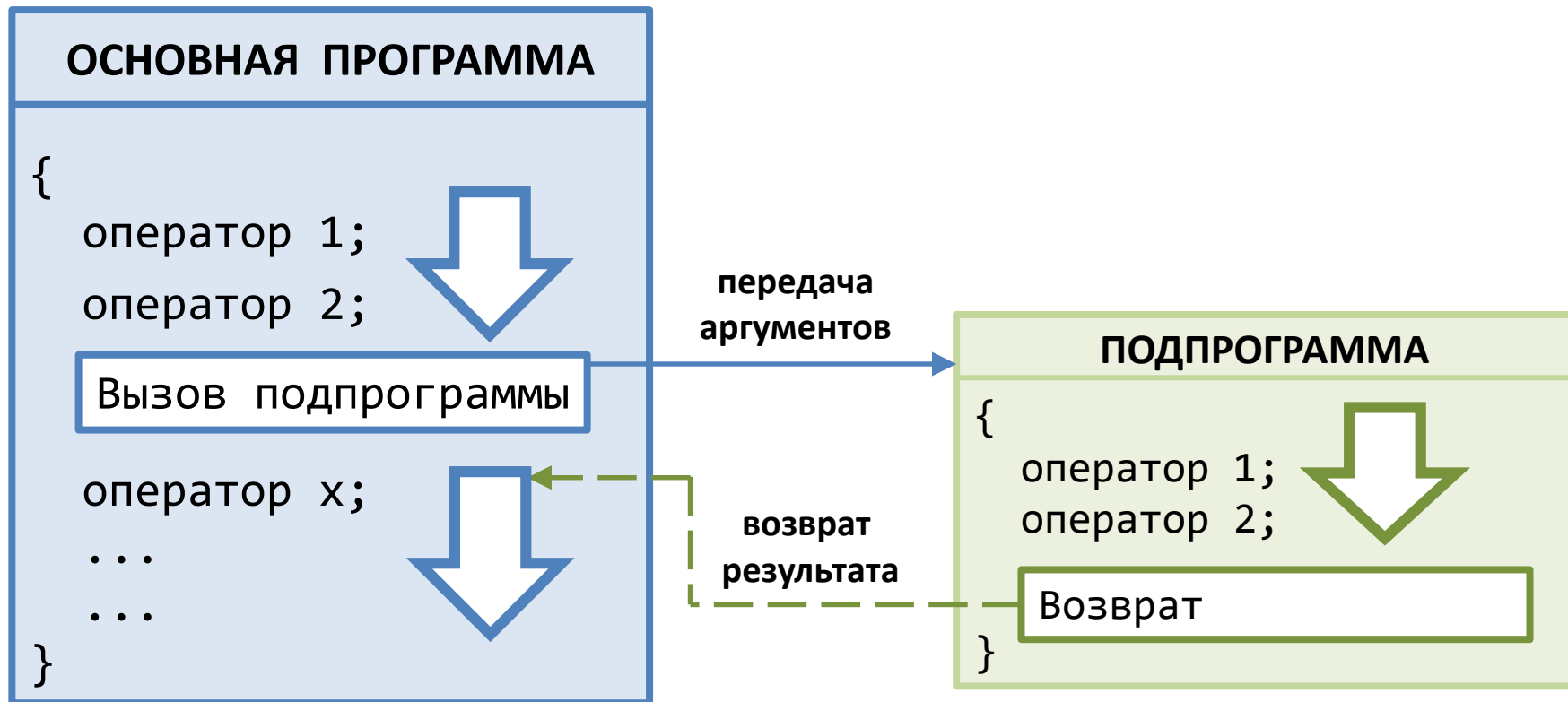
Циклы: пример for

C#

```
int Count; //счетчик обработанных клиентов
int ClientCount = 10; //общее количество клиентов, оформивших заказы
.
.
for (Count=1; Count <= ClientCount; Count++)
{
    if (PaymentStatus == true)
    {
        Console.WriteLine("Оплата поступила. Заказ клиенту
                            номер "+Count+" можно выдавать");
    }
    else
        Console.WriteLine("Заказ клиента номер "+Count+" не оплачен");
}
```

