# Съвременни софтуерни технологии

**Лекции 2 ч.**
**Лабораторни упражнения 2 ч.**
**Изпит**
**Лектор:**

**Проф. д-р Мариана Горанова**
**Катедра „Програмиране и компютърни технологии“, ФКСУ**
**Технически университет – София**
**Кабинет: 2304**
**Електронен адрес:** mgor@tu-sofia.bg
**Интернет адрес:** http://pct.tu-sofia.bg/Moodle001/
Username: student
Password: pktt

## Литература

**I. Основна литература**

1. Jeffrey Richter, CLR via C#, Microsoft Press, 2010.
2. Tom Archer, Andrew Whitechapel, Inside C#, Second Edition, Microsoft Press, 2002.
3. John Sharp, Jon Jagger, Microsoft Visual C# .NET Step by Step, Microsoft Press, 2002.
4. Jesse Liberty, Programming C#, Second Edition, O'Reilly, 2002.
5. Damien Watkins, Mark Hammond, Brad Abrams, Programming in the .NET Environment, Microsoft Corporation, 2003.
6. Чарлз Петцолд, Windows на C#, том I и том II, СофтПрес, ООД, 2003.
7. Светлин Наков и колектив, Програмиране за .NET Framework, Българска асоциация на разработчиците на софтуер, Фабер, 2004.
8. Donis Marshal, Programming Microsoft Visual C# 2005: The Language, Microsoft Press, 2006.
9. Джефри Рихтер, Microsoft .NET Framework – приложно програмиране, СофтПрес ООД, 2002.

**II. Допълнителна литература**

1. Judith Bishop, Nigel Horspool, C# Concisely, Pearson Education Limited, 2004.

**III. Учебник и Ръководство за лабораторни упражнения**

1. Goranova M., V. Dimitrova, Advanced Software Technologies (C#), Technical University Publishing Complex, Sofia, 2009.
2. М. Горанова, В. Димитрова, Д. Гоцева, Ръководство по програмиране на C#, ТУ – София, 2006.

## Основи на платформата .NET

**Платформа .NET – стратегия на Microsoft за разработка на големи разпределени софтуерни системи.**

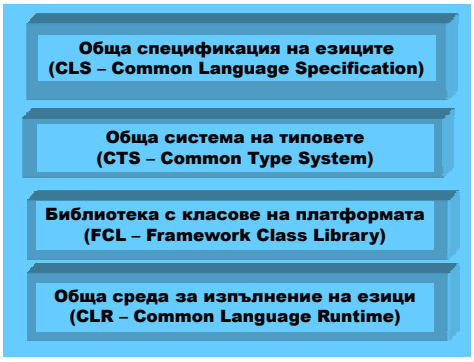
**.NET Framework – компонентен модел за Интернет – отделни софтуерни компоненти, написани на различни езици, се комбинират във функционираща система.**

**Различия на .NET Framework от:**

1. **COM (Component Object Model) на Microsoft – компонентен модел за десктоп (не за разпределени системи).**
2. **CORBA (Common Object Request Broker Architecture) – програмен модел за Интернет с обектно-ориентирана архитектура за разпределени системи – няма компонентна архитектура. CORBA3 е с компонентна архитектура.**

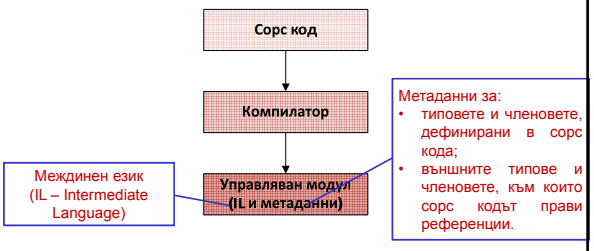
**Сравнява се с Java – програмен език за Интернет с характеристики на COM, CORBA и .NET, но за единствен програмен език.**

