Компютърно програмиране. Езици, системи и среди за програмиране

Sofia, 2022

C# - Основен синтаксис

C# е обектно-ориентиран език за програмиране.

В методологията на обектно-ориентираното програмиране програмата се състои от различни обекти, които взаимодействат помежду си чрез действия.

Действията, които един обект може да предприеме, се наричат методи.

За обекти от един и същи вид се казва, че имат един и същи тип или се казва, че са от един и същи клас.

Например, нека разгледаме обект Rectangle. Има атрибути дължина и ширина. Този обект разполага с метод за получаване на стойностите на атрибутите на правоъгълника, метод за изчисляване на площта и метод за показване на резултата. Нека разгледаме имплементацията на клас Rectangle и обсъдим основния синтаксис на C#:

Програма лице на правоъгълник

```
using System;
namespace RectangleApplication {
   class Rectangle {
      double length, width;       // member variables
      public void GetDimensions() {
         length = 4.5;     width = 3.5;
      }
      public double GetArea() {
         return length * width;
      }
      public void Display() {
         Console.WriteLine("Length: {0}", length);
         Console.WriteLine("Width: {0}", width);
         Console.WriteLine("Area: {0}", GetArea());
      }
   }
}
```

Клас, данни и методи на класа

```
class ExecuteRectangle {
      static void Main(string[] args) {
         Rectangle r = new Rectangle();
         r. GetDimensions();
         r.Display();
      }
   }
```

Когато горният код се компилира и изпълни, той дава следния резултат:

Length: 4.5
Width: 3.5
Area: 15.75

Променливите length и width са атрибути или членове на класа, използвани за съхраняване на данни.

Функциите са набор от изрази, които изпълняват конкретна задача. Функциите, членове на клас, се декларират в класа. Нашият примерен клас Rectangle съдържа три функции: `AcceptDetails, GetArea и Display`, които наричаме методи.

Създаване на клас ExecuteRectangle, който съдържа метода Main() и създава обект r от класа Rectangle. Методът Main() е входна точка за нашата програма.

Идентификатори

Идентификаторът е име, използвано за идентифициране на клас, променлива, функция или всеки друг дефиниран от потребителя елемент.

Основните правила за именуване на класове в C# са следните:

- Името трябва да започва с буква, която може да бъде последвана от поредица от букви, цифри (0 - 9) или долна черта.
- Първият знак в идентификатора не може да бъде цифра.
- Не трябва да съдържа вграден интервал или символ като? - + ! @ # % ^ & * ( ) [ ] { } . ; : " ' / и \.
- Въпреки това може да се използва долна черта (_).
- Не трябва да е C# ключова дума.

Ключови думи на C#

Ключовите думи са запазени думи, предварително дефинирани за C# компилатора. Те не могат да се използват като идентификатори. Обаче, ако искаме да използваме тези ключови думи като идентификатори, можем да поставим пред ключовата дума знака @.

В C# някои идентификатори имат специално значение в контекста на кода, като get и set се наричат контекстуални ключови думи.

Таблица на запазените ключови думи в C#:

abstract	as	base	bool	break	byte	case
catch	char	checked	class	const	continue	decimal
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	in	in gen.mo	int
interface	internal	is	lock	long	namespace	new

null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					
Контекстуални ключови думи						
add	alias	ascending	descending	dynamic	from	get
global	group	into	join	let	orderby	partial ty
partial me	remove	select	set			

Типове данни

Тип	Представлява	Диапазон	Стойност
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0}$ to 28	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F

Типове Данни

int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0

Вместо да се помнят числовите стойности от таблицата, да се използват служебните идентификатори на тези стойности, дефинирани в клас System.

Например:

```
Int32.MinValue , Int32.MaxValue за тип int
long.MaxValue, long.MinValue за тип long
float.MaxValue, float.MinValue за тип float
```

Разгледай програми: MinMaxValues.cs и MAX_Values.cs

Определяне на размера на типовете данни

Вътрешното представне на типовете данни в различните реализации на езика може да бъде различно. За да получим точния размер на тип или променлива на определена платформа, можем да използвате метода sizeof. Изразът sizeof(type) дава размера на паметта на обекта или типа в байтове.

