Технология программирования

Язык программирования C#

(Си Шарп)

Платформа .NET

(dot NET

или

точка NET)

Немного о языке программирования C#

Немного истории

60-е годы

Combined Programming Language (CPL) — совместный язык программирования; Кристофер Стрейчи, Кембриджский и Лондонский университеты

Basic Combined Programming Language (BCPL) — основной совместный язык программирования; Мартин Ричардс, Кембриджский университет

B — Би; Кен Томпсон и Денис Ритчи, Bell Laboratories.

70-е годы

C — Си; Кен Томпсон и Денис Ритчи, Bell Laboratories.

80-е годы

C++ — Си плюс плюс; Бьёрн Страуструп, Bell Laboratories

2000-е годы

C# — Си шарп; Андерс Хейлсберг, Microsoft

Рассмотрим:

стандартные типы данных
значения и ссылки
управляющие структуры
параметры функций
массивы

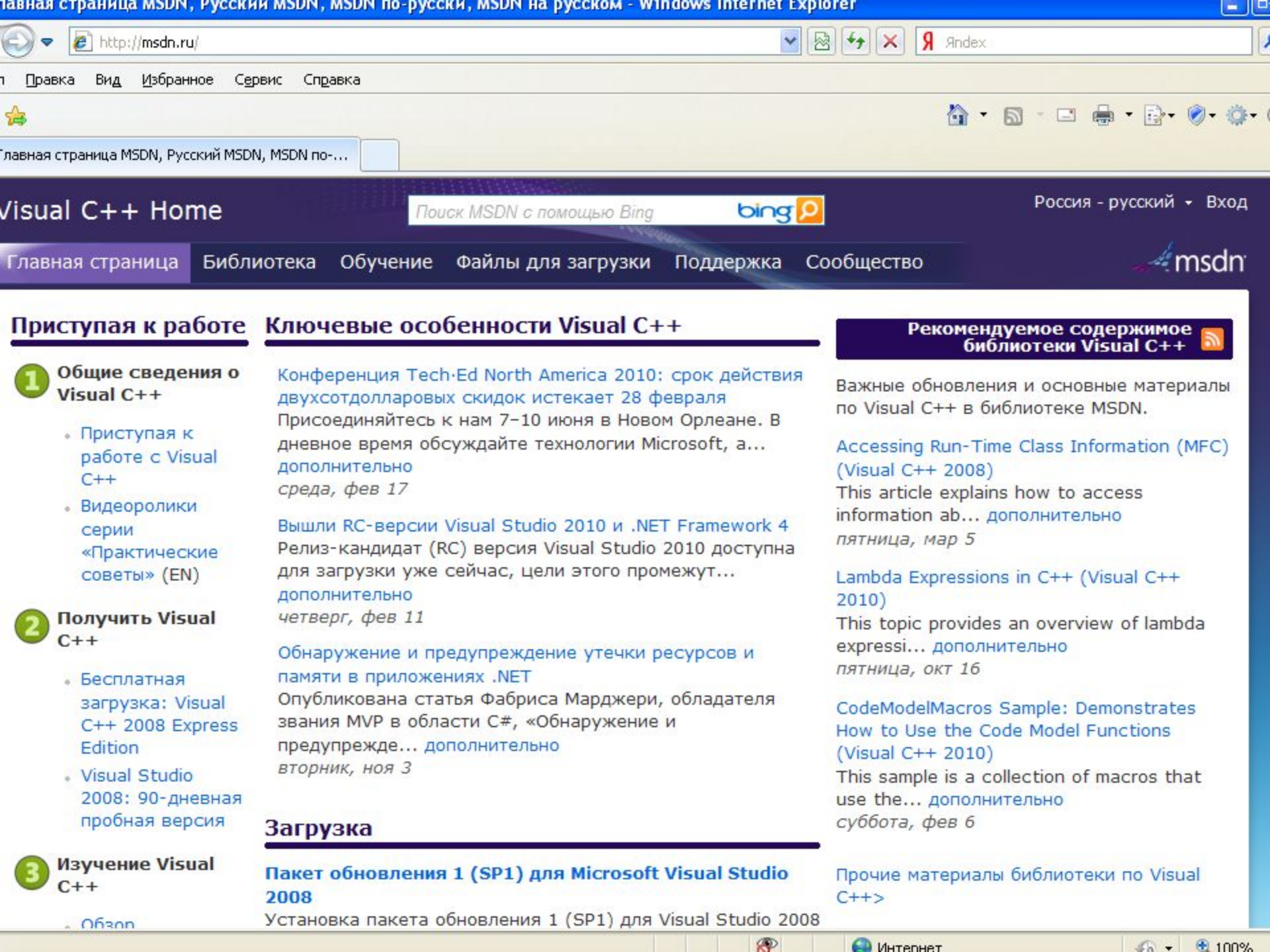
Литература

Эндрю Троелсен. Язык программирования C# 2008 и платформа .NET 4.0, 5-е издание

Интернет

Microsoft Developer Network

http://http://msdn.ru


Главная страница MSDN, Русский MSDN, MSDN по-русски, MSDN на русском - Windows Internet Explorer
http://msdn.ru/
Яndex
Правка Вид Избранное Сервис Справка
Главная страница MSDN, Русский MSDN, MSDN по-...
Visual C++ Home
Поиск MSDN с помощью Bing
bing
Россия - русский
Вход
Главная страница
Библиотека
Обучение
Файлы для загрузки
Поддержка
Сообщество
msdn
Приступая к работе
1 Общие сведения о Visual C++
Приступая к работе с Visual C++
Видеоролики серии «Практические советы» (EN)
2 Получить Visual C++
Бесплатная загрузка: Visual C++ 2008 Express Edition
Visual Studio 2008: 90-дневная пробная версия
3 Изучение Visual C++
Обзор
Ключевые особенности Visual C++
Конференция Tech·Ed North America 2010: срок действия двухсотдолларовых скидок истекает 28 февраля
Присоединяйтесь к нам 7–10 июня в Новом Орлеане. В дневное время обсуждайте технологии Microsoft, а... дополнительно
среда, фев 17
Вышли RC-версии Visual Studio 2010 и .NET Framework 4
Релиз-кандидат (RC) версия Visual Studio 2010 доступна для загрузки уже сейчас, цели этого промежут... дополнительно
четверг, фев 11
Обнаружение и предупреждение утечки ресурсов и памяти в приложениях .NET
Опубликована статья Фабриса Марджери, обладателя звания MVP в области C#, «Обнаружение и предупрежде... дополнительно
вторник, ноя 3
Загрузка
Пакет обновления 1 (SP1) для Microsoft Visual Studio 2008
Установка пакета обновления 1 (SP1) для Visual Studio 2008
Рекомендуемое содержимое библиотеки Visual C++