Подпрограммы

Подпрограмма - поименованная часть программы, которая вызывается из определенной точки программы, получает аргументы, решает определенную подзадачу и возвращает управление в точку вызова



Преимущества подпрограмм

Подпрограммы позволяют:

- ➔ сократить объем программы (подпрограмма оформляется один раз и вызывается многократно);
- ➔ выполнить структуризацию программы по функциональности (каждая программа решает свою строго определенную подзадачу);
- ➔ использовать ранее разработанные подпрограммы, размещенные в специальной библиотеке.

Подпрограммы как методы

В языке программирования C#:

- ➔ Подпрограмма может быть определена только в составе класса
- ➔ Подпрограмма, определенная в составе класса, называется **методом**

`void` - если метод не возвращает значение

C#

```
static тип_возвращаемого_объекта имя_Метода(параметры)
{
    /* тело метода (локальные переменные, реализация
        алгоритма решения подзадачи и т.д.) */
}
```

Объявление метода

- ➔ **Объявление метода, не возвращающего значение:**

C#

```
static void MethodX(int parametr1)
{
    /* тело метода
}
```

- ➔ **Объявление метода, возвращающего значение:**

C#

```
static int MethodY(double parametr2)
{
    /* тело метода
}
```

Параметры метода

Параметр

=

[модификатор]

+

тип данных

+

имя_параметра

Модификатор:

- ➔ **если отсутствует - это входной параметр.** Его значение должно быть определено в вызывающей программе. Если метод изменит значение параметра, аргумент в вызывающей программе останется без изменений
- ➔ **out - это выходной параметр.** Его значение определяется в методе и после завершения метода аргумент в вызывающей программе получит это значение
- ➔ **ref - это входной-выходной параметр.** Его значение должно быть определено в вызывающей программе. Если метод изменит значение параметра, аргумент в вызывающей программе получит это значение

Вызов метода

C#

```
static void Main(string[] args)
{
    double x = 10.0, y;
    MethodP(x, out y);
}
```

```
static void MethodP(double a, out double b)
{
    b = a + 10;
}
```

← **Вызов метода**

← **Объявление метода**

Примеры неверного вызова:

C#

```
string s;
```

```
MethodP(x);           // ОШИБКА: не совпало количество аргументов и параметров
```

```
MethodP(s, out y);    // ОШИБКА: не совпали типы данных аргумента и параметра
```


```
MethodP(x, y);        // ОШИБКА: не совпал модификатор аргумента и параметра
```

Возврат из метода

Возврат из метода выполняется оператором **return**

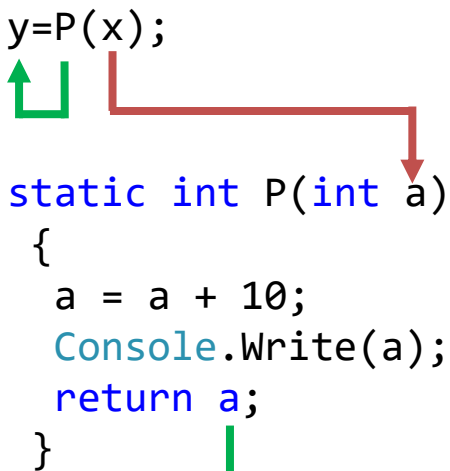
C#

```
int x = 5, y;  
P(x);  
.  
.  
static void P(int a)  
{  
    a = a + 10;  
    Console.Write(a);  
    return;  
}
```



C#

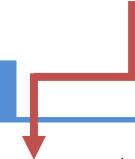
```
y=P(x);  
static int P(int a)  
{  
    a = a + 10;  
    Console.Write(a);  
    return a;  
}
```



ОШИБКА: метод обязан вернуть целое число

C#

```
static int P(int a)  
{  
    a = a + 10;  
    Console.Write(a);  
    return;  
}
```



Массивы

Массив – структура данных, содержащая несколько объектов одного типа данных.



