




Modulul 3


Tipuri de instrucțiuni

itacad
you@technology

Overview

- ▶ Tipuri definite de utilizator
- ▶ Conversia tipurilor
- ▶ Tipuri de instrucțiuni





2

În acest modul vom continua discuția despre tipurile de valori și vom vedea care sunt tipurile definite de utilizator și cum pot fi folosite. De asemenea, vom afla cum se poate face conversia de la un tip la altul.

În final, vom prezenta instrucțiunile folosite în C#, sintaxa lor și modul de utilizare.

La finalul acestui modul, studenții vor putea să:

- Creeze și să folosească propriile tipuri de date
- Convertească o variabilă de un tip în alt tip
- Descrie diferitele tipuri de instrucțiuni
- Folosească instrucțiuni de salt
- Folosească instrucțiuni de selecție
- Folosească instrucțiuni de iterație

Tipuri predefinite și tipuri definite de utilizator

Tipuri predefinite


- ▶ int
- ▶ float
- ▶ double
- ▶ bool
- ▶ char


Tipuri definite de utilizator

- ▶ enum
- ▶ struct


3

▪ **Tipurile valoare** includ tipurile predefinite și tipurile definite de utilizator. Diferența dintre cele două este minimală având în vedere că variabilele, indiferent de tip, sunt folosite în același fel. Toate tipurile predefinite, care sunt de fapt tipurile valoare, conțin date, ceea ce înseamnă că nu pot avea valoarea **null**.





Tipuri definite de user - Enumerația



- ▶ Definirea unei enumerații


```
enum Color {Red, Green, Blue};
```
- ▶ Utilizarea unei enumerații


```
Color colorPalette = Color.Red;
```
- ▶ Afișarea unei variabile de tip enumerație


```
Console.WriteLine("{0}", colorPalette);
```

4

➤ Enumerațiile reprezintă o modalitate eficientă de a defini un set de constante ce pot fi atribuite unei variabile.

➤ **Declararea unei enumerații**

- Pentru a declara o enumerație se utilizează cuvântul cheie **enum** urmat de numele variabilei și valorile inițiale
- De exemplu, enumerația următoare definește trei constante întregi, numite valorile enumerației:

enum Color {Red, Green, Blue}

- Valorile enumerației încep întotdeauna de la 0. Prin urmare avem: Red corespunde valorii 0, Green valorii 1, iar **Blue** valorii 2

➤ **Utilizarea unei enumerații**

- O variabilă de tip **Color** se poate declara în felul următor:

Color colorPalette;
- Setarea valorii variabilei declarate:

colorPalette = Color.Red;
- Operația anterioară se poate realiza și prin castare din **int** în **Color**

colorPalette = (Color)0;

➤ **Afișarea unei enumerații**

- Pentru a afișa variabila declarată mai sus se procedează astfel:

Console.WriteLine("{0}", colorPalette);

Tipuri definite de user - Structura



► Definirea unei structuri

```
public struct Employee {
    public string firstName;
    public int age;
}
```

► Utilizarea unei structuri

```
Employee companyEmployee;
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;
```

5

► Structura reprezintă un tip valoare utilizat, în general, pentru a encapsula mai multe variabile ce au o legătură între ele. Deoarece structurile nu sunt alocate în *heap* este mai puțină muncă din partea **garbage collector**-ului decât în cazul creării unei clase.

► Definirea unei structuri

- O structură se poate folosi pentru a grupa tipuri predefinite ca în exemplul următor:

```
public struct Employee
{
    public string firstName;
    public int age;
}
```

- Acest cod crează o structură **Employee** care are două elemente: **first name** și **age**.

► Utilizarea unei structuri

- Declararea unei variabile de tip **Employee**:
Employee companyEmployee;
- Setarea valorii variabilei declarate mai sus:
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;

Conversia implicită de tipuri



► Conversie **int** în **long**

```
using System;
class Test
{
    static void Main ( )
    {
        int intValue = 125;
        long longValue = intValue;
        Console.WriteLine("(long){0} = {1}", intValue,
            longValue);
    }
}
```

► Conversia implicită nu poate eșua

► Conversia implicită este acea conversie care este sigură, prin care nu se pierd informații (valoarea variabilei rămâne aceeași)

► Un exemplu de conversie implicită este următorul:

```
using System;
class Test
{
    static void Main( )
    {
        int intValue = 123;
        long longValue = intValue;
        Console.WriteLine("(long) {0} = {1}", intValue, longValue);
    }
}
```

► Deoarece valoarea maximă, respectiv minimă, a unei variabile de tip **int** se încadrează în intervalul de valori **long**, întotdeauna conversia dintr-un **int** în **long** este sigură

Conversia explicită de tipuri



► Conversie **long** în **int** prin cast

```
using System;
class Test
{
    static void Main ( )
    {
        long longValue = Int64.MaxValue;
        int intValue = (int) longValue;
        Console.WriteLine("(long){0}={1}",intValue,
            longValue);
    }
}
```

► Conversia explicită poate eșua

- Poate modifica valoarea variabilei

7

➤ O altă formă de conversie este cea explicită (prin care se specifică tipul), care se mai numește și castare.