Важные обновления и основные материалы по Visual C++ в библиотеке MSDN.
Accessing Run-Time Class Information (MFC) (Visual C++ 2008)
This article explains how to access information ab... дополнительно
пятница, мар 5
Lambda Expressions in C++ (Visual C++ 2010)
This topic provides an overview of lambda expressi... дополнительно
пятница, окт 16
CodeModelMacros Sample: Demonstrates How to Use the Code Model Functions (Visual C++ 2010)
This sample is a collection of macros that use the... дополнительно
суббота, фев 6
Прочие материалы библиотеки по Visual C++>
Интернет
100%

Первая программа

```
using System;

class HelloClass
{
   public static void Main(string[] args)
   {
      Console.WriteLine("Hello, World!");
   }
}
```

варианты:

```
public static void Main()
{
}

public static int Main(string[] args)
{
   return 0;
}

public static int Main()
{
   return 0;
}
```

Обработка параметров командной строки

```
using System;

class HelloClass
{
     public static void Main(string[] args)
     {
        for (int i=0; i<args.Length; i++)
        {
           Console.WriteLine("Arg: {0}",
           args[i]);
        }
     }
}
```

или так:

```
foreach (string s in args)
{
   Console.WriteLine("Arg: {0}", s);
}
```

Стандартные типы данных

Обозна-чение	Диапазон	Размер	Назначение
sbyte	-128...127	1 байт	целое со знаком
byte	0…255	1 байт	целое без знака
short	-32768…32767	2 байта	целое со знаком
ushort	0...65565	2 байта	целое без знака
int	-2147483648… 2147483647	4 байта	целое со знаком
uint	0...4294967295	4 байта	целое без знака

Обозна-чение	Диапазон	Размер	Назначение
long	-9223372036854775808 …9223372036854775807	8 байт	целое со знаком
ulong	0… 18446744073709551615	8 байт	целое без знака
char	0000…FFFF	2 байта	символ Unicode
float	$\pm 1{,}5*10^{-45}$… $\pm\ 3{,}4*10^{38}$	4 байта	число с плавающей запятой
double	$\pm\ 5{,}0*10^{-324}$… $\pm\ 1{,}7*10^{308}$	8 байт	число с плавающей запятой

Обозна-чение	Диапазон	Размер	Назначение
bool	true, false	1 байт	булевский
string		ограничено памятью	набор символов Unicode

bool – логический тип

true – истина

false – ложь

Пример:

```
bool f1 = 5 > 1;   // f1 = true
bool f2 = 5 < 1;   // f2 = false
```

string – строковый тип

Пример:

```
string s1 = "Петр";
string s2 = "ГУ";
string s3 = s1 + s2;  // s3 = "ПетрГУ"
char a = s1[0];    // a = 'П'
```

Типы:

- значения
- ссылки

Типы, характеризуемые значениями, включают все числовые типы данных, символьный тип, булевский тип, перечни и структуры.

