Programare C#

Bibliografie

•Herbert Schildt, *C#: A Beginner's Guide, (2001)*;

•Herbert Schildt, *C#*, Ed.Teora (traducere, 2002);

•Karli Watson et al., *Beginning Visual C#*, Wrox Press Ltd. (2002);

•Karli Watson, *Beginning C# 2005 Databases*, Wiley Publishing, Inc. (2006);

•Bradley L. Jones, *SAMS Teach Yourself the C# Language in 21 Days*, *(*2004);

•Philip Syme si Peter Aitken, *SAMS Teach Yourself the C# Web Programming in 21 Days*, (2002);

•Kris Jamsa si Lars Klander, *Totul despre C si C++ Manualul fundamental de programare in C si C++*, *Ed. Teora*, (traducere 2007);

Introducere

- Scurt istoric;
- Relatia dintre C# si arhitectura .NET;
- Principiile programarii orientate obiect;
- Crearea, compilarea si executia programelor C#. Exemple;

Scurt istoric

Lansat publicului in iunie 2000 si oficial in primavara anului 2002, C# este un limbaj de programare care combina facilitati testate de-a lungul timpului cu inovatii de ultim moment. Creatorii acestui limbaj au fost o echipa de la firma Microsoft condusa de Anders Hejlsberg. Desi limbajul este creat de Microsoft, acesta nu este destinat doar platformelor Microsoft. Compilatoare C# exista si pentru alte sisteme precum Linux sau Macintosh. Creat ca instrument de dezvoltare pentru arhitectura .NET, limbajul ofera o modalitate facila si eficienta de a scrie programe pentru sistemul Windows, internet, componente software etc.

C# deriva din doua dintre cele mai de succes limbaje de programare: C si C++. De asemenea, limbajul este o “ruda” apropiata a limbajului Java. Pentru o mai buna intelegere a limbajului C# este interesant de remarcat care este natura relatiilor acestuia cu celelalte trei limbaje mentionate mai sus. Pentru aceasta, vom plasa mai intai limbajul C# in contextul istoric determinat de cele trei limbaje.

Limbajul C. Programarea structurata

Limbajul C a fost inventat de catre Dennis Ritchie in anii ‘70 pe un calculator pe care rula sistemul de operare UNIX. Limbajul C s-a dezvoltat in urma revolutiei programarii structurate din anii ’60. Inainte de programarea structurata, programele erau greu de scris si de inteles din cauza logicii. O masa incalcita de salturi, apelui si reveniri, greu de urmarit era cunoscuta sub numele de cod spaghetti. Datorita sintaxei sale concise si usor de utilizat, in anii ‘80, limbajul C a devenit cel mai raspandit limbaj structurat.

Limbajul C are insa limitele sale. Una dintre acestea o reprezinta incapacitatea de a lucra cu programe mari. Limbajul C ridica o bariera atunci cand programul atinge o anumita dimensiune. Acest prag depinde de program, instrumentele folosite, programator, dar este posibil sa se situeze in jurul a 5000 de linii de cod.

Limbajul C++. Programarea orientata obiect

La sfarsitul anilor ’70 dimensiunile multor programe erau aproape de limitele impuse de limbajul C. Pentru a rezolva problema a aparut o modalitate noua de programare si anume programarea orientata obiect (POO). Limbajul C nu permitea programarea orientata obiect. Fiind cel mai raspandit limbaj, s-a dorit extinderea sa in vederea implementarii noii modalitati de programare: programarea POO.

Limbajul C++ a fost creat de catre Bjarne Stroustrup incepand din 1979, la laboratoarele Bell din Murray Hill, New Jersey. Limbajul a fost denumit initial C cu clase, iar in 1983 numele acestuia a fost modificat in C++. In esenta, C++ reprezinta versiunea orientata obiect a limbajului C. In anii ’80, limbajul C++ a suferit dezvoltarii si perfectionari masive, astfel ca in anii ’90 a devenit cel mai raspandit limbaj de programare.

Limbajul Java. Problema portabilitatii

Lucrul la acest limbaj a fost demarat in 1991 la firma Sun Microsystems. Java este un limbaj structurat si orientat pe obiecte, cu o sintaxa si filozofie derivate din C++. Aspectele novatoare se refera mai mult la modificarile mediului de programare. Aspectul esential in Java este posibilitatea de a crea cod portabil pe platforme diferite. Inainte de explozia Internetului, majoritatea programelor erau compilate si destinate utilizarii pe un anumit procesor si sub un anumit sistem de operare. Programele scrise in C si C++ se compilau intodeauna pana la cod masina executabil. Codul masina este legat de un anumit procesor si de un anumit sistem de operare. Dupa aparitia Internetului insa, la care sunt conectate sisteme cu procesoare si sisteme de operare diferite, problema portabilitatii a devenit foarte importanta.

Java a realizat portabilitatea prin transformarea codului sursa al programului intr-un cod intermediar numit *bytecode.* Acest format intermediar este executat apoi de asa numita Masina Virtuala Java (MVJ). Asadar, programele Java pot rula in orice mediu in care este disponibila o MVJ. Deoarece MVJ este usor de implementat, aceasta a fost imediat disponibila pentru un numar mare de medii.