## Архитектура на .NET Framework



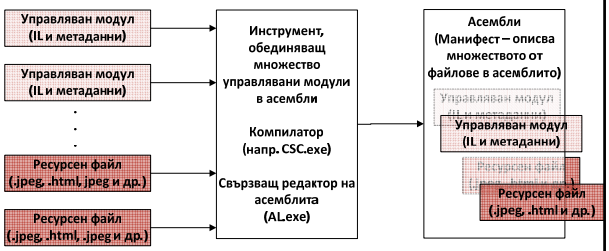
## Архитектура на .NET Framework


## CLR – Основният компонент на .NET Framework



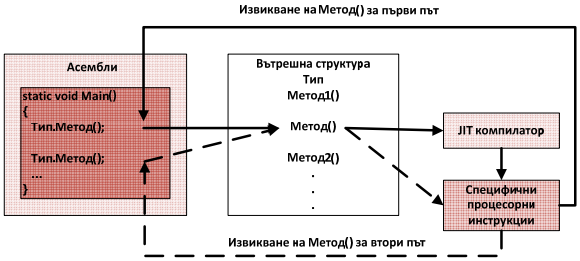

**Компилиране на сорс код в управляван модул**

## CLR




**Обединяване на управлявани модули в асемблии**

## CLR




**Изпълнение на асемблии**

## Предимства на .NET Framework

1. **Едни и същи концепции и услуги за всички типове приложения.**
2. **Многократно използване на компонентите.**
3. **Поддръжка на много програмни езици.**

## Езици в .NET Framework

1. **Visual Basic .NET**
2. **Visual C++ .NET**
3. **C#**
4. **Python за .NET**
5. **Perl за .NET**
6. **Компонентен Pascal за CLR**
7. **HotDog Scheme**
8. **Mondrian**
9. **Active Oberon за .NET**
10. **J#**
11. **F#**

# Въведение в езика за програмиране C#

**Anders Hejlsberg**
**(Delphi, Java Foundation classes)**
**Scott Wiltamuth (Microsoft)**
**Peter Golde (Microsoft)**

**C# – стандарт**
**ECMA (European Computer Manufacturer's Association)**
**ISO 2003**

## Цели

1. **Развит за .NET Framework**
2. **Удобен**
3. **Съответства на .NET CLR (Common Language Runtime) – обща среда за изпълнение на различни езици.**
4. **Опростява модела на C++**
   **а) липсва заглавен файл и препроцесор;**
   **б) липсва управление на паметта чрез система от указатели и събиране на боклука – използва система от референтни типове.**
5. **Гъвкав**
6. **Поддържа разработка на компоненти**

## Характеристики

**Много близък до Java**

**70% Java, 10% C++, 5% Visual Basic, 15% ново**

| Както в Java: | Както в C++: |
|---|---|
| Обектно-ориентиран (единично наследяване) | Презареждане на оператори |
| Интерфейси | Аритметика с указатели в несигурен код |
| Изключения | Някои детайли от синтаксиса |
| Нишки | |
| Наименовани пространства (като пакетите) | |
| Силно типизиран | |
| Събиране на боклука | |
| Динамично зареждане на кода | |

**Нови характеристики: (спрямо Java)**

| | |
|---|---|
| Референтни и изходни параметри | Компонентно-базирано програмиране<br>– свойства<br>– събития |
| Обекти, разположени в стека (struct) | Делегати |
| Правоъгълни масиви | Индексатори |
| Изброим тип (enum) | Презареждане на оператори |
| Унифицирана система на типовете | Оператор foreach* |
| Оператор goto | Опаковане/разопаковане |
| Създаване на версия | Атрибути |

## Структура на програма на C#

```
using <име_на_пространство>;
namespace <име_на_потребителско_пространство>
{
   class <име_на_потребителски_клас>
   {
        static void Main (string[] args)
        {
             // тяло
        }
   }
}
```

namespace – **пространство от имена**

**Пълно име на клас:**
<име_на_пространство>.<име_на_клас>

using – **директива – път за търсене на** <име_на_пространство>
**(не се прилага за** <име_на_клас>**)**

**Пример:**

```
using System;
class Welcome
{
   static void Main()
   {
        Console.WriteLine ("Добре дошли!");
        Console.WriteLine ("Днес е " + DateTime.Now);
   }
}
```

System – **пространство от имена**

**Клас** System.Console – **представя стандартните потоци за конзолни приложения.**

**Метод** Console.WriteLine – **извежда данните в стандартния изходен поток и добавя край на ред.**

**Форматиране на изхода:**

Console.WriteLine("{N[,M][:S]}", аргумент0, …, аргументN);