Пример за получаване на размера на типа int на коя да е платформа:

```
using System;
namespace DataTypeApplication {
   class Program {
      static void Main(string[] args) {
         Console.WriteLine("Size of int: {0}", sizeof(int));
         Console.WriteLine("Size of double: {0}", sizeof(double));
      }
   }
}
```

Резултат:

Size of int: 4

Size of double: 8

Типове, допускащи стойност null

Или **Nullable value types** представят всички стойности на основния си тип плюс допълнителна нулева стойност. Например, да можем да присвоим някоя от следните три стойности на **bool?** променлива: true, false или null. Основният тип **bool** не може да приеме стойност **null**.

Деклариране и присвояване:

Можем да присвояваме стойност на променлива от тип допускащ стойност **null**, както бихме направили това за основния й тип. Можем обаче също така да присвоим нулева стойност (**null**).

Например:

```
double? pi = 3.14;
char? letter = 'a';
int m2 = 10;
int? m = m2;  /* Втори начин */ Nullable<int> m = null;

bool? flag = null;
int?[] arr = new int?[10]; // целичислен масив от тип допускащ
стойност null
```

Стойността по подразбиране на тип допускащ стойност null винаги е null !!!

Референтен тип

Референтните типове не съдържат действителните данни, съхранени в променлива, но съдържат препратка към променливите.

С други думи, те се отнасят към място в паметта. Използвайки множество променливи, референтните типове могат да се отнасят до даден адрес в паметта. Ако данните в местоположението на паметта се променят от една от променливите, другата променлива автоматично отразява тази промяна в стойността. Пример за вградени референтни типове са: `object, array, string`.

Нормално референтните променливи сочат към динамично заделена памет. Изпълнителната система автоматично извършва освобождаване на тази памет и реорганизира динамичните променливи, когато тя не е необходима повече. Инструментът, който извършва това се нарича garbage collector.

```
object obj; obj = 100; // this is boxing
object obj; obj = 100; // this is boxing
```

Тип обект (object)

Обектният тип е най-добрият базов клас за всички типове данни в C# Common Type System (CTS).

Object е псевдоним за клас System.Object. На типовете обекти могат да бъдат присвоени стойности от всякакви други типове, типове стойности, референтни типове, предварително дефинирани или дефинирани от потребителя типове. Въпреки това, преди да присвоите стойности, той се нуждае от преобразуване на типа.Когато даден тип стойност се преобразува в тип обект, това се нарича **boxing**(опаковане), а от друга страна, когато тип обект се преобразува в тип стойност, това се нарича **unboxing** (разопаковане).

object obj; obj = 100; // this is boxing
object obj; obj = 100; // this is boxing

Тип обект (object)

Например:

```
object obj;
Obj = 100; // Опаковане (boxing)
//-----------------------------------------

object container1 = 5;

object container2 = "Five";

// Print the results on the console

Console.WriteLine("The value of container1 is: " + container1);

Console.WriteLine("The value of container2 is: " + container2);

// Console output:

// The value of container is: 5

// The value of container2 is: Five.
```

Динамичен тип (dynamic)

Можете да съхранявате произволен тип стойност в променливата за динамичен тип данни. Проверката на типа за тези типове променливи се извършва по време на изпълнение.
Синтаксисът за деклариране на динамичен тип е:

```
dinamic <име_на_променлива> = стойност;
```

Например:

```
dynamic d = 20;
```

Динамичните типове са подобни на обектните типове, с изключение на това, че проверката на типа за променливите на обектния тип се извършва по време на компилиране, докато тази за променливите на динамичния тип се извършва по време на изпълнение.

Указатели

Променливите тип указател съхраняват адреса на паметта от друг тип.

Указателите в C# имат същите възможности като указателите в C или C++.