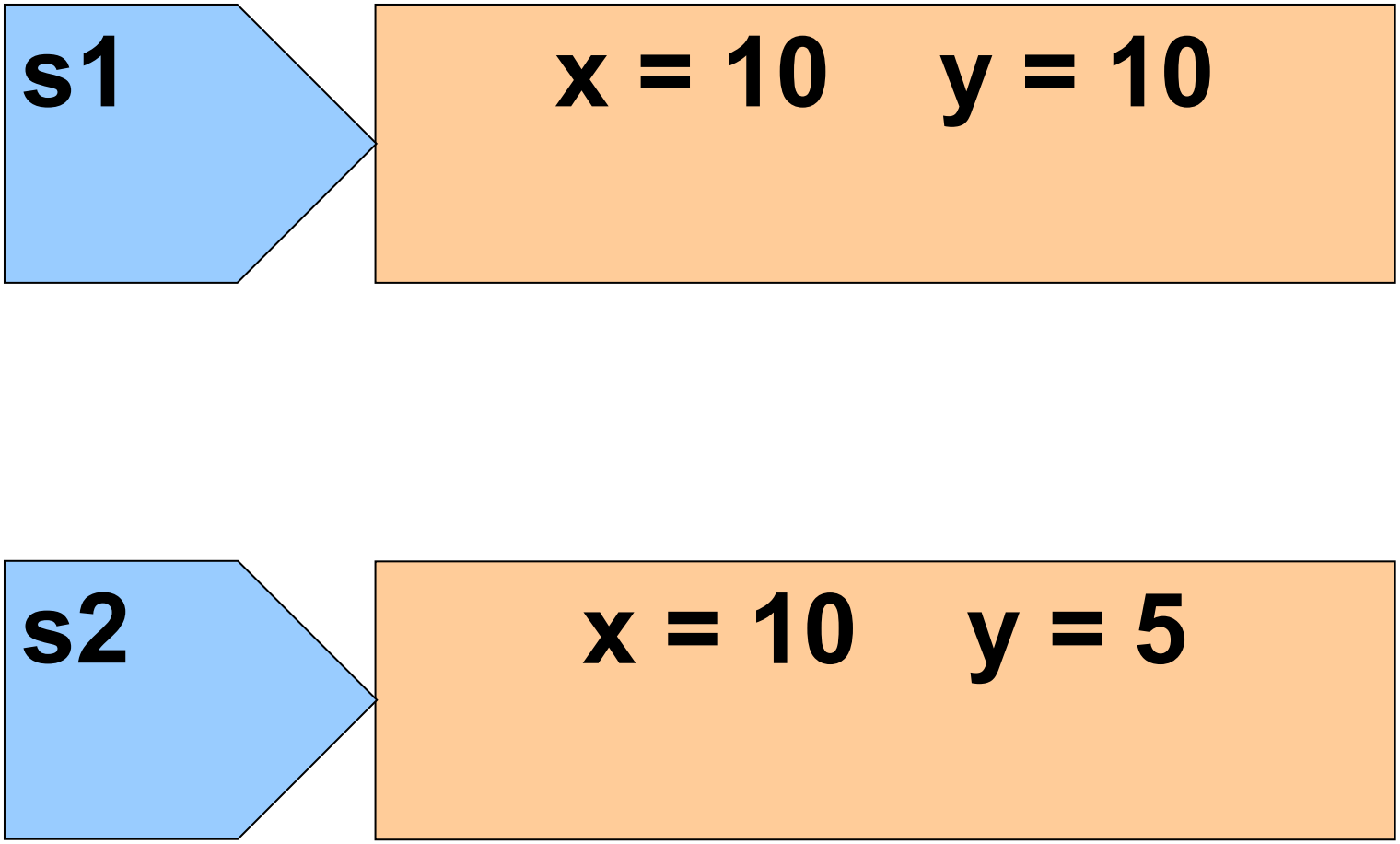
Все они размещаются в стеке. При присваивании происходит создание копии – «побайтовое» копирование.

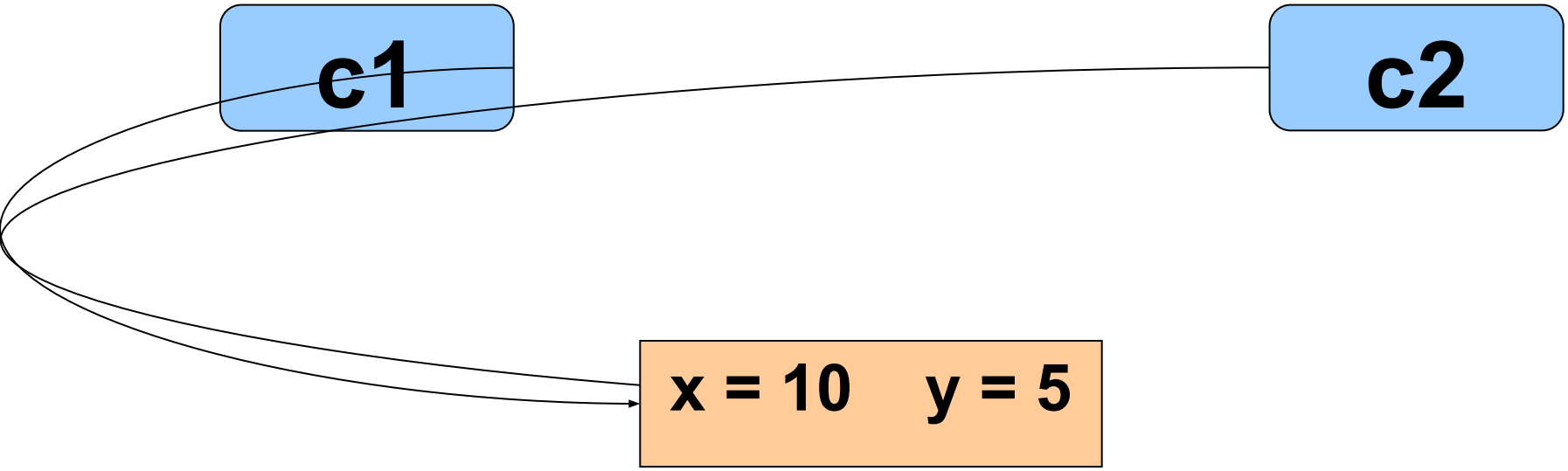
Большинство остальных типов являются ссылочными. Ссылочные типы размещаются в динамической памяти. При присваивании копируется ссылка (адрес) на объект. Создание объекта только при помощи оператора new.