N – **номер на позицията на аргумента в списъка от стойности (начална позиция 0);**

M – **(незадължителна) ширина и подравняване с добавени празни позиции:**

- M<0 **или липсва – подравняване от ляво;**
- M>0 **– подравняване от дясно;**

S – **(незадължителен) форматиращ низ – при липса съответният метод** ToString **определя форматирането:**

Xm X – **форматен спецификатор;** m – **точност;**

- C/c **парична единица**
- D/d **десетично цяло число**
- E/e **експоненциално реално число**
- F/f **реално число с фиксирана точка**
- G/g **основно представяне**
- N/n **числено представяне подобно на** F **с разделител за хилядите**
- P/p **процент**
- R/r **закръглено (само за плаваща точка) – гарантира коректно преобразуване**
- X/x **шестнадесетично**

**Потребителски форматни спецификатори:**

| | |
|---|---|
| 0 | **допълва с незначещи нули** |
| # | **изобразява само значещи цифри** |
| . | **място на десетичния разделител** |
| , | **разделител за хилядите** |
| % | **изобразява %** |
| E+0 E-0 e+0 e-0 | **експоненциално представяне** |
| \ | **за форматиращи последователности (нов ред** \n**)** |
| 'ABC' "ABC" | **изобразява низ в литерали или кавички** |
| ; | **разделител на секция – различно форматиране в зависимост от това дали стойността е >0, <0 или 0** |
| аргументN | **израз; ако е** null**, се използва празен низ.** |

**Структура** DateTime **– представя време (дата и час).**

**Свойства**

| | |
|---|---|
| Now | **(статично) връща текущата дата и време;** |
| Date | **връща датата;** |
| TimeOfDay | **връща времето;** |
| Today | **(статично) връща текущата дата.** |

**Форматиращи символи:**

| | | |
|---|---|---|
| D | LongDatePattern | dddd, mmmm dd, yyyy |
| d | ShortDatePattern | mm/dd/yyyy |
| T | LongTimePattern | hh:mm:ss |
| t | ShortTimePattern | HH:mm |
| m, M | MonthDayPattern | mmmm dd |

**Пример:**

```
DateTime dt = DateTime.Now;
Console.WriteLine (dt);
Console.WriteLine ("Дата={0:d}, време={1:T}, днес е {2:m}",
                   dt.Date, dt.TimeOfDay, DateTime.Today);
```

**Резултати:**

```
27.2.2014 г. 9:32:46
Дата=27.2.2014 г., време=9:32:46.1234567, днес е 27 Февруари
```

**Пример:**

```
using System;
class TestWriteLine
{
   static void Main(string[] args)
   {
         Console.WriteLine("{0,5},{1:D5}", 123, 456);
         // □□123,00456
         Console.WriteLine("{0,-10:D6},{1,-10:D6}", 123, 456);
         // 000123□□□□,000456□□□□
         Console.WriteLine("{0,-10}{1,-8}", "Име", "Фак.N");
         // Име□□□□□□□Фак.N□□□
         Console.WriteLine("------------------");
         // - - - - - - - - - - - - - - - - - -
         Console.WriteLine("{0,-10}{1,8}", "Иван", 123456);
         // Иван□□□□□□□□123456
         Console.WriteLine("{0,-10}{1,8}", "Гергана", 7890);
         // Гергана□□□□□□□7890
         Console.WriteLine("{0:C}{1,5:F2}", 7890, 5.6);
         // 7□890,00□лв□5,60
```

```
            float f = -123456.7890F;
            Console.WriteLine("{0:$#,##0.00;($#,##0.00);Zero}", f);
            // ($123□456,80)
            int i = 1234567890;
            Console.WriteLine("{0:(###) ### - ####}", i);
            // (123) □456□-□7890
            Console.WriteLine("{0:#%}", i);
            // 123456789000%
        }
    }
```