➤ Un exemplu de cast este următorul:

```
using System;
class Test
{
    static void Main( )
    {
        long longValue = Int64.MaxValue;
        int intValue = (int) longValue;
        Console.WriteLine("(int) {0} = {1}", longValue,intValue);
    }
}
```

➤ Deoarece are loc o depășire de limită ceea ce se va afișa este:

(int) 9223372036854775807 = -1

➤ Vom vedea in modulul urmator ce trebuie sa facem pentru a evita o astfel de situatie

 itacad
you@technology

Microsoft
.NET


Concepte de bază – Instrucțiuni bloc

► Un bloc de instrucțiuni este delimitat de acolade


```
{
    .....
}
```

► Un bloc copil și blocul părinte nu pot avea definite variabile cu același nume; blocuri alăturate, însă, da

```
{
    int i;
    {
        int i;
    }
}
```



```
{
    int i;
}
...
{
    int i;
}
```



8

Un program este alcătuit dintr-o înlănțuire de instrucțiuni. În timpul rulării, aceste instrucțiuni sunt executate una după alta, așa cum apar în program, de la stânga la dreapta și de sus în jos.

În momentul dezvoltării aplicațiilor C#, instrucțiunile sunt grupate așa cum se întâmplă și în alte limbaje de programare. Acest lucru se realizează folosind sintaxa asemănătoare limbajelor precum C, C++ și Java, adică închizând grupul de instrucțiuni între acolade: { și }.

Un set de instrucțiuni între acolade este numit bloc. Un bloc poate să fie format dintr-una sau mai multe instrucțiuni și poate chiar să includă un alt bloc (nesting).

Fiecare bloc definește un context. O variabilă declarată într-un bloc este numită o variabilă locală. O astfel de variabilă este definită din locul unde a fost declarată până la închiderea acoladei ce marchează sfârșitul blocului.

În cazul existenței mai multor blocuri nested, este recomandată definirea variabilelor în cel mai specific bloc posibil, acest lucru făcând programul mai clar.


În C# nu este posibilă declararea unor variabile într-un bloc intern cu același nume ca variabilele din blocul părinte. De exemplu următorul cod va genera eroare:


```
int i;
{
    int i; //Eroare: i a fost deja definită în blocul părinte
}
```

Totuși, variabile cu același nume pot fi definite în blocuri alăturate


```
{    int i;}
...
{    int i;}
```

Variabilele se pot declara oriunde într-un bloc. Se recomandă inițializarea variabilelor la momentul declarării.

 itacad
you@technology

 Microsoft
.NET

Tipuri de instrucțiuni



Instrucțiuni de selecție
if și switch

Instrucțiuni de iterație
while, do, for și foreach

Instrucțiuni de salt
goto, break și continue

9

Pe măsură ce complexitatea problemelor rezolvate de programe crește, la fel crește și complexitatea logicii programului. În consecință, cursul programului trebuie să fie controlat într-un mod cât mai structurat. Acest lucru se poate realiza folosind instrucțiuni de nivel mai înalt. Acestea se grupează în 3 categorii:

- **Instrucțiuni de selecție**


Instrucțiunile **if** și **switch** intră în această categorie. Ele permit luarea unei decizii pe baza evaluării unei expresii și executarea selectivă a instrucțiunilor următoare în funcție de decizia luată

- **Instrucțiuni de iterație**

Instrucțiunile **while**, **do**, **for** și **foreach** se execută repetitiv cât timp o anumită condiție este îndeplinită. Mai sunt numite și instrucțiuni de ciclare. Fiecare dintre aceste instrucțiuni este potrivită pentru un anumit tip de iterație


- **Instrucțiuni de salt**

Instrucțiunile **goto**, **break** și **continue** sunt folosite pentru transferul necondiționat al controlului către o altă instrucțiune.

 itacad
you@technology

Microsoft
.NET

Instrucțiunea if



- ▶ Sintaxă:


```
if (expresie booleană)
    bloc instrucțiuni;
else
    bloc instrucțiuni;
```
- ▶ Folosirea în cascadă


```
if (i < 0)
    Console.Write("Negativ");
else if (i == 0)
    Console.Write("Zero");
else
    Console.Write("Pozitiv");
```

10

Instrucțiunea **if** este principala instrucțiune de luare de decizii. Poate fi cuplată cu o clauză **else** opțională, așa cum se vede mai sus.

Instrucțiunea **if** evaluează o expresie booleană pentru a determina ce trebuie executat mai departe. Dacă valoarea expresiei este **true**, controlul este dat blocului de instrucțiuni ce urmează după **if**. Dacă valoarea este **false**, controlul este dat blocului **else**, dacă există.