Исключением являются строки. Для строк можно не использовать оператор new, при присваивании происходит создании копии.

```
struct MyStruct
{
   public int x, y;
}
static void Main()
{
   MyStruct s1, s2;

   s1.x = 10;
   s1.y = 10;

   s2 = s1;
   s2.y = 5;
   // s1 = {10, 10}, s2 = {10, 5}
}
```

```
class MyClass
{
   public int x, y;
}
static void Main()
{
   MyClass c1, c2;
   c1 = new MyClass();
   c1.x = 10;
   c1.y = 10;

   c2 = c1;
   c2.y = 5;
   // c1 = {10, 5}, c2 = {10, 5}
}
```


s1
x = 10 y = 10
s2
x = 10 y = 5


c1
c2
x = 10 y = 5

Преобразование строки в число

```
string s = "125";
int n = int.Parse(s);

if (int.TryParse(s, out n) == false)
{
   MessageBox.Show("Введите число");
   return;
}
```

Управляющие структуры

if
if … else
switch
for
while
do … while
foreach

Немного о массивах в языке программирования C#

Рассмотрим

объявление,

создание

и инициализацию массивов

```
// Объявление
int[] A;

// и создание
A = new int [5];
```

```
// Объявление и создание
int[] A = new int [5];
```

```
// Объявление, создание и инициализация
int[] A = new int[5] {1, 2, 3, 4, 5};

int[] A = new int[] {1, 2, 3, 4, 5};

int[] A = {1, 2, 3, 4, 5};
```

Одномерный массив

```
string t = Console.ReadLine();
int N = int.Parse(t);

string[] str = new string [N];

for (int i=0; i<N; i++)
   str[i] = Console.ReadLine();

foreach (string s in str)
   Console.WriteLine("{0}", s);
```

Двумерный массив

```
int[,] mas = new int [5,7];

for (int i=0; i<5; i++)
   for (int j=0; j<7; j++)
      mas[i,j] = i*j;

for (int i=0; i<5; i++)
{
   for (int j=0; j<7; j++)
      Console.Write("{0}\t", mas[i,j]);
   Console.WriteLine();
}
```

Ступенчатый двумерный массив

```
int[][] mas = new int [5][];

for (int i=0; i<5; i++)
{
   mas[i] = new int [i+1];
   for (int j=0; j<=i; j++)
      mas[i][j] = i*j;
}
for (int i=0; i<5; i++)
{
   for (int j=0; j<mas[i].Length; j++)
      Console.Write("{0}\t", mas[i][j]);
   Console.WriteLine();
}
```

Параметры функции

- параметры – значения
- ссылочные параметры
- выходные параметры

Передача по значению

```
int sum(int a, int b)
{
    int s = a + b;
    a = 500;
    b = 750;
    return s;
}
static void Main()
{
    int x = 15, y = 25;
    int z = sum(x ,y);
    // x = 15, y = 25
}
```

Передача по ссылке

```
void swap(ref int a, ref int b)
{
    int c = a;
    a = b;
    b = c;
}
static void Main()
{
    int x = 1, y = 5;
    swap(ref x ,ref y);
    // x = 5, y = 1
}
```

Передача по ссылке выходного параметра

```
void f(int a, int b, out int sum, out int pr)
{
    sum = a + b;
    pr = a * b;
}

static void Main()
{
    int x = 15, y = 25, s, p;
    f(x ,y, out s, out p);
    // s = 40, p = 375
}
```

Передача множества аргументов в виде одного параметра

```
int sum(params int[] mas)
{
    int s = 0;
    for (int i = 0; i < mas.Length; i++)
        s += mas[i];
    return s;
}

static void Main()
{
    int a = sum(5, 7, 3, 10);
    // a = 25
}
```

Немного о Microsoft Visual Studio.NET

Недостатки других подходов

Недостаток C – трудоемкий язык.

Недостаток C++ – трудоемкий язык, небезопасный.

Недостаток Visual Basic – меньше возможностей, чем у C++, нет наследования, нет параметризованных классов и т.д.

Недостаток Java – при создании приложения все пишется только на одном языке, мало средств межъязыкового взаимодействия (существует много готовых программ на разных языках, часть функций лучше реализовывать на других языках).

Преимущества модели .NET

Полное межъязыковое взаимодействие.

Общая среда для любых приложений.

Наличие библиотеки базовых классов.

Упрощение процесса развертывания приложения (например, разные версии DLL могут одновременно находится на одном компьютере).

Кроме того – поддержка работы с XML, в Web, работа с базами данных и прочее.

.NET – это среда выполнения
и библиотека базовых классов.

Блоки .NET

CLR – Common Language Runtime,
стандартная среда выполнения.