## Създаване на конзолно приложение във Visual Studio .NET

1. **Стартиране на Visual Studio .NET.**
2. **File ⇒ New ⇒ Project**
   **Project Type ⇒ Visual C# Project**
   **Templates ⇒ Console Application**
   **Location ⇒ директория на проекта**
   **Name ⇒ име на проекта**
3. **View ⇒ Solution Explorer**
   име_на_приложение.sln – **файл на решението (един за приложение с няколко проекта);**
   име_на_проект.csproj – **C# проектен файл (няколко сорс файла на един и същ програмен език);**
   име_на_клас.cs – **C# сорс файл;**
   AssemblyInfo.cs – **C# сорс файл за добавяне на атрибути към програмата;**
   App.ico – **икона на приложението.**
4. **Въвеждане кода на програмата**
5. **Изграждане и изпълнение на конзолно приложение**
   **Build ⇒ Build**
   **Debug ⇒ Start Without Debugging**

## Създаване на конзолно приложение от команден режим

**C# компилатор**

csc.exe

**Установяване на пътя до изпълнимия файл** csc.exe

VCVARS32.bat

**(напр.** C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\vcvars32.bat**)**

**1. Компилация**

csc приложение.cs

**2. Изпълнение**

приложение

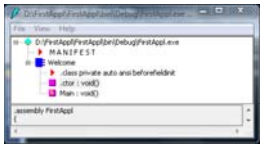
**Асембли – фундаментална част в .NET;**
- **създава се от компилатора;**
- **съдържа код, изпълняващ се от системата;**
- **колекция от модули, експортирани типове и ресурси, която има име и версия (от гледна точка на клиента);**
- **начин за пакетиране на модули, типове и ресурси, използвани от клиента (от гледна точка на създателя).**

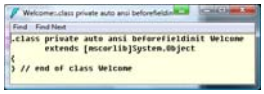
**Манифест – съдържа информация за мета данните; включва три записа:**
- .assembly **за обръщение към външно асембли;**
- .assembly **с информация за асемблито;**
- .module**, съдържащ името на физическия файл и друга външна информация.**

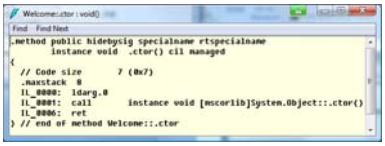
**Деасемблиране на изпълним файл – приложение** ILDASM

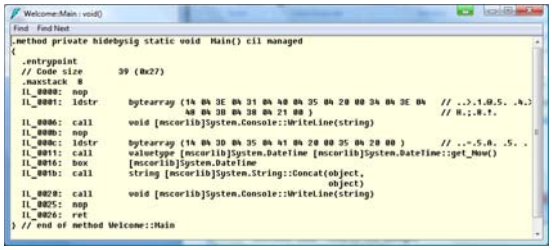
**Start ⇒ Programs ⇒**
**Microsoft Visual Studio 2012 ⇒**
**Visual Studio Tools ⇒**
**Development Command Prompt for VS2012 ⇒**
**команден прозорец :** ildasm

**прозорец на ILDASM :**
**File ⇒ Open ⇒**
драйв:\...\директория_на_приложение\bin\Debug\приложение.exe









## Основни входно-изходни операции

1. **Входни операции**

   **Метод** Console.ReadLine – **въвежда низ от стандартния входен поток.**

   **Метод** Console.Parse – **преобразува символен низ в друг тип стойност.**

   низова_променлива = Console.ReadLine();
   друга_променлива = тип.Parse(Console.ReadLine());

2. **Изходни операции**

   **Методи** Console.WriteLine **и** Console.Write**.**

   Cosole.WriteLine (данни);
   Console.Write (данни);

**Пример:**

```
using System;
class CalculateFee
{
   static void Main(string[] args)
   {
         float dailyRate, fee;
         int noOfDays;
         Console.Write("Въведете дневна ставка: ");
         dailyRate = float.Parse(Console.ReadLine());
         Console.Write("Въведете брой работни дни: ");
         noOfDays = int.Parse(Console.ReadLine());
         fee = dailyRate * noOfDays;
         Console.WriteLine("Заплата: {0:C}", fee);
   }
}
```