Синтаксисът за деклариране на тип указател е:

```
тип * идентификатор;
```

Например:

```
char* cptr;
int* iptr;
```

Низове

Символният низ е последователност от символи, записана на даден адрес в паметта. В променливите от тип **char** можем да запишем само един символ. Когато е необходимо да обработваме повече от един символ на помощ идват низовете.

В .NET Framework всеки символ има пореден номер от **Unicode** таблицата. Класът **System.String** позволява обработка на символни низове в C#. За декларация на низовете ще продължим да използваме служебната дума **string.**

```
string name = “Pano Panov";
```

Вътрешното представяне на класа е масив от символи.Типът **string** е по-особен от останалите типове данни. Той е клас и като такъв той спазва принципите на обектно-ориентираното програмиране. Стойностите му се записват в динамичната памет, а променливите от тип **string** пазят препратка към паметта (референция към обект в динамич-ната памет). Класът **string** има важна особеност – последователностите от символи, записани в променлива от класа, са **неизменими (immutable)**. След като е веднъж зададено, съдържанието на променливата не се променя директно – ако опитаме да променим стойността, тя ще бъде записана на ново място в динамичната памет, а променливата ще започне да сочи към него.

Низове и масиви от символи

Низвете много приличат на масиви от символи (**char[]**), но за разлика от тях не могат да се променят. Подобно на масивите те имат свойство **Length**, което връща дължината на низа, и позволяват достъп по индекс. Индексирането, както и при масивите, става по индекси от **0** до **Length-1**. Достъпът до символа на дадена позиция в даден стринг става с оператора **[]** (индексатор), но е позволен само за
четене:

```
string str = "abcde";
char ch = str[1]; // ch == 'b'
str[1] = 'a'; // Compilation error
ch = str[50]; // IndexOutOfRangeException
string name = “Pano Panov";
Console.WriteLine(“name = {0}", name);
Console.WriteLine(“name.Length = {0}", name.Length);
```

Деклариране на символен низ: `string str;`

Декларацията на символен низ представлява декларация на променлива от тип **string**.

Това не е еквивалентно на създаването на променлива и заделянето на памет за нея! С декларацията уведомяваме компилатора, че ще използваме променлива **str** и очакваният тип за нея е **string**. Ние не създаваме променливата в паметта и тя все още не е достъпна за обра-ботки (има стойност **null**, което означава липса на стойност).

Създаване на символен низ

За да използваме декларираната стрингова променлива, трябва да я създадем и инициализираме. Преди да зададем конкретна стойност на символния низ, стойността му е **null**. Неинициализираните променливи от типа **string** не съдържат празни стойности, а специалната стойност **null** – и опитът за манипулация на такъв стринг ще генерира грешка (изключение за достъп до липсваща стойност **NullReferenceException**)!

Можем да инициализираме променливи по 3 начина:

1. Чрез задаване на низов литерал.

```
string website = "http://www.introprogramming.info/";
```

2. Чрез присвояване стойността от друг символен низ.

```
string source = "Some source";
string assigned = source;
```

3. Чрез предаване стойността на операция, връщаща символен низ.

```
string email = "some@gmail.com";
string info = "My mail is: " + email;
// My mail is: some@gmail.com
```

Четене и печатане на конзолата

```
string name = Console.ReadLine();
Console.WriteLine("Your name is: " + name);
```

Сравнение за еднаквост на два символни низа:

```
string word1 = "C#";
string word2 = "c#";
Console.WriteLine(word1.Equals("C#"));
Console.WriteLine(word1.Equals(word2));
Console.WriteLine(word1 == "C#");
Console.WriteLine(word1 == word2);
// Console output:
// True
// False
// True
// False
```

Манипулиране на символни низове

Игнориране на разликата между малки и главни букви.
В примера по-долу при сравнение на "C#" със "c#" методът ще върне стойност **true**:

```
Console.WriteLine(word1.Equals(word2,
StringComparison.CurrentCultureIgnoreCase));
// True
```