Статический массив

Размер массива определяется при создании массива и в процессе исполнения программы не изменяется



Динамический массив

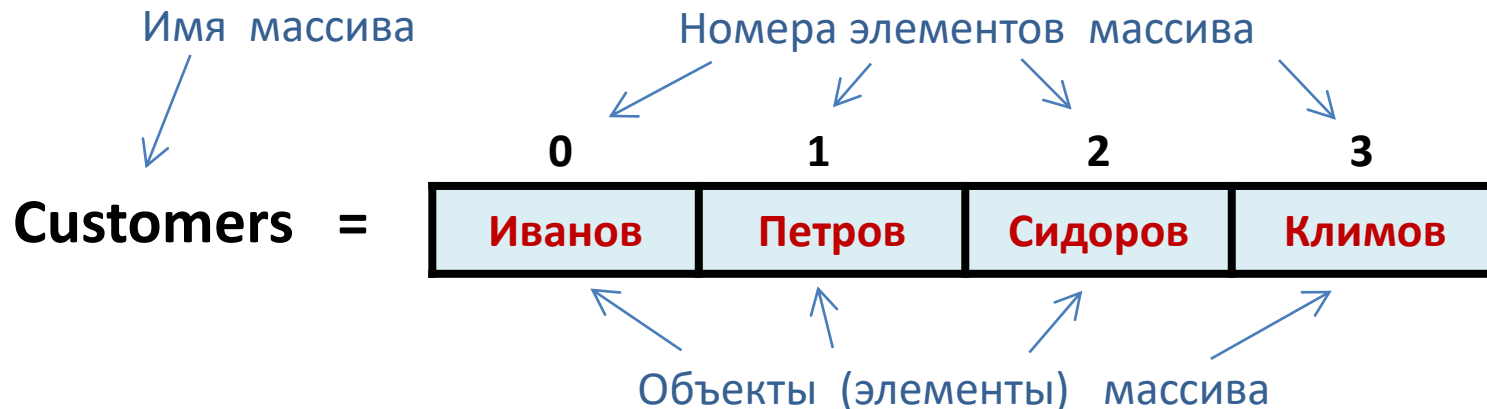
Размер массива может меняться в процессе исполнения программы. Объекты могут добавляться в массив и удаляться из него

Примечание: аналогом одномерного массива в математике является вектор
аналогом двумерного массива в математике является матрица

Свойства массивов

- ➔ Индексация элементов массивов начинается с нуля.
- ➔ Размер (длина) массива - это количество элементов в массиве.
- ➔ Размерность массива - количество измерений (двухмерный, трехмерный и т.д.)

Пример одномерного массива:



Свойства массивов

Пример двумерного массива:

Номера элементов массива

The diagram illustrates a 2D array structure. It features a grid of 3 rows and 4 columns. The columns are indexed 0, 1, 2, and 3 from left to right. The rows are indexed 0, 1, and 2 from top to bottom. Each cell in the grid contains a name in red text. Blue arrows point from the column indices to the top of each column and from the row indices to the left of each row. Additional blue arrows point from the text 'Объекты (элементы) массива' at the bottom to each of the 12 individual cells in the array.

	0	1	2	3
0	Иванов	Петров	Сидоров	Климов
1	Джонсон	Питерсон	Джексон	Робертсон
2	Ченг	Ванг	Конг	Нгуен