**Пример: Числено преобразуване на низ**

```
int j = int.Parse (" 123456 ");
Console.WriteLine ("j={0}", j);                    // j=123456

float f = float.Parse ("-123.456");
Console.WriteLine ("f={0}", f);                    // f=-123.456

string s = j.ToString ();
Console.WriteLine ("s={0}", s);                    // s=123456
```

## Създаване на документация чрез XML (Extensible Markup Language)

/// <етикет>
/// описание
/// </етикет>

**Етикети**

| | |
|---|---|
| <summary> | **кратко резюме от един ред за клас, метод или свойство;** |
| <remarks> | **по-дълга и детайлна информация;** |
| <value> | **описание на свойство;** |
| <exception> | **изключения, вдигнати от методи и свойства;** |
| <param> | **параметри на метод.** |

1. **Solution Explorer ⇒ Project ⇒ Properties**
2. **Build ⇒ XML documentation file**
   **(bin/Debug/име_на_проект.xml)**

   **Създава се структурирана последователност с хипервръзки от HTML документи, основани на XML.**
3. **Solution Explorer ⇒ Show All Files**

**Пример:**

```
using System;
class CalculateFee
{   /// <summary>
    /// Изчислява заплатата при дадени дневна ставка и  брой работни дни.
    /// </summary>
    /// <remarks>
    /// Write и WriteLine извеждат данни в стандартния изходен поток.
    /// ReadLine въвежда символен низ от стандартния входен поток.
    /// Parse преобразува символен низ в друг тип стойност.
    /// </remarks>
    static void Main(string[] args)
    {   float dailyRate, fee;
        int noOfDays;
        Console.Write ("Въведете дневна ставка: ");
        dailyRate = float.Parse(Console.ReadLine());
        Console.Write("Въведете брой работни дни: ");
        noOfDays = int.Parse(Console.ReadLine());
        fee = dailyRate * noOfDays;
        Console.WriteLine ("Заплата: {0:C}", fee);
    }
}
```

```
<?xml version="1.0"?>
<doc>
   <assembly>
      <name> CalculateFee</name>
   </assembly>
   <members>
      <member name="M:CalculateFee.Main(System.String[])">
         <summary>
         Изчислява заплатата при дадени дневна ставка и  брой работни дни.
         </summary>
         <remarks>
         Write и WriteLine извеждат данни в стандартния изходен поток.
         ReadLine въвежда символен низ от стандартния входен поток.
         Parse преобразува символен низ в друг тип стойност.
         </remarks>
      </member>
   </members>
</doc>
```

## Обща система на типовете

### Клас System.Object

**Основният базов клас на всички класове в .NET Framework. Намира се на върха на йерархията от класове.**

**Методи**

```
public virtual bool Equals (object obj);
```

```
public static bool Equals (object objA, object objB);
```

**Определя дали два екземпляра на класа Object са равни.**

```
public virtual int GetHashCode ()
```

**Служи като хеш функция за даден тип, която се използва в хеширащи алгоритми и структури данни като хеш таблица.**

```
public Type GetType ();
```

**Връща типа на текущия екземпляр (инстанция). Класът Type представя типа на декларацията (клас, интерфейс, масив от стойностен тип, изброим тип).**

```
public static bool ReferenceEquals (object objA, object objB);
```

**Определя дали дадените екземпляри на класа Object са един и същ екземпляр.**

```
public virtual string ToString ();
```

**Връща String, който представя текущия Object.**

```
~Object();
```

**Методът Finalize в C# се представя чрез синтаксиса на деструктор. Позволява Object да освободи ресурсите и да осъществи почистващи операции преди това да се извърши от събирача на боклука (garbage collector). Извиква се автоматично, след като обектът стане недостижим, освен ако обектът е освободен чрез извикването на метода GC.SuppressFinalize.**

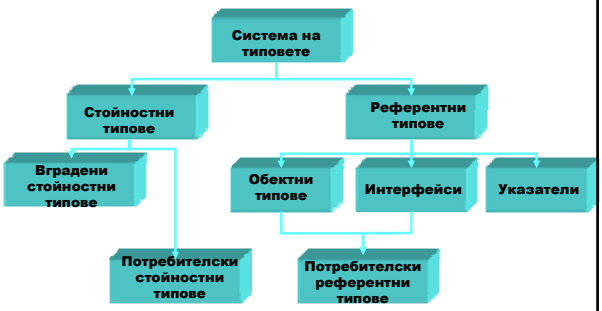
```
protected object MemberwiseClone ();
```