Limbajul C#

Desi Java a rezolvat cu succes problema portabilitatii, exista unele aspecte care ii lipsesc. Una dintre acestea este *interoperabilitatea limbajelor diferite*, sau *programarea in limbaj mixt* (posibilitatea codului scris intr-un limbaj de a lucra in mod natural cu codul scris in alt limbaj). Interoperabilitatea limbajelor diferite este esentiala la realizarea sistemelor software de dimensiuni mari.

Ca parte a ansamblului strategiei .NET, dezvoltata de Microsoft, la finele anilor ’90 a fost creat limbajul C#. C# este direct inrudit cu C, C++ si Java. “Bunicul” limbajului C# este C-ul. De la C, C# mosteneste sintaxa, multe din cuvintele cheie si operatorii. De asemenea, C# construieste peste modelul de obiecte definit in in C++. Relatia dintre C# si Java este mai complicata. Java deriva la randul sau din C si C++. Ca si Java, C# a fost proiectat pentru a produce cod portabil. Limbajul C# nu deriva din Java. Intre C# si Java exista o relatie similara celei dintre “veri”, ele deriva din acelasi stramos, dar deosebindu-se prin multe caracteristici importante.

Limbajul C# contine mai multe facilitati novatoare, dintre care cele mai importante se refera la suportul incorporat pentru componente software. C# dispune de facilitati care implementeaza direct elementele care alcatuiesc componentele software, cum ar fi proprietatile, metodele si evenimentele. Poate cea mai importanta facilitate de care dispune C# este posibilitatea de a lucra intr-un mediu cu limbaj mixt.

Relatia dintre C# si arhitectura .NET

C# are o legatura deosebita cu mediul sau de rulare, arhitectura .NET. Pe de o parte, C# a fost dezvoltat pentru crearea codului pentru arhitectura .NET, iar pe de alta parte bibliotecile utilizate de C# sunt cele ale arhitecturii .NET.

Ce este arhitectura .NET ?

Arhitectura .NET defineste un mediu de programare care permite dezvoltarea si executia aplicatiilor indiferent de platforma. Aceasta permite programarea in limbaj mixt si ofera facilitati de securitate si portabilitate a programelor. Este disponibila deocamdata pentru platformele Windows.

Legat de C#, arhitectura .NET defineste doua entitati importante si anume *biblioteca de clase .NET* si *motorul comun de programare* sau *Common Language Runtime (CLR)*.

C# nu are o biblioteca de clase proprie ci utilizeaza direct biblioteca de clase .NET. De exemplu, cand se ruleaza un program care efectueaza operatii de intrare-iesire, cum ar fi afisarea unui text pe ecran, se utilizeaza biblioteca de clase .NET.

Motorul comun de programare (CLR) se ocupa de executia programelor C#. El asigura de asemenea programarea in limbaj mixt, securitatea si portabilitatea programelor. Atunci cand este compilat un program C#, sau un program in limbaj mixt, rezultatul compilarii nu este un cod executabil. In locul acestuia, se produce un fisier care contine un tip de pseudocod numit limbaj intermediar sau pe scurt IL (Intermediate Language). Acest fisier IL poate fi copiat in orice calculator care dispune de .NET CLR. Prin intermediul unui compilator denumit JIT (Just In Time), motorul comun de pogramare transforma codul intermediar in cod executabil. Procesul de conversie decurge astfel: atunci cand un program .NET este executat, CLR activeaza compilatorul JIT. Compilatorul JIT converteste IL in cod executabil pe masura ce fiecare parte a programului este neceasra. In concluzie, orice program compilat pana in format IL poate rula in orice mediu pentru care CLR este implementat. In acest fel arhitectura .NET asigura portabilitatea.

Principiile programarii orientate obiect

Metodologiile de programare s-au modificat continuu de la aparitia calculatoarelor pentru a tine pasul cu marirea complexitatii programelor. Pentru primele calculatoare programarea se facea introducand instructiunile masina scrise in binar. Pe masura ce programele au crescut s-a inventat limbajul de asamblare, in care se puteau gestiona programe mai mari prin utilizarea unor reprezentari simbolice ale instructiunilor masina. Cum programele continuau sa creasca, s-au introdus limbaje de nivel inalt, precum FORTRAN si COBOL, iar apoi s-a inventat programarea structurata.

POO a preluat cele mai bune idei de la programarea structurata, combinanu-le cu concepte noi. A rezultat o modlitate diferita de a organiza un program. In fapt, un program poate fi organizat in doua moduri: in jurul codului (mod de lucru descris de sintagma *“codul actioneaza asupra datelor”*, valabil in cazul programarii structurate) sau in jurul datelor (abordare descrisa de sintagma *“datele controleaza accesul la cod”*, valabila in cazul programarii orientate obiect).

Toate limbajele POO au patru caracteristici comune: incapsularea, polimorfismul, mostenirea si reutilizarea.