Объекты (элементы) массива

Объявление и инициализация массивов

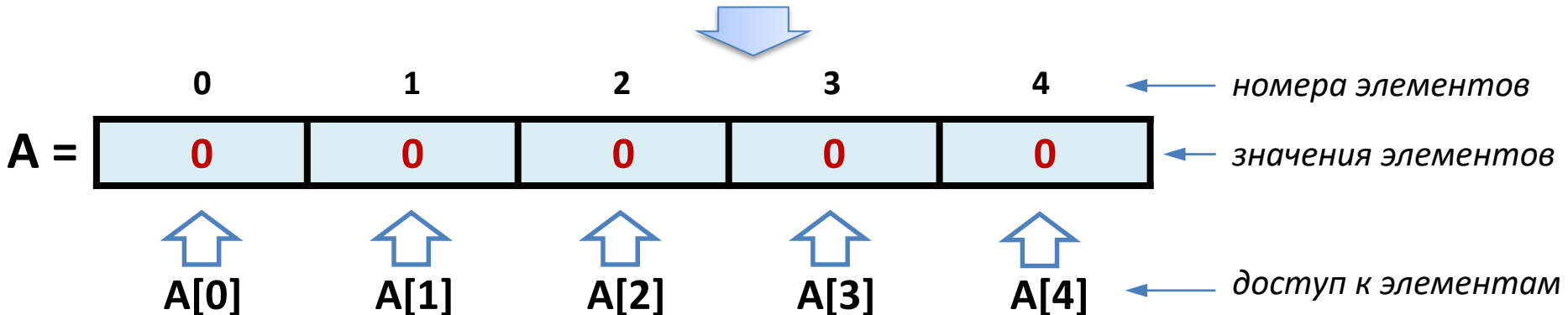
Синтаксис объявления массива:

Тип_данных[] ИмяМассива = new Тип_данных[Количество_элементов_массива]

Пример объявления одномерного массива:

C#

```
int[] A = new int[5];           // целочисленный массив A
string[] strA = new string[4];  // строковый массив strA
```

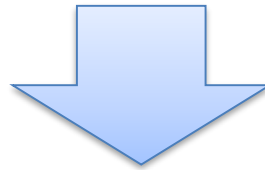


Объявление и инициализация массивов

Пример инициализации одномерного массива:

C#

```
int[] A = new int[5] {5,2,8,9,1};  
string[] strA = new string[4] { "Иванов", "Петров", "Сидоров", "Климов" };
```



A =

5	2	8	9	1
---	---	---	---	---

strA =

Иванов	Петров	Сидоров	Климов
--------	--------	---------	--------

Объявление и инициализация массивов

Пример объявления и инициализации двумерного массива:

C#

```
int[,] array2D = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
```



array2D =

1	2
3	4
5	6

Обращение к элементам массива

Для обращения к элементам массива используются номера элементов (индексы):

C#

```
array2D[2, 0] = 10;
```

```
strA[2] = "Кузнецов";
```

array2D

	0	1
0	1	2
1	3	4
2	10	6

strA =

Иванов	Петров	Кузнецов	Климов
0	1	2	3

Класс Array

Класс Array - предоставляет методы для создания, изменения, поиска и сортировки массивов.

➔ пространство имен **System**

Пример объявления массива A класса Array:

C#

```
Array A = new int[5];
```

Класс Array

Свойства

Length	длина (количество элементов) массива класса Array
Rank	ранг (размерность) массива класса Array
IsFixedSize	значение, показывающее, имеет ли Array фиксированный размер

Методы

Sort(A)	Сортирует элементы во всем одномерном массиве A
Reverse(A)	Перестановка элементов в обратной последовательности
Copy(A,B,N)	Копирование N объектов из массива A в массив B
Clear(A,Start,N)	становка N элементов массива A в “нулевое” значение, начиная с позиции Start
IndexOf(A,Z)	Целое число – позиция первого вхождения объекта Z в массив A
LastIndexOf(A,Z)	Целое число – позиция последнего вхождения объекта Z в массив A
GetLength	Получает целое число, представляющее количество элементов в заданном измерении массива Array
GetValue(index)	получает значение, хранящееся в позиции index массива Array
ConvertAll	Преобразует массив одного типа в массив другого типа

Класс Array

Пример работы с массивами используя класс Array:

C#

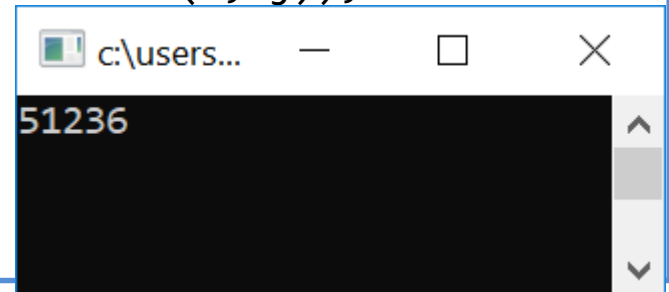
```
switch (A.Rank)
{
    case 1:
        for (int i = 0; i < A.GetLength(0); i++)
            Console.Write(A.GetValue(i));

        break;

    case 2:
        for (int i = 0; i < A.GetLength(0); i++)
        {
            for (int j = 0; j < A.GetLength(1); j++)
                Console.Write(A.GetValue(i, j));

            break;
        }

    default: break;
}
```



Класс Array

Пример работы с методами класса Array:

C#

```
int[] num = { 4, -5, 2, 0, 23 };  
  
Array.Reverse(num); // изменяем порядок элементов  
  
Array.Sort(num); // сортируем массив  
  
int[] oldmass = { 3, 4, 1 };  
int[] newmass = new int[4];  
oldmass.CopyTo(newmass, 1);
```

Динамические типизированные массивы

Динамический типизированный массив определен как класс `List` в пространстве имен `System.Collections.Generic`

Синтаксис объявления массива:

```
List<Тип_данных_элементов> ИмяМассива = new List<Тип_данных_элементов>();
```

Пример объявления динамического типизированного массива:

C#

```
List<int> A = new List<int>(); // целочисленный динамический массив A  
List<string> strA = new List<string>(); // строковый динамич. массив strA
```

Динамические типизированные массивы

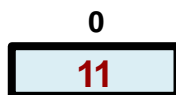
Методы класса List	
A.Add(Z)	Добавление объекта Z в конец массива A
A.AddRange(M)	Добавление всех элементов массива M в конец массива A
A.Insert(Start,Z)	Добавление объекта Z в массив A в позицию Start
A.InsertRange(Start,M)	Добавление всех элементов массива M в массив A с позиции Start
A.Remove(Z)	Удаление объекта Z из массива A
A.RemoveAt(Start)	Удаление объекта из массива A в позиции Start
A.RemoveRange(Start,N)	Удаление N объектов из массива A с позиции Start
A.Clear()	Удаление всех объектов из массива A

Динамические типизированные массивы

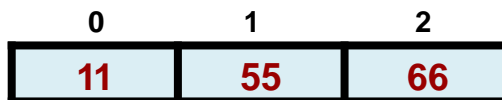
C#

```
List<int> A = new List<int>();  
int[] B = new int[] { 55, 66 };  
int[] C = new int[] { 33, 44 };
```

A.Add(11);



A.AddRange(B);



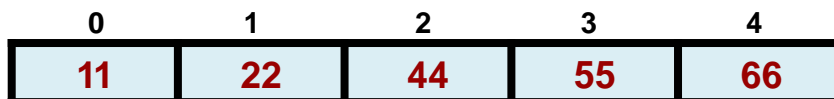
A.Insert(1, 22);



A.InsertRange(2, C);



A.Remove(33);



A.RemoveAt(1);



A.RemoveRange(1, 2);



Динамические не типизированные массивы

Класс ArrayList из пространства имен `System.Collections` позволяет работать с динамическими не типизированными массивами.