CTS – Common Type System,
стандартная система типов.

CLS – Common Language Specification,
общая языковая спецификация.

а так же:

MSIL, IL – Microsoft intermediate language,
промежуточный язык Microsoft.

JIT – just in time compilation,
компиляция в процессе выполнения.

IDE – Integrated Development Environment,
интегрированная среда разработки.

Управляемый код – это код (программа), предназначенный для работы в середе .NET.

Сборка – двоичный файл, содержащий управляемый код.

Сборка хранится в виде файла DLL или EXE. Сборка содержит код на языке IL и метаданные.

Преимущество IL – выполнение на любой платформе.

Во время выполнения программы при необходимости код с языка IL с помощью JIT компилируется в машинный язык и помещается в кэш-память.

Преимущества C#:

Нет необходимости в указателях.

Управление памятью производится автоматически.

Возможность работы с перечислениями,
структурами и свойствами классов.

Перегрузка операторов (как и в C++, но проще).

Полная поддержка использования
программных интерфейсов.

Наличие атрибутов.

Объектно-ориентированное программирование, классы

Принципы ООП:

- инкапсуляция
- наследование
- полиморфизм

Инкапсуляция – объединение данных и методов в независимые целостные объекты.

Наследование – создание производных классов на основе базовых классов.

Полиморфизм – изменение способа работы с объектами в зависимости от типа объекта.

Класс – пользовательский тип данных, содержащий члены класса (данные, называемые полями и функции, называемые методами). Могут быть и другие члены класса, например, события.

Объект – экземпляр (переменная) класса в памяти компьютера во время работы программы.

Для создания объектов класса используется оператор new.

Описание класса:

```
[<модификатор>] class <имя класса>
[: <базовый класс>]
{
   <тело класса>
}
```

Некоторые модификаторы класса:

abstract – класс является только базовым для других классов

sealed – класс не может быть базовым

Некоторые модификаторы класса:

public – класс доступен вне пределов сборки

internal - класс доступен только в пределах сборки (по умолчанию)

private - только ко вложенным классам, класс доступен в пределах класса, в котором объявлен

protected – только ко вложенным классам, класс доступен в пределах класса, в котором объявлен и из производных классов этого класса

abstract – класс является только базовым для других классов

sealed – класс не может быть базовым

static – статический класс, содержит только статические члены, запрет создания объектов

Некоторые члены класса:

константа

поле

метод

конструктор

свойство

индексатор

событие

тип (класс, структура, перечисление)

Члены класса:

константа

поле

метод

свойство

событие

индексатор

конструктор

деструктор

статический конструктор

тип

Для доступа к членам класса используется точка.

Примеры:

```
label1.Text

// Класс Студент и объекты класса:
Student st = new Student();
st.Name = “Иванов”;
st.Department = “Математический”;
st.SetPoint(“Алгебра”, 5);
double a = st.AveragePoint();
Stunent st1 = new Student();
```

Описание константы:

[<модификатор>] const <тип> <имя> [= <значение>]

Пример описания константы:

```
const double Pi = 3.1416;
```

Описание поля:

[<модификатор>] <тип> <имя> [= <значение>]

Некоторые модификаторы полей:

new – замещение поля

public – доступно вне класса

private – доступно только внутри класса

protected – доступно внутри класса и в порожденных классах

static – статическое поле (1 на всех)

readonly – только для чтения (инициализируется конструктором)

Описание метода:

[<модификатор>] <тип> <имя> (<список параметров>)

{

<тело метода>

}

Некоторые модификаторы методов:

public – доступен вне класса

private – доступен только внутри класса

protected – доступен внутри класса и в порожденных классах

static – статический метод (1 на всех)

virtual – виртуальный метод (может быть перегружен)

override – перегруженный метод

new – замещение метода

abstract – виртуальный метод без реализации

Конструктор – это метод, вызываемый при создании объекта. Имя конструктора совпадает с именем класса. Класс может содержать несколько конструкторов, отличающихся типом или количеством входных параметров.

```
class Student
{
    public string name;
    public int num;
    public Student()
    {   name = “Noname”; num = -1;
    }
    public Student(string s, int n)
    {   name = s; num = n;
    }
    public void print()
    {
        Console.WriteLine(“{0},{1}”,name,num);
    }
}
```

```
class Program
{
    static void Main()
    {
        Student s = new Student();
        s.print();
        s.name = "Иванов";
        s.num = 10;
        s.print();
        Student s1 = new Student("Петров", 15);
        s1.print();
    }
}
```

```
int GCD(int a, int b)
 {
      int c;
      while (b > 0)
      {
           c = a % b;
           a = b;
           b = c;
      }
      return a;
 }
```

```
int GCD(int a, int b)
{ // два параметра – a и b
      int c; // объявление переменной c
      while (b > 0) // пока b больше нуля
      {    // переменной c присвоить остаток
           c = a % b; //  от деления a на b
           a = b; // a присвоить b
           b = c; // b присвоить c
      }
      return a; // вернуть a
}
```

```
int GCD(int a, int b)
{ // вычисление наибольшего общего делителя
      int c; // чисел a и b алгоритмом Евклида
      while (b > 0)
      {
            c = a % b;
            a = b;
            b = c;
      }
      return a;
}
```

```
class Student
{
    private string name;
    private readonly int num;
    private static int N;

    public Student(string s)
    {
        name = s;
        N++;
        num = N;
    }

    public void print()
    {
        Console.WriteLine("{0},{1}",name,num);
    }
}
```

```
class Program
{
    static void Main()
    {
        Student s1 = new Student("Иванов");
        s1.print();
        Student s2 = new Student("Петров");
        s2.print();
    }
}
```