Incapsularea

Incapsularea este un mecanism care combina codul si datele pe care le manipuleaza, mentinand integritatea acestora fata de interferenta cu lumea exterioara. Incapsularea mai este numita si realizarea de cutii negre, intrucat se ascunde functionalitatea proceselor. Cand codul si datele sunt incapsulate se creaza un obiect. In cadrul unui obiect, codul si datele pot fi publice sau private. Codul si datele private sunt accesibile doar in cadrul aceluiasi obiect, in timp ce codul si datele publice pot fi utilizate si din parti ale programului care exista in afara acelui obiect.

Unitatea fundamentala de incapsulare este clasa. Clasa specifica datele si codul care opereaza asupra datelor. O clasa defineste forma unui obiect. Sau altfel spus, o clasa reprezinta o matrita, iar un obiect reprezinta o instanta a clasei.

Polimorfismul

Polimorfismul este calitatea care permite unei interfete sa aiba acces la un grup generic de actiuni. Termenul este derivat dintr-un cuvant grecesc avand semnificatia “cu mai multe forme”. Spre exemplu, sa presupunem ca avem o nevoie de o routina care sa returneze aria unei forme geometrice, care poate fi un triunghi, cerc sau trapez. Intrucat ariile celor trei forme se calculeaza diferit, routina trebuie sa fie adaptata la datele pe care le primeste incat sa distinga despre ce fel de forma este vorba si sa returneze rezultatul corect.

Conceptul de polimorfism este exprimat prin sintagma “o singura interfata mai mlte metode”.

Mostenirea

Mostenirea este procesul prin care un obiect poate dobandi caracteristicile altui obiect. Analogia cu conceptul de animal este elocventa. Spre exemplu, sa consideram o reptila. Aceasta are toate caracteristicile unui animal, insa in plus are si o alta caracteristica, si anume: sangele rece. Sa consideram un sarpe. Acesta este o reptila lunga si subtire care nu are picioare. Sarpele are toate caracteristicile unei reptile, insa poseda si propriile sale caracteristici. Asadar, un sarpe mosteneste caracteristicile unei reptile. O reptila mosteneste caracteristicile unui animal. Asadar, mecanismul mostenirii este cel care face posibil ca un obiect sa fie o instanta a unui caz mai general.

Reutilizarea

Atunci cand este creata o clasa, aceasta poate fi utilizata pentru a crea o multime de obiecte. Prin utilizarea mostenirii si incapsularii clasa amintita poate fi reutilizata. Nu mai este nevoie sa testam codul respectiv ci doar a il utlizam corect.

Crearea, compilarea si executia programelor C#. Exemple;

Sa consideram urmatorul program C# simplu:

```
/* Acesta este un program simplu in C#
Denumiti programul: Example1.cs */
using System;
class Example1
{
   //orice program in C# contine metoda Main()
   public static void Main()
   {
      Console.WriteLine("This is my first C# program");
   }
}
```

Exista doua moduri de a edita, compila si rula un program in C#. In primul rand se poate utiliza compilatorul linie de comanda csc.exe. A doua posibilitate este de a utiliza utilizati mediul Visual Studio .NET. In primul caz trebuie parcursi urmatorii pasi: introduceti textul programului cu ajutorul unui editor de texte si salvati fisierul utilizand extensia *cs,* spre exemplu *Example1.cs;* apoi compilati programul precizand numele fisierului in linia de comanda (C:\>csc Example1.cs); in final rulati programul in linia de comanda (C:\>Example). In cel de-al doilea caz creati un nou proiect C# selectand: File|New|Project, apoi Visual C# Projects|Empty Project. Dupa ce ati creat proiectul, executati click dreapta pe fereastra Solution. Utilizand meniul aparut selectati Add apoi Add New Item | Local Project Items| C# Code File. Introduceti textul, salvati proiectul, compilati proiectul selectand Build si in fine rulati programul selectand Start Without Debugging din meniul Debug.

Programul de mai jos creaza o aplicatie Windows.

```
using System;
using System.Windows.Forms;

 public class MyForm : Form
 {
   private TextBox txtEnter;
   private Label lblDisplay;
   private Button btnOk;

   public MyForm()
   {
      this.txtEnter = new TextBox();
      this.lblDisplay = new Label();
      this.btnOk = new Button();

      this.Text = "Prima mea aplicatie
                              Windows!";
      this.Size=new System.Drawing.Size(320,
                300);

      // txtEnter 1
       this.txtEnter.Location = new
                System.Drawing.Point(16, 32);
      this.txtEnter.Size = new
                System.Drawing.Size(264, 20);
      // lblDisplay
      this.lblDisplay.Location = new
                System.Drawing.Point(16, 72);
      this.lblDisplay.Size = new
                System.Drawing.Size(264, 128);

       // btnOk
      this.btnOk.Location = new
                System.Drawing.Point(88, 224);
      this.btnOk.Text = "OK";
      this.btnOk.Click +=new
                System.EventHandler(this.btnOK_Click);
      // MyForm
      this.Controls.AddRange(new Control[] {
                this.txtEnter, this.lblDisplay, this.btnOk});
      }

      static void Main ()
      {
                Application.Run(new MyForm());
      }

      private void btnOK_Click(object sender,
      System.EventArgs e)
      {
                lblDisplay.Text = txtEnter.Text + "\n" +
      lblDisplay.Text;
      }
}
```