**Създава клонирано копие на текущия Object, което е от същия тип, но съдържа само нестатичните полета (за стойностните типове копира бит-по-бит, а за рефрентните типове копира референцията, но не и самия обект – така референцията на оригиналния обект и на клонирания сочат към един и същ обект).**

## Типове данни

**Тип - дефиниция, от която може да се създаде инстанция.**

**Обща система на типовете**




## Типове данни

I. **Стойностни типове – примитивни типове, структури и изброими типове**
   1. **Директно наследяват System.ValueType или System.Enum, които са наследници на класа System.Object.**
   2. **Съдържат действителните данни – директен достъп.**
   3. **Съхраняват се в стека.**
   4. **Не може да имат стойност null.**
   5. **Предава се стойността на променливата като параметър – променливата не се модифицира.**

II. **Референтни типове – класове, масиви, интерфейси и делегати**
   1. **Съдържат адреса на обект от определен тип – косвен достъп чрез указател към обекта.**
   2. **Съхраняват се в хийпа.**
   3. **Може да имат стойност null.**
   4. **Предава се адресът на обекта като параметър – обектът се модифицира.**

**Идентификатори – имена на елементите в програмата.**
- **букви (малки и главни) и цифри;**
- **започват с буква (долна черта _ е буква);**
- **разлика между главни и малки букви.**

result _score twentyOne plan9 TwentyOne

**Променливи – място за съхранение на стойност. Чрез името се осъществява достъп до съдържащата се стойност.**

**Правила за имена на променливи в .NET Framework**
- **да започват с малка буква;**
- **всяка следваща дума да започва с главна буква;**

  twentyOne
- **да не се използват долни черти;**
- **идентификаторите да не се различават само в това дали буквите са главни и малки.**

  myVariable **и** MyVariable

## Примитивни типове данни

| Тип | Размер [бит] | Област | Примери |
|---|---|---|---|
| byte | 8 | 0÷255 | byte b=42; |
| short | 16 | $-2^{16}$÷$2^{16}$-1 | short s=42; |
| int | 32 | $-2^{31}$÷$2^{31}$-1 | int count=42; |
| long | 64 | $-2^{63}$÷$2^{63}$-1 | long wait=42L; |
| float | 32 | ±3.4x$10^{38}$ | float away=0.42F; |
| double | 64 | ±1.7x$10^{308}$ | double trouble=0.42; |
| decimal | 128 | 28 значещи цифри | decimal coin=0.42M; (за финансови изчисления) |
| string | 16/символ | не се използва | string vest="42"; |
| char | 16 | 0÷$2^{16}$-1 | char grill='4'; |
| bool | 8 | true или false | bool flag=false; |
| object | базов тип за всички типове | | |

$2^{16}$=32768 $2^{31}$=2147483648 $2^{63}$=9223372036854775808

**Деклариране на променливи**

тип идентификатор;

**Оператор за присвояване =**

идентификатор = израз;

**Нулеви типове (за стойностни типове)**

- **разширяват стойностните типове със стойност** null
- **не трябва да бъдат декларирани преди да се използват**
- **за всеки стойностен тип** T **има съответстващ нулев тип** T?**, който може да съдържа допълнителната стойност** null**.**

```
int x = 3;
int? y = 5;
y += x;
if(y==null)
    Console.WriteLine("y=null");
else
    Console.WriteLine(y);              // 8
```

**Неявен тип** var

- **статично определяне на типа – по време на компилация според израза от дясната страна на инициализиращия оператор;**
- **само за локални променливи;**
- **изисква явно инициализиране на променливите.**

```
var идентификатор = израз;
```