Сравнение на низове по азбучен ред: методът **CompareTo(...)**
Методът **CompareTo(...)** от класа **String** връща отрицателна стойност, 0 или положителна стойност в зависимост от лексикографската подредба на двата низа, които се сравняват. Отрицателна стойност означава, че първият низ е лексикографски преди втория, нула означава, че двата низа са еднакви, а положителна стойност означава, че вторият низ е лексикографски преди първия.

```
string score = "sCore";
string scary = "scary";
Console.WriteLine(score.CompareTo(scary));
Console.WriteLine(scary.CompareTo(score));
Console.WriteLine(scary.CompareTo(scary));

// Console output: 1 -1 0
```

Долепване на низове (конкатенация)

Долепването на символни низове и получаването на нов низ като резултат, се нарича **конкатенация**. То може да бъде извършено по няколко начина: чрез метода **Concat(...)** или чрез операторите + и +=.

Пример за използване на функцията **Concat(...)**:

```
string greet = "Hello, ";
string name = "reader!";
string result = string.Concat(greet, name);
```

Долепване на низове (конкатенация)

Вторият вариант за конкатенация е чрез операторите + и +=.

Горният пример може да реализираме още по следния начин:

```
string greet = "Hello, ";
string name = "reader!";
string result = greet + name;
result = result + " How are you?";
result += " How are you?";

string message = "The number of the beast is: ";
int beastNum = 666;
string result = message + beastNum;
// The number of the beast is: 666
```

Извличане на част от низ

Можем да проверим дали даден подниз се среща в даден текст и на кои позиции се среща. Как обаче да извлечем част от низа в отделна променлива?

Решението на проблема ни е методът **Substring(...)**. Използвайки го, можем да извлечем дадена част от низ (подниз) по зададени начална позиция в текста и дължина. Ако дължината бъде пропусната, ще бъде направена извадка от текста, започваща от началната позиция до неговия край.

Следва пример за извличане на подниз от даден низ:

```
string path = "C:\\Pics\\Rila2010.jpg";
string fileName = path.Substring(8, 8);
// fileName = "Rila2010"
```

Извикването на метода Substring(startIndex, length) извлича подниз от даден стринг, който се намира между startIndex и (startIndex + length – 1) включително. Символът на позицията startIndex + length не се взима предвид! Например ако посочим Substring(8, 3), ще бъдат извлечени символите между индекс 8 и 10 включително.

Замяна на подниз с друг

Текстообработката в .NET Framework предлага готови методи за замяна на един подниз с друг.

Например, ако сме допуснали една и съща техническа грешка при въвеждане на email адреса на даден потребител в официален документ, можем да го заменим с помощта на метода **Replace(...)**:

```
string doc = "Hello, some@gmail.com, " +
"you have been using some@gmail.com in your registration.";
string fixedDoc =
doc.Replace("some@gmail.com", "osama@bin-laden.af");
Console.WriteLine(fixedDoc);
// Console output:
// Hello, osama@bin-laden.af, you have been using
// osama@bin-laden.af in your registration.
```

Разделяне на низ по множество от разделители

```
string listOfBeers = "Amstel, Zagorka, Tuborg, Becks";
char[] separators = new char[] {' ', ',', '.'};
string[] beersArr = listOfBeers.Split(separators);
```

Конвертиране на типове

Преобразуването на тип е преобразуване на един тип данни в друг тип. Известен е още като тип кастинг. В C# преобразуването на типове има две форми

–**Неявно преобразуване на тип (Implicit type conversion)**– Тези преобразувания се извършват от C# по безопасен за типа начин. Например, са преобразувания от по-малки към по-големи интегрални типове и преобразувания от производни класове към базови класове.