Programul de mai jos descompune un numar natural in factori primi

```
using System;

class Descompunere
{
    public static void Main()
    {
        int n;
        int count=2;
        string l;
        Console.WriteLine("Introduceti numarul natural n");
        l=Console.ReadLine();
        n = int.Parse(l);
        Console.Write("{0}=", n);
        while (count <= n)
        {
            while (n % count == 0)
            {
                n = n / count;
                Console.Write("{0} ", count);
            }
            count++;
        }

    }
}
```

Tipuri de date si operatori

- Tipuri valorice in C#
- Partile componente ale unei aplicatii C#
- Literali
- Variabile
- Operatori
- Conversia tipurilor de date
- Studiul expresiilor

Tipuri valorice in C#

Tipurile de date si operatorii stau la baza oricarui limbaj de programare. C# ofera o gama larga de tipuri de date si operatori. Vom incepe prin examinarea tipurilor de date fundamentale in C#. Inainte de aceasta, amintim ca limbajul este puternic tipizat. Aceasta inseamna ca pentru toate operatiile, compilatorul realizeaza verificari asupra compatibilitatii tipurilor.

Limbajul C# include doua categorii generale de tipuri predefinite: *tipuri valorice* si *tipuri referinta*. Tipurile referinta din C# sunt definite de clase. Studiul acestora il vom face atunci cand vom discuta despre clase. La baza limbajului C# stau 13 tipuri valorice numite si tipuri simple. Aceasta datorita faptului ca exista o relatie directa intre tipurile de date C# si tipurile de date .NET.

Din ratiuni de portablitate, in C#, fiecare dintre tipurile valorice are domeniu fix de valori. Daca de exemplu in limbajul C, o varabila de tip int este reprezentata pe 2 octeti sau 4 octeti, in functie de platforma utilizata, in C# unei variabile de tip int, calculatorul ii aloca 4 octeti, indiferent de mediul de executie.

In tabelul de mai jos sunt prezentate aceste tipuri

Tipul in C#	Tipul in .NET	Semnificatia	Largimea (in octeti)	Domeniul
bool	System.Boolean	Valorile de adevar (adevarat/fals)	1	false(0) *la* true(1)
char	System.Char	Caractere	2	0 *la* 65535
byte	System.Byte	Intregi pe 8 biti, fara semn	1	0 *la* 255 (0 la 2^8-1)
sbyte	System.Sbyte	Intregi pe 8 biti, cu semn	1	-128 *la* 127
short	System.Int16	Intregi in forma scurta	2	-32768 *la* 32767
ushort	System.Uint16	Intregi in forma scurta, fara semn	2	0 *la* 65535 (0 la 2^{16}-1)
int	System.Int32	Intregi	4	-2147483648 *la* 214748367
uint	System.Uint32	Intregi, fara semn	4	0 *la* 4294967295 (0 la 2^{32}-1)
long	System.Int64	Intregi in forma lunga	8	-9223327036854775808 *la* 9223372036854775807
ulong	System.Uint64	Intregi in forma lunga, fara semn	8	0 *la* 18446744073709551615 (0 la 2^{64}-1)
float	System.Single	Virgula mobila, simpla precizie	4	$1.5x10^{-45}$ *la* $3.4x10^{38}$
double	System.Double	Virgula mobila, dubla precizie	8	$5.0x10^{-324}$ *la* $1.7x10^{308}$
decimal	System.Decimal	Tip numeric cu 28 cifre semnificative	16	$1.0x10^{-28}$ *la approx.* $7.9x10^{28}$

Intregi

In C# sunt definite noua tipuri intregi: *char, byte, sbyte, short, ushort, int, uint, long* si *ulong.*

Tipul *char.* Caracterele nu sunt reprezentate pe 8 biti ca in alte limbaje (spre exemplu C sau C++). In C# se utilizeaza modelul Unicode. Acesta defineste un set de caractere care poate repezenta caracterele din toate limbile de pe Pamant. Setul de caractere ASCII pe 8 biti, cuprins intre 0 si 127 este o submultime a modelului Unicode. Putem atribui o valoare de tip caracter daca includem caracterul intre apostrofuri simple. Exemplu: *char ch='M' ;*

Desi *char* este de tip intreg, nu poate fi amestecat la intamplare cu valori intregi deoarece nu se efectueaza conversii automate intre char si celelalte tipuri intregi. Codul de mai jos este incorect: *char ch=77;* Motivul pentru care instructiunea nu functioneaza este ca 77 este o valoare intreaga si nu este convertita automat la tipul char. Pentru a converti o valoare intreaga intr-un char vom realiza o conversie explicita (un cast). Pentru corectarea codului din exemplul anterior trebuie sa rescriem codul in forma: *char ch=(char) 77;*