Доступ к статическим членам осуществляется через имя класса, к нестатическим членам – через имя объекта.

```
class Table
{
    static public int N = 50;
    public int M = 50;
}
class Program
{
    public static void Main()
    {
        int a = Table.N;
        Table t = new Table();
        int b = t.M;
    }
}
```

Конструктор с модификатором static используется для инициализации статических членов класса.

В этом случае нельзя указывать модификаторы доступа (public, private, protected)

```
class Table
{
    static public int N;
    static Table()
    {
        N = 100;
    }
}
```

Конструктор с модификатором private используется в том случае, когда нужно запретить создание объектов класса.

```
class Table
{
    static public int N = 100;
    private Table() { }
}
```

Наследование. Полиморфизм. Перегрузка методов базового класса.

```
abstract class Component
{
    protected double h;
    public void set_h(double x)
    {
        if (x > 0 && x < 100)
            h = x;
        else
            h = 0;
    }
    public abstract double square();
    public double volume()
    {
        return square() * h;
    }
}
```

```
class Cylinder : Component
{
    private double r;
    public void set_r(double y)
    {
        if (y > 0 && y < 200)
            r = y;
        else
            r = 0;
    }
    public override double square()
    {
        return 3.14 * r * r;
    }
}
```

```
class Prism : Component
{
    private double a;
    public void set_a(double z)
    {
        if (z > 0 && z < 50)
            a = z;
        else
            a = 0;
    }
    public override double square()
    {
        return a * a;
    }
}
```

```
class Program
{
    static void Main()
    {
        Cylinder cyl = new Cylinder();
        cyl.set_r(5.1);
        cyl.set_h(3.5);
        Prism pr = new Prism();
        pr.set_a(27);
        pr.set_h(94);
        double V1 = cyl.volume();
        double V2 = pr.volume();
    }
}
```

Интерфейсы

Интерфейс – это ссылочный тип, в состав которого входят только абстрактные члены.

Члены интерфейса:

- методы;
- события;
- свойства;
- индексаторы.

Других членов у интерфейса быть не может.

Интерфейс содержит только объявления членов, их реализация должна находится в классе, реализующем интерфейс.

Объявление интерфейса:

```
[<модификатор>] interface <идентификатор>
[: <базовый интерфейс>]
{<тело интерфейса>}
```

Модификаторы интерфейса: `new, public, protected, internal, private`.

Члены интерфейса описываются без модификаторов.

Если класс реализует некоторый интерфейс, то имя интерфейса указывается после имени класса и двоеточия:

```
class <имя класса> : [<базовый класс>,]
<интерфейс>
```

Класс, реализующий интерфейс, обязательно должен реализовывать все члены этого интерфейса.

Класс может реализовывать несколько интерфейсов:

```
class <имя класса> : [<базовый класс>,]
<интерфейс1>, <интерфейс2>
```

Тем самым имитируется множественное наследование.

Примечание:

Наследование класса от нескольких базовых классов в языке C# запрещено.

Интерфейсы сходны с абстрактными классами и отличаются от них тем, что поддерживают:
множественное наследование;

содержат ограниченный набор членов;

не содержат реализаций.

Интерфейс может иметь несколько базовых интерфейсов. В объявлении интерфейса базовые интерфейсы отделяются запятыми.

Пример:

```
public interface MyInterface
{
    void Method1();
    void Method2();
}
class MyClass : MyInterface
{
    public void Method1()
    {
        // Реализация метода
    }
    public void Method2()
    {
        // Реализация метода
    }
}
```

Под явной реализацией члена интерфейса понимается объявление с использованием полного имени члена интерфейса:

```
<возвращаемое значение>
<интерфейс>.<метод>(аргументы)
```

В этом случае для членов не используются модификаторы доступа.

Явная реализация необходима в случае, если класс реализует несколько интерфейсов, содержащих одноименные методы.

Для вызова такого метода необходимо использовать интерфейсную ссылку.

Оператор `as` можно использовать для получения интерфейсной ссылки.

Оператор `is` можно использовать для проверки того, реализует ли класс данный интерфейс.

В производных классах можно использовать интерфейс, реализованный в базовом классе.

Абстрактные классы могут реализовывать интерфейсы. При этом члены интерфейса должны быть либо реализованы, либо объявлены как абстрактные.

```
public interface Description
{
    void Name();
}

class Student : Description
{
    string str;
    public Student(string s)
    {
        str = s;
    }
    public void Name()
    {
        Console.WriteLine("Имя студента " + str);
    }
}
```

```
class Car : Description
{
    string str;
    public Car(string s)
    {
        str = s;
    }
    void Description.Name()
    {
        Console.WriteLine("Марка машины " + str);
    }
}
class Program
{
    static void Print(Description d)
    {
        d.Name();
    }
```

```
    static void Main()
    {
        Student st = new Student("Иванов");
        Print (st);
        Car car = new Car("Тойота");
        if (car is Description)
        {
            // получение интерфейсной ссылки
            Description d = (Description)car;
            Print (d);
        }
        // либо
     // получение интерфейсной ссылки
        Description g = car as Description;
        if (g != null)
            Print (g);
    }
}
```

Свойства

Свойства внешне выглядят как поля, используются для чтения и/или записи, однако фактически при обращении к ним происходит вызов методов доступа.