**Динамичен тип** dynamic

- **динамично определяне на типа – по време на изпълнение;**
- **за локални променливи, полета и аргументи;**
- **не изисква явно инициализиране на променливите.**

```
dynamic идентификатор;
```

```
using System;
class Program
{
   public static dynamic Add(dynamic x, dynamic y)
   {
      return x + y;
   }
   static void Main()
   {
      var v1 = 5;
      v1 += 6;
      Console.WriteLine(v1);                    // 11
      var v2 = "123";
      v2 += 4;
      Console.WriteLine(v2);                    // 1234
      dynamic sum1 = Add(5, 6);                 // 11
      Console.WriteLine(sum1);
      dynamic sum2 = Add("123","4");
      Console.WriteLine(sum2);                  // 1234
   }
}
```

**Проверка за препълване при аритметични операции и преобразувания** checked/unchecked

```
checked блок
checked (израз)
unchecked блок
unchecked (израз)
```

**Преобразуване на типовете**

1. **Неявно преобразуване – към следващия по-висш тип.**
2. **Явно (принудително) преобразуване**

```
(тип) израз
checked ((тип) израз)
```

**При включена проверка** checked **се получава** OverflowException**, ако** типът **не може да побере резултантната стойност.**

**Опаковане (boxing) и разопаковане (unboxing)**

1. **Опаковане – преобразуване на стойностен тип в референтен тип.**

```
int i = 42;                 // стойностен тип
object bar = i;             // i е опакован за bar
```

2. **Разопаковане – преобразуване на референтен тип в стойностен тип – чрез принудително преобразуване на типа.**

```
int i = 42;                 // стойностен тип
object bar = i;             // i е опакован за bar
int baz = (int)bar;         // обратно разопаковане в int
```

**Изрази и оператори**

**Оператор**

1. **Символ за операция върху операнди.**
2. **Резултат**
   - **нова стойност от операцията върху операндите;**
   - **запазва се в паметта в променлива.**
3. **Видове – според броя на операндите:**
   - **унарни;**
   - **бинарни;**
   - **тринарни.**

**Приоритет и асоциативност**

1. **Ред на изпълнение на операторите – според приоритета.**
2. **Изчисление на ляв или десен операнд на израз – според асоциативността.**

## Приоритет на операциите

| Група на оператора | Оператори | Асоциа-тивност |
|---|---|---|
| Първични | (x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked | дясна |
| Унарни | + - ! ~ ++x --x (T)x | дясна |
| Мултипликативни | * / & | лява |
| Адитивни | + - | лява |
| Преместване | << >> | лява |
| Отношение | < > <= >= is as | лява |
| Равенство | == != | лява |
| Логически AND | & | лява |
| Логически XOR | ^ | лява |
| Логически OR | \| | лява |
| Условен AND | && | лява |
| Условен OR | \|\| | лява |
| Условен | ?: | лява |
| Присвояване | = += -= *= /= %= >>= <<= &= ^= != | дясна |
| Запетая | , | лява |

## Управляващи структури

**Категории:**

1. **Оператори за разклонения**
2. **Оператори за цикли**
3. **Оператори за преход**

## Оператори за разклонения

**Оператор** if

```
if (израз)
        оператор1
[else
        оператор2]
```

израз – **тип** bool

**Оператор** if-else-if

```
if (израз1)
        оператор1
else if (израз2)
        оператор2
else
        оператор3
```

**Оператор** switch

```
switch (израз)
{       case константен_израз1:
                оператор1
                оператор_за_преход
        ...
        case константен_изразN:
                операторN
                оператор_за_преход
        [default
                операторN+1
                оператор_за_преход]
}
```

израз – **тип** byte, short, int, long, char, string;
оператор_за_преход **(напр.** break**) – за всеки оператор** case**.**

## Оператори за цикли

**Оператор** while

```
while (логически_израз)
        оператор
```

**Оператор** do-while

```
do
        оператор
while (логически_израз);
```

**Оператор** for

```
for (инициализация; логически_израз; актуализация)
        оператор
```

**Оператор** foreach

foreach (тип идентификатор in израз)
оператор

```
string s = "Programming in the .NET Environment";
int count = 0;
foreach (char c in s)
  if (c>='A' && c<='Z')
    count++;
Console.WriteLine ("Броят на главните букви е " + count);
```

## Оператори за преход

**Оператор** break

break;

**Оператор** continue

continue;

**Оператор** goto

goto идентификатор

идентификатор: оператор

goto case константен_израз

goto default

**Оператор** return

return [върнат_израз]