Celelalte tipuri intregi sunt utilizate pentru calcule numerice. Sunt definite atat versiuni cu semn cat si fara semn. La fel ca in limbajul C, diferenta intre intregii cu semn si cei fara semn este data de interpretarea bitului cel mai semnificativ. Daca este specificat un intreg cu semn, atunci numarul este pozitiv daca bitul de semn este 0 si respectiv negativ daca bitul de semn are valoarea 1. Sa consideram cateva exemple:

sbyte: 00000001 (nr. 1); 01111111 (nr. 127); 10000000 (nr. -128); 11111111 (nr. -1);

byte: 00000001 (nr. 1); 01111111 (nr. 127); 10000000 (nr. 128); 11111111 (nr. 255);

Exemple: 1. Conversie implicita. 2. Conversie Explicita

```
using System;
class Demo
{   public static void Main()
    {   char sourceVar='a';
        ushort destinationVar;
        destinationVar = sourceVar;
        Console.WriteLine("destinationVar val:{0}", destinationVar);
    }
}
```

```
using System;
class Demo
{   public static void Main()
    {   ushort sourceVar = 97;
        char destinationVar;
        destinationVar = (char)sourceVar;
        Console.WriteLine("destinationVar val:{0}", destinationVar);
     }
}
```

Tipuri in virgula mobila

Tipurile in virgula mobila se utilizeaza pentru specificarea numerelor care au parte fractionara. Aceste tipuri sunt: *float* si *double*. Tipul float poate reprezenta in mod precis pana la 7 pozitii zecimale, in timp ce tipul double reprezinta 15 sau 16 zecimale exacte. Dintre cele doua, double este cel mai intrebuintat.

Tipul decimal

Tipul decimal nu este definit in C, C++ si Java. Acest tip utilizeaza 128 biti de memorie pentru a reprezenta valori cuprinse intre $1.0x10^{-28}$ si $7.9x10^{28}$. Este destinat calculelor monetare, putand reprezenta in mod precis 28 de pozitii zecimale. Nu exista conversii automate intre tipurile decimal si float sau decimal si double. Exemplu:

```
using System;
class FolosDecimal
{
   public static void Main()
   {   decimal sold, dobanda;
       //calculul noului sold
       sold = 10000.5m;      //literalii de tip decimal trebuie urmati de m sau M
       dobanda = 0.04m;
       sold = sold * dobanda + sold;
       Console.WriteLine("Noul sold este {0} EUR",sold);
   }
}
```

Rezultat: Noul sold este 10400.520 EUR

Tipul bool

Tipul bool reprezinta valorile de adevar true sau false. Orice variabila de tip bool va lua una dintre aceste valori. Nu este definita o regula de conversie intre tipul bool si valori intregi.

Cateva optiuni de afisare

Sa consideram instructiunea: Console.WriteLine("Valoarea lui 10/3: "+10.0/3.0). Aceasta afiseaza rezultatul: Valoarea lui 10/3: 3.33333333333. Afisarea unui numar mare de zecimale este inadecvata de cele mai multe ori. De exemplu, in calcule financiare se afiseaza doua zecimale. Pentru a controla formatarea datelor numerice, utilizam urmatoarea forma a metodei WriteLine(): Console.WriteLine("sir de formatare", arg0, arg1,…, argN); Sirul de formatare contine doua elemente: caractere afisabile si specificatori de format. Specificatorii de format au forma generala *{Nr_arg, width:fmt #}*.

Nr_arg precizeaza numarul argumenului care trebuie afisat. Latimea minima este precizata de valoarea *width* , iar formatul este specificat de *fmt.* De asemenea, simbolul # marcheaza numarul minim de pozitii numerice. *Width, fmt* si # sunt optionale. Pentru afisarea valorilor numerice se pot utiliza urmatorii formati *fmt:*

fmt	Descriere	Format default	Exemple
C or c	Currency	$xx,xxx.xx	$12,345.67
D or d	Decimal	xxxxxxx -xxxxxxx	1234567 -1234567
E or e	Expo-nential	x.xxxxxxe+xxx -x.xxxxxxe+xxx x.xxxxxxe-xxx -x.xxxxxxe-xxx	1.234567e+123 -1.234567e+123 1.234567e-123 -1.234567e-123
F or f	Punct fix	xxxxxxx.xx -xxxxxxx.xx	1234567.89 -1234567.89
N or n	Numeric	xx,xxx.xx -xx,xxx.xx	12,345.67 -12,345.67
X or x	Hexa-decimal		ff (nr. 255)
G or g	General	Se utilizeaza forma cea mai compacta	

Partile componente ale unei aplicatii C#

Un limbaj de programare este compus dintr-o serie de cuvinte cheie care au semnificatii speciale. Un program utilizeaza aceste cuvinte impreuna cu alte cuvinte aditionale si simboluri intr-o forma organizata. Un program C# include urmatoarele: *spatii, cuvinte cheie C#, literali* si *identificatori*.