Изрично преобразуване на тип (Explicit type conversion) – Тези преобразувания се извършват изрично от потребители, използващи предварително дефинираните функции. Явните преобразувания изискват оператор за преобразуване.

```
using System;
namespace TypeConversionApplication { //explicit type conversion
      class ExplicitConversion {
       static void Main(string[] args) {
       double d = 5673.74;
       int i;              // cast double to int.
       i = (int )d;
       Console.WriteLine(i);
    }
  }
}
```

Методи за преобразуване на типа

ToBoolean - Преобразува тип в булева стойност, където е възможно.
ToByte - Преобразува тип в байт.
ToChar - Преобразува тип в един Unicode знак, където е възможно.
ToDateTime - Преобразува тип (цяло число или тип низ) в структури дата-час.
ToDecimal - Преобразува тип float, double или int в десетичен тип.
ToDouble - Преобразува тип в двоен тип.
ToInt16 - Преобразува тип в 16-битово цяло число.
ToInt32 - Преобразува тип в 32-битово цяло число.
ToInt64 - Преобразува тип в 64-битово цяло число.
ToSbyte - Преобразува тип в подписан тип байт.
ToSingle - Преобразува тип в малко число с плаваща запетая.
ToString - Преобразува тип в низ.
ToType - Преобразува тип в определен тип.
ToUInt16 - Преобразува тип в unsigned int тип.
ToUInt32 - Преобразува тип в неподписан дълъг тип.
ToUInt64 - Преобразува тип в голямо цяло число без знак.

Конвертиране на типове - примерна програма

```
using System;
namespace TypeConversionApplication {
  class StringConversion {
    static void Main(string[] args) {
      int i = 75;
      float f = 53.005f;
      double d = 2345.7652;
      bool b = true;
      Console.WriteLine(i.ToString());
      Console.WriteLine(f.ToString());
      Console.WriteLine(d.ToString());
      Console.WriteLine(b.ToString());
    }
  }
}
```

Резултат: 75 53.005 2345.7652 True

Конвертиране на низ в цяло число int

В C# можем да конвертираме низово представяне на число в цяло число по следните начини:

- Parse() method
- Convert class
- TryParse() method – Препоръчително е да се използва

Методите Parse() са приложими за всички примитивни типове данни. Това е най-лесният начин за преобразуване от низ в цяло число.

- Int16.Parse()
- Int32.Parse()
- Int64.Parse()

Методите приемат 1, 2 или 3 параметъра в зависимост от представянето на числото в низа

Parse(string s)
Parse(string s, numberstyle style)
Parse(String s, NumberStyles style, IFormatProvider provider)

```
Int16.Parse("100"); // returns 100
Int16.Parse("(100)", NumberStyles.AllowParentheses); // returns -100
int.Parse("30,000", NumberStyles.AllowThousands, new CultureInfo("en-au"));
// returns 30000
int.Parse("$ 10000", NumberStyles.AllowCurrencySymbol); //returns 10000
int.Parse("-100", NumberStyles.AllowLeadingSign); // returns -100
int.Parse(" 100 ", NumberStyles.AllowLeadingWhite | NumberStyles.AllowTrailingWhite); // returns 100
Int64.Parse("2147483649"); // returns 2147483649
```

Валиден числов низ може да бъде преобразуван в цяло число. Методът Parse() позволява преобразуване на числовия низ в различни формати в цяло число с помощта на числови стилове enum.Предаденият низ обаче трябва да бъде валиден числов низ или в обхвата на типа. Следните твърдения хвърлят изключения.

Пример: Невалидно преобразуване

```
int.Parse(null);                    // хвърля FormatException
int.Parse("");                      // хвърля FormatException
int.Parse("100.00");                // хвърля FormatException
int.Parse( "100a");                 // хвърля formatexception
int.Parse(2147483649); // хвърля overflow exception use Int64.Parse()
```

Клас Convert за преобразуване

Друг начин за преобразуване на низ в цяло число е чрез използване на статичен клас Convert. Класът Convert включва различни методи, които преобразуват основния тип данни в друг основен тип данни.Класът Convert включва следните методи за преобразуване от различни типове данни в тип int.