Отличие от типизированного динамического массива:

- ➔ возможность хранения объектов любого типа.
- ➔ не типизированный массив хранит **ссылки**, а не значения

Свойства класса ArrayList	
Count	число элементов, которое в действительности содержится в массиве ArrayList
Item	получает или задает объект с указанным индексом
Capacity	получает или задает число элементов, которое может содержать класс ArrayList
IsFixedSize	получает значение, показывающее, имеет ли список ArrayList фиксированный размер

Динамические не типизированные массивы

Методы класса ArrayList	
Add()	Добавить элемент в массив
Clear()	Удалить все элементы массива
Contains()	Проверяет нахождение объекта в массиве
Remove()	Удаляет элемент массива
RemoveAt()	Удаляет элемент массива по индексу
Sort()	Сортирует массив

Динамические не типизированные массивы

C#

```
ArrayList myAL = new ArrayList();  
  
myAL.Add("Hello");  
myAL.Add("World");  
myAL.Add("!");  
myAL.Add(1000);
```

Динамические не типизированные массивы

Особенности ArrayList:

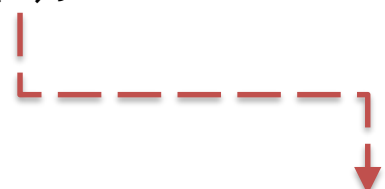
- ➔ Когда в список **ArrayList** добавляются элементы, его емкость автоматически увеличивается должным образом посредством перераспределения. Емкость может быть уменьшена посредством вызова метода [TrimToSize\(\)](#) или с помощью явного задания свойства [Capacity](#).
- ➔ Доступ к элементам осуществляется с помощью целочисленного индекса. Индексы в этой коллекции начинаются с нуля.
- ➔ Использование многомерных массивов в качестве элементов коллекции **ArrayList** не поддерживается.

Передача массивов в качестве аргументов

C#

```
static void Main(string[] args)
{
    int[] A = new int[5]{ 1, 3, 5, 7, 9 };
    PrintArray(A); //вызов метода PrintArray и передача в метод массива A
}

static void PrintArray(int[] arr)
{
    for (int i=0; i<arr.Length; i++)
    {
        Console.Write(arr[i]);
    }
}
```



Параметром метода является массив, а не переменная, т.к. тип данных указан как: `int[]`

Возврат массивов

C#

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

```
{
```

```
    int[] A;
```

```
    A = Fill();
```

```
}
```

```
static int[] Fill()
```

```
{
```

```
    int[] arr = new int[] { 10, 20 };
```

```
    return arr;
```