Limbajul C# contine urmatoarele cuvinte cheie:

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	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
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
while				

Literali

In C# literalii desemneaza valorile fixate, reprezentate intr-un mod accesibil utilizatorului. De exemplu, nr. 35 este un literal. Literalii se mai numesc si constante. Literalii in C# pot fi de orice tip valoric. Constantele de tip caracter sunt incluse intre apostrofuri (exemple: ‘a’, ‘$’, etc.) In ceea ce priveste literalii intregi, tipul fiecarui literal este cel mai mic tip intreg care permite memorarea sa, incepand de la *int.* Un literal intreg poate fi de tip: *int, uint, long sau ulong.* Pentru specificarea unui literal *uint* se adauga un *u* sau *U.* De exemplu 123 este de tip *int* in timp ce 123u este de tip *uint.* In mod analog pentru literali de tip *long* se adauga *l* sau *L* in timp ce pentru literali de tip *ulong* se adauga *ul* sau *UL*.

Literalii in virgula mobila sunt in mod implicit de tip *double.* Daca dorim sa specificam un literal de tip float adaugam *f* sau *F* (de exemplu, 123.4f este de tip float).

Pentru specificarea unui literal de tip *decimal* se adauga sufixul *m* sau *M (ex: 1.43m)*.

Pentru specificarea unui literal hexazecimal se utilizeaza prefixul 0x (ex: 1) c=0xFF; //255 in zecimal, 2) in=0x1a; //26 in zecimal).

Secvente escape pentru caractere

Majoritatea caracterelor pot fi afisate incluzand constantele de tip caracter intre apostrofuri. Exista insa cateva caractere care ridica probleme deosebite precum ghilimelele, apostroful etc., care au semnificatii speciale. Din acest motiv, C# pune la dispozitie secvente escape, care sunt utilizate in locul caracterelor pe care le reprezinta. Secventele escape sunt: \a (alarma), \b (sterge un caracter in urma, backspace), \n (linie noua), \r (revenire la cap de rand), \t (tab orizontal), \v (tab verical), \0 (nul), \’ (apostrof), \” (ghilimele), \\ (backslash).

In C# intalnim si un alt tip de literali: tipul *string*. Un *string*, reprezinta un sir de caractere inclus intre ghilimele (ex: “acesta este un string”). Pe langa caracterele obisnuite, un literal de tip string poate contine mai multe secvente escape.

De asemenea, in C# se pot utiliza literalii *“copie la indigo”.* Un astfel de program incepe cu @, urmat de un sir de ghilimele. Se pot include astfel caractere tab, linie noua, etc fara a utiliza secvente escape. Exista o singura exceptie. Pentru a obtine ghilimelele (“), trebuie utilizate doua caractere unul dupa altul (“”).

```
using System;
class StrDemo
{
   public static void Main()
   {
      Console.WriteLine("Prima linie \nAdoua linie");
      Console.WriteLine("a \t b \t c \nd \t e\t f \t");
   }
}
```

```
using System;
class Indigo
{
   public static void Main()
   {
      Console.WriteLine(@"Acesta este un literal
copie la indigo care
ocupa trei linii");
      Console.WriteLine(@"Alt exemplu
a   b   c
d   e   f");
      Console.WriteLine(@"Putem spune:""hello! """);
        }
}
```

Variabile

O variabila reprezinta o locatie de memorie cu nume, careia ii poate fi atribita o valoare. Valoarea unei variabile poate fi modificata pe parcursul programului. Variabilele sunt declarate printr-o instructiune de forma:

tip nume_var;

unde *tip* reprezinta tipul variabilei, iar *nume_var* numele variabilei. Variabilele trebuie declalarate inainte de a fi folosite, de asemenea tipul variabilei nu poate fi modificat pe parcursul duratei sale de viata. Tipul variabilei determina operatiile permise asupra variabilei.

Dupa declararea variabilei, aceasta trebuie initializata. Initializarea unei variabile poate fi facuta printr-o instructiune de atribuire:

nume_var=val;

unde *val* reprezinta valoarea atribuita variabilei *nume_var* (Exemple: *int i=10; bool f=true; foat fn=12.4f; long o=126L*). De asemenea initialzarea poate fi facuta dinamic, utilizand orice expresie valida la momentul in care variabila este initializata.

```
using System;
class Initdinamica
{
    public static void Main()
    {
        double raza, inaltime, volum;
        double Pi=4*Math.Atan(1);
        raza = 4;
        inaltime = 2;
        volum = Pi * raza * raza * inaltime;
        Console.WriteLine("Valoarea lui pi
este: {0:#.####}", Pi);
       Console.WriteLine("Volumul
cilindrului este:{0:g}", volum);
    }
}
```

Domeniul de valabilitate si durata de viata

La fel ca alte limbaje de programare, C# permite declararea unei variabile in cadrul unui bloc. Un bloc incepe cu o acolada deschisa si se incheie cu o acolada inchisa. Un bloc delimiteaza un spatiu de declarare numit si *domeniu de valabilitate*. De asemenea, acesta determina si *durata de viata* a acelor variabile. O variabla declarata in cadrul unui bloc isi pierde valoarea cand blocul este folosit.

Cele mai importate domenii de valabilitate sunt acelea definite de o clasa si de o metoda.

In cadrul unui bloc variabilele pot fi declarate in orice punct, dar sunt valide numai dupa declaratie.