При записи вызывается метод set.

При чтении вызывается метод get.

```
class Book
{  private int Npage = 0;
   public int NPage
   {
      get
      {
         return Npage;
      }
      set
      {
         if (value >= 0 && value <= 2000)
            Npage = value;
         else
            Npage = 0;
}  }  }
```

```
class Program
{
   static void Main()
   {
      Book b = new Book();
      b.NPage = 1000;
      Console.WriteLine(b.NPage.ToString());
      b.NPage = 3000;
      Console.WriteLine(b.NPage.ToString());
   }
}
```

Индексаторы

Индексаторы позволяют работать с элементами классов как с массивами, то есть использовать квадратные скобки [ ]

```
class Book
{
   private string[] title = { "Война и мир",
   "Отцы и дети", "Дубровский" };
   private int[] price = { 500, 270, 80 };

     public int this[string str]
     {
         get
         {
             for (int i=0; i<title.Length; i++)
                 if (str == title[i])
                     return price[i];
             return -1;
         }
```

```
        set
        {
            for (int i=0; i<title.Length; i++)
                if (str == title[i])
                    price[i] = value;
        }
    }
}
class Program
{
    static void Main(string[] args)
    {
        Book MyBook = new Book();
        MyBook["Война и мир"] = 525;
        int p = MyBook["Отцы и дети"];
    }
}
```

Делегаты

Делегат – это тип данных, содержащий информацию о типе возвращаемого значения и сигнатуре (списке формальных параметров) метода.

Делегаты используются для вызова одного или нескольких методов, соответствующих его типу.

Объявление делегата

[<модификаторы>] delegate <тип> <имя> (<список параметров>)

Модификаторы:

- new
- public
- protected
- private

Пример:

```
public delegate void StDel (Student s);
```

Инициализация делегата

Под инициализацией делегата понимается создание экземпляра делегата и установление привязки к методу.

Тип возвращаемого значения, количество и типы параметров метода должны быть такими же, как у делегата.

Инициализация делегата статическим методом

```
public delegate void StDel (Student s);

class Group
{
    public static void PrintName(Student s)
    {
        Console.WriteLine(s.Name);
    }
}

class App
{
    static void Main()
    {
        StDel d = new StDel(Group.PrintName);
    }
}
```

Инициализация делегата методом экземпляра

```
public delegate void StDel (Student s);

class Group
{
    public void PrintName(Student s)
    {
        Console.WriteLine(s.Name);
    }
}

class App
{
    static void Main()
    {
     Group g = new Group();
        StDel d = new StDel(g.PrintName);
    }
}
```

Вызов делегата

Делегат вызывается с помощью указания его имени, за которым в скобках записаны передаваемые параметры.

Пример:

```
* * * * * * * * * * * * * * *
Student st = new Student();
d(st);
```

```
class Student
{
   public string Name;
   public int Number;
   public Student(string Name, int Number)
   {
      this.Name=Name;
      this.Number=Number;
   }
   public delegate void StDel (Student s);
}
```

```
class Group
{
   private ArrayList Gr = new ArrayList();
   public Group()
   {
      Gr.Add(new Student("Иванов",1));
      Gr.Add(new Student("Петров",2));
      Gr.Add(new Student("Сидоров",3));
   }
   public void f(Student.StDel sdf)
   {
      foreach(Student s in Gr)
         sdf(s);
   }
}
```

```
class App
{
   public static void PrintName(Student s)
   {
      Console.WriteLine(s.Name);
   }
   public static void PrintNumber(Student s)
   {
      Console.WriteLine(s.Number);
   }
   static void Main()
   {
      Group gr = new Group();
      gr.f(new Student.StDel(PrintName));
      gr.f(new Student.StDel(PrintNumber));
      Student.StDel a = new
Student.StDel(PrintName);
      a += new Student.StDel(PrintNumber);
      gr.f(a);
   }
}
```

Делегаты позволяют упростить процесс вызова методов классов из других классов.

Делегаты используются для реализации «обратного вызова»

Обработка событий

События – механизм, с помощью которого класс может посылать уведомления об определенных событиях различным приложениям.