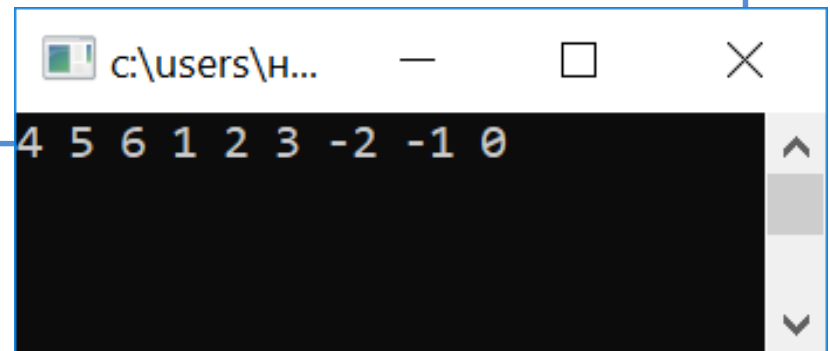
```
}
```

Цикл foreach

C#

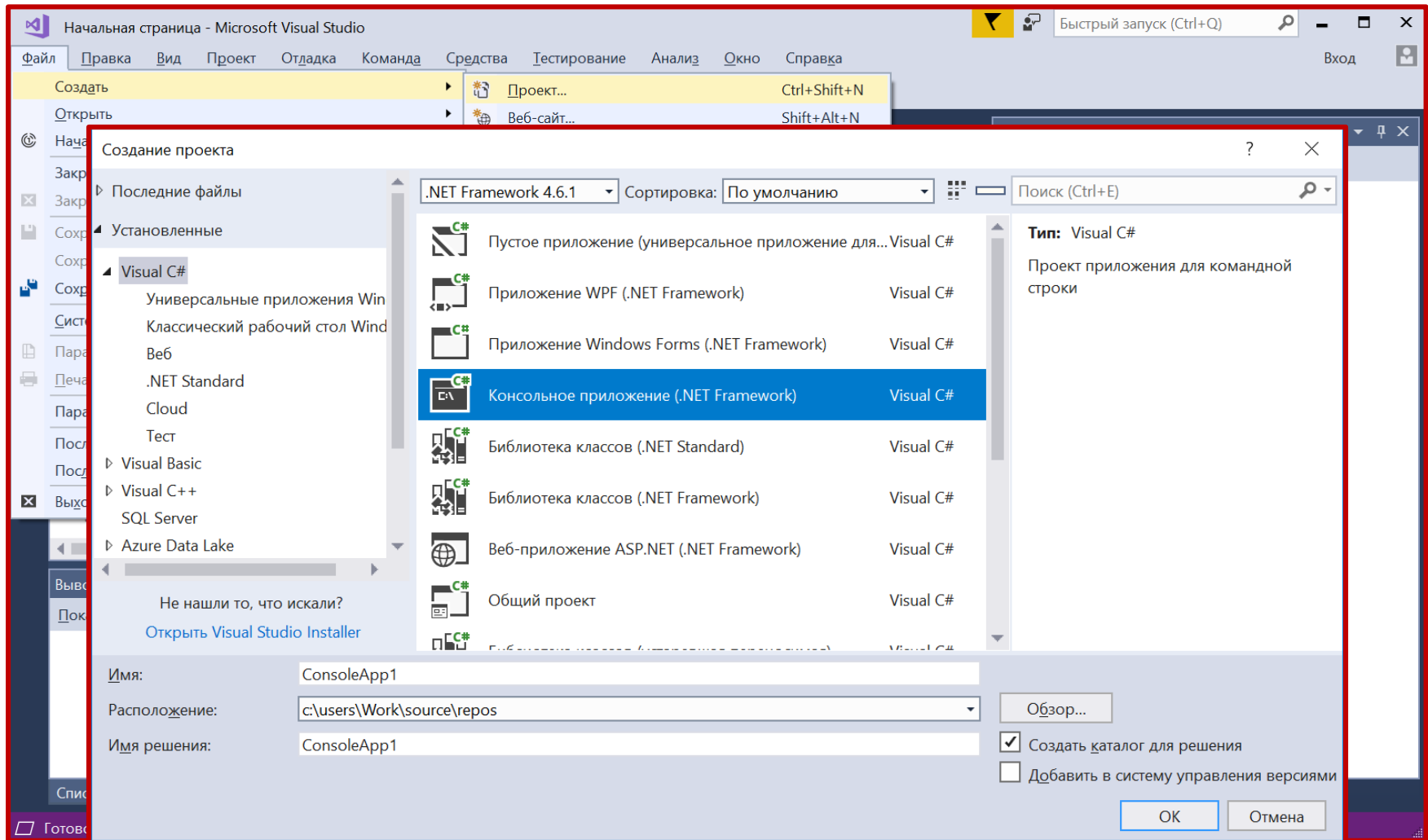
```
static void Main(string[] args)
{
    int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };

    foreach (int i in numbers)
    {
        Console.Write("{0} ", i);
    }
}
```



The screenshot shows a Windows console window with the title bar 'c:\users\н...'. The console output displays the numbers 4, 5, 6, 1, 2, 3, -2, -1, and 0, each followed by a space, as specified by the C# code. The text is rendered in a monospaced font with a light blue color on a black background.

Создание проекта в Visual Studio



Структура программы

C#

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

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            //код программы
        }
    }
}
```

Пространство имен

Директива **using** разрешает использование типов в пространстве имен

- ➔ Пространства имен представляют собой способ организации различных типов, присутствующих в программах C#.
- ➔ Программа C# содержит одно или несколько пространств имен, каждое из которых либо определено программистом, либо определено как часть написанной ранее библиотеки классов.

ПРИМЕР: пространство имен System содержит класс Console, который включает методы для чтения и записи в окне консоли (WriteLine и ReadLine).

Пространство имен