Un alt aspect interesant care diferentiaza limbajul C# de celelalte limbaje este urmatorul: nici o variabila din interiorul unui domeniu interior nu poate avea acelasi nume cu o variabila declarata intr-un domeniu care il contine.

```
using System;
class NestVar
{
   public static void Main()
   {
      int x = 10;
      if (x == 10)
      {
         int y = 20;
         Console.WriteLine("x si y: " + x + " repectiv " + y);
      }
      Console.WriteLine("x este: " + x);
      //y=20;
    }
}

//Programul de mai jos nu poate fi compilat
using System;
class NestVar
{
   public static void Main()
   {
      int i, j;
      j=10;
      for (i = 0; i < 1; i++)
      {
         int j=1;
         j = j + 1;
         Console.WriteLine("variabila din interiorul blocului
      este: {0}", j);
      }
         Console.WriteLine("variabila din exteriorul blocului
      este: {0}", j);
    }
}
```

Exemplu: Programul de mai jos nu poate fi compilat

```
using System;
class Demo
{
    public static void Main()
    {
        int i;
            for (int j=1; j<10; j++)
            {
                i=j;
                Console.WriteLine("Valoarea variabilei i este: {0}", i);
            }
            Console.WriteLine("Ultima valoare a variabilei i este: {0}", i);
    }
}
```

Numele unei variabile

Numele unei variabile trebuie sa satisfaca urmatoarele reguli:

- primul caracter al unei variabile trebuie sa fie o litera, caracterul underscore “_” sau simbolul *at* “@” ;
- urmatoarele caractere pot fi litere, numere sau caractere underscore;
- nu pot fi utilizate cuvinte cheie drept identificatori.

De-a lungul timpului au fost utilizate diverese conventii pentru numirea variabilelor.

In prezent, platforma .NET utilizeaza urmatoarele conventii: *PascalCase* si *camelCase,* unde *Case* ar trebui sa explice in ce scop este utilizata variabila. Ambele conventii specifica utilizarea unor cuvinte multiple, scrise cu litere mici cu exceptia primei litere care este mare. Pentru camelCase exista o regula suplimentara, si anume primul cuvant este scris cu litera mica.

Exemple:

camelCase: age, firstName, placeOfBirth

PascalCase: Age, FirstName, PlaceOfBirth

Pentru variabilele simple se utilizeaza *camelCase*.

Pentru spatii de nume, clase, metode se utilizeaza *PascalCase*.

Operatori

Un operator reprezinta un simbol care determina compilatorul sa realizeze o anumita operatie matematica sau logica. Limbajul C# ofera patru categorii de operatori: *aritmetici, pe biti, relationali si logici.*

Operatori aritmetici

Limbajul C# defineste urmatorii operatori aritmetici: + (adunare), - (scadere), * (inmultire), / (impartire), % (rest sau operatorul modulo), ++ (incrementare), --(decrementare).

Atunci cand se aplica operatorul / asupra unor intregi, restul este trunchiat, de exemplu: 14/3 este egal cu 4.

O diferenta fata de C si C++ este faptul ca operatorul modulo se poate aplica atat tipurilor intregi cat si celor in virgula mobila (Ex: 14.0 % 3.0=2,

14.2 %3.0=2.2, 14.0%3.1=1.6.

Operatorii unari de incrementare si decrementare functioneaza la fel ca in C. Acestia sunt: x++ (forma postfixata), ++x (forma prefixata), x-- (forma postfixata), --x (forma prefixata)

```
using System;
class Moddemo
{
    public static void Main()
    {
        int cat, rest;
        double doubleCat, doubleRest;
        cat = 14 / 3;
        rest = 14 % 3;
        doubleCat = 14.0 / 3.0;
        doubleRest = 14.0 % 3.0;
        Console.WriteLine("14=3*{0}+{1}", cat, rest);
        Console.WriteLine("14/3={0}",doubleCat);
    Console.WriteLine("14.0 % 3.0={0}", doubleRest);
    }
}
X=10; y=++X;      //y=11
X=10; y=X++;     //y=10
using System;
class Moddemo
{    public static void Main()
   {   double doubleCat;
       doubleCat = 14 / 3;
       Console.WriteLine("14/3={0:.##}", doubleCat);
   }
}
```

Operatori relationali si logici

Operatorii relationali se refera la relatiile de ordine care pot exista intre doua valori, iar operatorii logici desemneaza modalitatile in care se pot asocia valorile de adevar true si false. Operatorii relationali se utilizeaza deseori cu cei logici. Rezultatele intoarse de operatorii relationali si logici sunt de tip bool.

Operatorii relationali sunt urmatorii: == (egal cu), != (diferit de), > (mai mare decat), < (mai mic decat), >= (mai mare sau egal cu), <= (mai mic sau egal cu).

Operatorii logici sunt: & (si), | (sau), ^ (sau exclusiv), || (sau scurtcircuitat), && (si scurtcircuitat), ! (non).

In cazul operatorilor relationali <,>,<=,>= operanzii trebuie sa apartina tipurilor pe care este definita o relatie de ordine.