Обработка щелчка мышкой по кнопке

обработчик события:

```
private void button1_Click(object sender, EventArgs e)
{
   label1.Text = “Привет!”;
}
```

привязка метода к событию:

```
this.button1.Click += new
System.EventHandler(this.button1_Click);
```

Создание и использование событий:

1. Объявление делегата:

```
public delegate void del(int[] mas);
```

2. Создание события как экземпляра делегата:

```
public event del use;
```

3. Вызов события:

```
use(P);
```

4. Создание метода:

```
public void plus(int[] A) { *** }
```

5. Привязка метода в качестве обработчика события:

```
Name.use += new Name.del(plus);
// Name – имя объекта или класса
```

```
class Car
{
   private bool good=true;
   public delegate void CarHandler(string msg);
   public static event CarHandler Crack;
   public static event CarHandler OOPS;
   int Speed;
   int maxSpeed = 150;
```

```
   public void SpeedUp(int delta)
   {
      if (!good)
      {
         if (Crack!=null)
            Crack("Машина вышла из строя!!!");
      }
      else
      {
         Speed+=delta;
         if (maxSpeed - Speed <= 10)
            if (OOPS!=null)
               OOPS("Осторожно!!!");

         if (Speed>=maxSpeed)
            good=false;
         else
            Console.WriteLine("Текущая
            скорость: "+Speed);
      }
   }
}
```

```
class CarEvent
{
   public void OnCrack(string s)
   {
      Console.WriteLine("Пришло сообщение:
      {0}", s);
   }
   public void OnCrack2(string s)
   {
      Console.WriteLine("Повторяю: {0}", s);
   }
   public void OnOOPS(string s)
   {
      Console.WriteLine("Пришло сообщение:
{0}", s);
   }
}
```

```
class App
{
   static void Main()
   {
      Car c = new Car();
      CarEvent e = new CarEvent();
      Car.Crack += new
      Car.CarHandler(e.OnCrack);
      Car.Crack += new Car.CarHandler(e.OnCrack2);
      Car.OOPS += new Car.CarHandler(e.OnOOPS);
      for (int i=0; i<10; i++)
         c.SpeedUp(20);
      Car.Crack -= new
      Car.CarHandler(e.OnCrack);
      Car.Crack -= new Car.CarHandler(e.OnCrack2);
      Car.OOPS -= new Car.CarHandler(e.OnOOPS);
   }
}
```

```
class Trans
{
    private static int[] P;

    public delegate void del(int[] mas);
    public static event del use;

    // Переворачивание массива
    private static void Reverse(int k)
    {
        int j = 1;
        while (j < k)
        {
            int t = P[j];
            P[j] = P[k];
            P[k] = t;
            j++;
            k--;
        }
    }
```

```
// Рекурсивная реализация перебора
   private static void Antilex(int m)
   {
       if (m == 1)
       {
           if (use != null)
               use(P);
           return;
       }
       for (int i = 1; i <= m; i++)
       {
           Antilex(m - 1);
           if (i < m)
           {
               int t = P[i];
               P[i] = P[m];
               P[m] = t;
               Reverse(m - 1);
           }
       }
   }
```

```
// Запуск рекурсии
    public static void DoIt(int n)
    {
        P = new int[1 + n];
        for (int i = 1; i <= n; i++)
            P[i] = i;
        Antilex(n);
    }
}
```

```
class Program
{
    static void print(int[] A)
    {
        for (int i = 1; i < A.Length; i++)
            Console.Write("{0} ", A[i]);
        Console.WriteLine();
    }

    static int count = 0;

    static void plus(int[] A)
    {
        count++;
    }
```

```
    static void Main()
    {
        Trans.use += new Trans.del(print);
        Trans.use += new Trans.del(plus);

        Trans.DoIt(5);
        Console.WriteLine(count);
        Console.ReadKey();

     Trans.use -= new Trans.del(print);
        Trans.use -= new Trans.del(plus);
    }
}
```

Обработка исключений

```
string str = Console.ReadLine();
int a;
try
{
   a = int.Parse(str);
}
catch (Exception e)
{
   Console.WriteLine(e.Message);
   a = 0;
}
```

Пример записи в текстовый файл и чтения из текстового файла

```
using System.IO;
class Program
{
    static void Main(string[] args)
    {
        StreamWriter writer =
        File.CreateText("output.txt");
        writer.WriteLine("Привет!");
        writer.Close();
        StreamReader reader = File.OpenText("output.txt");
        string str = reader.ReadLine();
        string[] s = File.ReadAllLines("input.txt");
        int m = int.Parse(s[0]);
        int n = int.Parse(s[1]);
     }
}
```

```
int[,] A = new int[m, n];
        for (int i = 2; i < m + 2; i++)
        {
            int j = 0;
            string num = "";
            for (int k = 0; k <= s[i].Length && j < n; k++)
            {
                if (k == s[i].Length || s[i][k] == ' `
                    || s[i][k] == '\t')
                {
                    A[i - 2, j] = int.Parse(num);
                    num = "";
                    j++;
                }
                else
                    num += s[i][k];
            }
        }
    }
}
```

Спроектируйте класс, содержащий информацию о студенте. Класс должен содержать закрытые поля: номер зачетной книжки (целое число) и фамилию студента (строка). Значения полей должны устанавливаться конструктором класса. Для доступа к полям необходимо использовать свойства, причем номер зачетной книжки изменять запрещено. Класс должен содержать отрытый метод, печатающий на консоль сообщение «Меня зовут » и далее – фамилия студента.

Вопросы:

1. Типы данных (значения, ссылки).
2. Виды параметров функции.
3. Понятие класса. Члены класса.
4. Инкапсуляция, наследование, полиморфизм.
5. Конструктор класса.
6. Статические члены класса.
7. Свойства. Индексаторы.
8. Делегаты.
9. События.
0. Интерфейсы.
1. Обработка исключений.