• Convert.ToInt16() връща 16-битовото цяло число
• Convert.ToInt32() връща 32-битови цели числа
• Convert.ToInt64() връща 64-битовото цяло число

Пример: Преобразувайте низ в int с помощта на клас ConvertConvert.

```
ToInt16("100"); // връща краткоConvert.
ToInt16(null);//връща 0
Convert.ToInt32("233300");// връща int
Convert.ToInt32("1234",16); // връща 4660 – шестнадесетичното 1234
Convert.ToInt64("1003232131321321");//връща дълго
// следните хвърлят изключения
Convert.ToInt16("");//хвърля FormatException
Convert.ToInt32("30 000"); //хвърля FormatException
Convert.ToInt16("(100)");//хвърля FormatException
Convert.ToInt16("100a"); //хвърля FormatException
Convert.ToInt16(2147483649);//хвърля OverflowException
```

Заключение за клас Convert

Плюсове: Предимства

1. Преобразува от всеки тип данни в цяло число.
2. Преобразува null в 0, така че не хвърля изключение.

Минуси:

1. Въведеният низ трябва да бъде валиден числов низ, не може да включва различни цифрови формати. Работи само с валиден низ от цели числа.
2. Входящият низ трябва да бъде в обхвата на извикания метод IntXX, напр. Int16, Int32, Int64.
3. Въведеният низ не може да включва скоби, запетая и др.
4. Трябва да използвате различен метод за различни цели числа, напр. не може да използва Convert.ToInt16() за целочислен низ, по-висок от "32767".

Метод TryParse

Методите TryParse() са достъпни за всички примитивни типове за преобразуване на низ в извикващия тип данни. Това е препоръчителният начин за преобразуване на низ в цяло число.Методът TryParse() преобразува низовото представяне на число в неговия 16, 32 и 64-битов еквивалент на цяло число със знак. Връща булево значение, което показва дали преобразуването е успешно или неуспешно и така никога не хвърля изключения.Методите TryParse() са налични за всички цели числа:

Int16.TryParse()
Int32.TryParse()
Int64.TryParse()

Методите се Overload-ват, както методите Parse приемайки допълнителни два параметъра, идентични на метода Parse() със същата функционалност.Следващият пример демонстрира метода TryParse().

Пример: Преобразуване на низ в int с помощта на TryParse():

```
static void Main(string[] args)
{
        string str = "2022";
        int intStr; bool intResultTryParse = int.TryParse(str, out intStr);
        if (intResultTryParse == true)
        {
           Console.WriteLine(intStr);
        }
        else
        {
           Console.WriteLine("Input is not in integer format");
        }
}
```

Въпроси и задачи

1. Декларирайте няколко променливи, като изберете за всяка една най-подходящия от типовете **sbyte**, **byte**, **short**, **ushort**, **int**, **uint**, **long** и **ulong**, за да им присвоите следните стойности: 52130, -115, 4825932, 97, -10000, 20000; 224; 970700000; 112; -44; -1000000; 1990; 12345678912345678 9.
2. Кои от следните стойности може да се присвоят на променливи от тип **float**, **double** и **decimal**: 34.567839023; 12.345; 8923.1234857; 3456.09112487595654215125668 3467?
3. Декларирайте две променливи от тип **string** със стойности "Hello" и "World". Декларирайте променлива от тип **object**. Присвоете на тази променлива стойността, която се получава от конкатенацията на двете стрингови променливи (добавете интервал, ако е необходимо). Отпечатайте променливата от тип **object**.
4. Напишете програма, която принтира на конзолата равнобедрен триъгълник, като страните му са очертани от символа звездичка "©".
5. Фирма, занимаваща се с маркетинг, иска да пази запис с данни на нейните служители. Всеки запис трябва да има следната характе-ристика – първо име, фамилия, възраст, пол („м" или „ж") и уникален номер на служителя (27560000 до 27569999). Декларирайте необходимите променливи, нужни за да се запази информацията за един служител, като използвате подходящи типове данни и описателни имена.
6. Декларирайте две променливи от тип **int**. Задайте им стойности съответно 5 и 10. Разменете стойностите им и ги отпечатайте. С и без трета променлива.

Изгревът е близо

Благодаря за вниманието!

Готов съм да отговарям на вашите въпроси