In cazul operatorilor logici, operanzii trebuie sa fie de tip bool. In tabelul de mai jos sunt precizate rezultatele opearatiilor logice. S-a folosit conventia: 0=false si 1=true.

p	q	p&q	p|q	p^q	!p
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

Singura diferenta dintre operatorii & si | si respectiv operatorii scurtcircuitati && si || este faptul ca operatorii obisnuiti evalueaza intodeauna ambii operanzi in timp ce vaiantele scurtcircuitate evalueaza al doilea operand doar daca este necesar.

Exemplu:

```
using System;
class Scurtcirc
{
   public static void Main()
   {
      int n, d;
      n = 10;
      d = 2;
      if ((d !=0) && ((n%d)==0))
      Console.WriteLine(d +" este divizor a lui " +n);
      d = 0;
      if ((d != 0) && ((n % d) == 0))
       Console.WriteLine(d + " este divizor a lui " + n);
   }
}
```

Operatori pe biti

Operatorii pe biti realizeaza operatii asupra asupra unuia sau mai multor biti dintr-o valoare. Operatorii pe biti sunt urmatorii: & (si pe biti), | (sau pe biti), ^ (sau exclusiv pe biti), ~ (complementare pe biti), >>(deplasare pe biti la dreapta), <<(deplasare pe biti la stanga.

Exemple:

3|4=7; 5&7=5, 4^4=0, ~2=-3, 10>>1=5, 10<<1=20.

Precedenta operatorilor

1 Primary operators	() . [ ] ++ x -- x
2 Unary	+ - ! ~
3 Multiplicative	* / %
4 Additive	+ -
5 Shift	<< >>
6 Relational	< > <= >=
7 Equality	== !=
8 Logical AND	&
9 Logical XOR	^
10 Logical OR	|
11 Conditional AND	&&
12 Conditional OR	||
13 Conditional	?:
14 Assignment	= *= /= % = += -= <<= >>= &= ^= |=
15 Increment and decrement	x++ x--

Conversia tipurilor de date

O practica frecventa in programare o reprezinta atribuirea valorii unei variabile de un anumit tip unei alte variabile avand tip diferit.

Exemplu: *int i=10; float f; f=i;*

Conversii implicite

Daca in atribuire sunt implicate tipuri compatibile atunci valoarea din partea dreapta este convertita automat la tipul din partea stanga.

Atunci cand un tip de date este atribuit unui alt tip de variabila, se efectueaza o conversie implicita (automata) de tip daca: cele doua tipuri sunt compatibile si tipul destinatie este mai cuprinzator decat tipul sursa. Daca cele doua conditii sunt indeplinite atunci are loc o conversie prin largire. (Ex: long in double se face prin conversie automata, in schimb double in long nu se poate realiza automat).

De asemenea nu exista conversii automate intre *decimal si double* sau *decimal si float* si nici intre tipurile numerice si *bool.*

Conversii explicite

In multe situatii (daca exista vreo legatura intre tipul sursa si tipul destinatie) se pot realiza conversii explicite de tip, utilizand un cast. Un cast este o directiva catre compilator de a converti un anumit tip in altul. Forma generala este: *(tip_tinta) expr;*

Exemple: double x,y; int z;… z= (int) (x/y);

int i=230; byte b; ... b=(byte)i;

Insa de fiecare data, responsabilitatea este cea a programatorului daca rezultatul obtinut este cel convenabil. Pentru a controla rezultatul se pot utiliza comenzile:

checked (expresie) ; sau unchecked (expresie)

Exemplu: Conversii explicite

```
using System;
class Demo
{
    public static void Main()
    {
        byte destinationVar;
        int sourceVar = 257;
        destinationVar = (byte)sourceVar;
     //destinationVar = checked((byte)sourceVar);
        Console.WriteLine("sourceVar val: {0}", sourceVar);
        Console.WriteLine("destinationVar val: {0}", destinationVar);
    }
}
```

Studiul expresiilor

O expresie este o combinatie valida de literali, identificatori, operatori.

In cadrul unei expresii este posibil sa amestecam doua sau mai multe tipuri de date atat timp cat acestea sunt compatibile intre ele. Conversiile se realizeaza utilizand *regulile de de promovare* a tipurilor din C#. Iata algoritmul definit de aceste reguli pentru operatii binare:

- Daca un operand este decimal atunci celalalt este promovat la decimal (cu exceptia cazului in care este de tipul double sau float, caz in care apare o eroare)
- Altfel daca un operand este *double* atunci celalalt este promovat la *double.*
- Altfel daca un operand este *float* atunci celalalt este promovat la *float.*
- Altfel daca un operand este *ulong* atunci celalalt este promovat la *ulong,* numai daca nu este de tip sbyte, short, int sau long, caz in care apare o eroare.
- Altfel daca un operand este *long* atunci celalalt este promovat la *long.*
- Altfel daca un operand este *uint, iar al doilea este de tip* sbyte, short sau int, ambii sunt promovati la long.
- Altfel daca un operand este *uint* atunci celalalt este promovat la *uint.*
- Altfel ambii operanzi sunt promovati la *int.*

Observatie: Rezultatul unei expresii nu poate fi un tip inferior lui *int.*