Atenție: Deși sintaxa instrucțiunii este aceeași ca în limbaje precum C sau C++, există diferențe. De exemplu, în C, instrucțiunea **if** realizează automat conversia unui întreg (int) la valoare Booleană, 0 fiind echivalentul lui **false**, orice alt număr fiind **true**. În C#, acest lucru nu este valabil.

În cazul în care blocul de instrucțiuni ce urmează instrucțiunii **if** sau clauzei **else** este format dintr-o singură instrucțiune, acoladele sunt opționale. Astfel, următorul cod este corect:

```
if (i < 0)
    Console.Write("Negativ");
```

Totuși, pentru a avea cod mai clar de înțeles și mai ordonat, se recomandă folosirea acoladelor pentru a delimita blocurile de instrucțiuni și în acest caz:

```
if (i < 0)
{
    Console.Write("Negativ");
}
```

În situația în care există mai mult de două variante posibile la un anumit punct, se pot folosi instrucțiuni **if** cascade.

Instrucțiunea switch



- Asemănătoare unor if-uri cascade, dar mai elegant și mai ordonat

```
switch (masina)
{
    case "Ferrari F50":
        Console.WriteLine("Nu ai nevoie de acest curs");
        break;
    case "bicicleta":
        Console.WriteLine("Spor la pedalat");
        break;
    default:
        Console.WriteLine("RATB???");
        break;
}
```


11

Instrucțiunea **switch** oferă un mecanism elegant pentru tratarea condițiilor complexe care, altfel, ar necesita instrucțiuni **if** folosite în cascadă. Este alcătuită din mai multe blocuri **case**, fiecare dintre acestea specificând o singură constantă și eticheta **case** asociată. Nu este posibilă asocierea a mai multe constante aceleiași etichete **case**.

Pentru fiecare bloc **case** urmează apoi un bloc de instrucțiuni ce se va termina, în cele mai multe cazuri, cu instrucțiunea **break**. În interiorul blocului se pot face declarații de variabile sau constante, acestea fiind definite de la declarație până la sfârșitul blocului **switch**.


Execuția unei instrucțiuni **switch**:

- 1) Dacă una dintre constantele specificate în etichetele **case** este egală cu valoarea expresiei pe care se face **switch**, controlul este transferat către blocul de instrucțiuni asociat.
- 2) Dacă nici una dintre constantele specificate nu este egală cu valoarea expresiei și instrucțiunea **switch** conține o etichetă **default**, controlul este transferat către lista de instrucțiuni ce urmează etichetei **default**.
- 3) Dacă nici una dintre constantele specificate nu este egală cu valoarea expresiei pe care se face **switch** și instrucțiunea **switch** nu are definită o etichetă **default**, controlul este transferat la finalul instrucțiunii **switch**.


itacad
 you@technology

 Microsoft
.NET

Identificați greșeala



- 1)

if number % 2 == 0 ...
- 2)

if (age < 0) || (age > 100) ...
- 3)

if (minute == 60);
minute = 0;
- 4)

switch (carteDeJoc)
{
case Set.Trefla:
Console.Write("bonus");
default:
Console.Write("incerca alta carte");
}

12

- 1) Expresia nu se află între paranteze
- 2) Expresia întreagă nu se află între paranteze
- 3) Instrucțiunea **if** nu execută nimic, pentru ca s-a pus caracterul ';' la finalul liniei
- 4) Blocul de instrucțiuni nu are instrucțiunea **break**

Instrucțiuni de iterație - while



- ▶ Execuție bazată pe valoarea unei expresii booleane
- ▶ Expresia este evaluată la începutul fiecărei iterații
- ▶ Iterația se execută cât timp expresia este adevărată

```
int i = 0;
while (i < 10)
{
    Console.Write(i);
    i++;
}
```

0 1 2 3 4 5 6 7 8 9

13

Instrucțiunea **while** este cea mai simplă dintre toate instrucțiunile de iterație. Instrucțiunea va executa repetitiv un bloc de instrucțiuni **while** (cât timp) expresia booleană asociată este adevărată. Este important faptul că expresia evaluată de **while** trebuie neapărat să fie booleană, din moment ce C# nu suportă conversie implicită de la un întreg la o valoare booleană.

Execuția instrucțiunii

- 1) Expresia booleană ce controlează instrucțiunea while este evaluată;
- 2) Dacă expresia este evaluată la **true**, controlul este transferat blocului de instrucțiuni atașat instrucțiunii while. La finalul blocului, controlul este automat trimis la începutul instrucțiunii while și expresia este reevaluată;
- 3) Dacă expresia este evaluată la **false**, controlul este transferat la finalul instrucțiunii while și nu se mai execută instrucțiunile din bloc.

Expresia ce controlează instrucțiunea este testată la începutul buclei while. Este posibil ca instrucțiunile din bloc să nu execute nici o dată.

Obs: după cum se observă și în slide, foarte des, în instrucțiunea **while**, există o zonă de actualizare, în cazul de față incrementarea contorului *i*. Această parte posibil să ridice probleme dacă este uitată, aici ar duce la ciclare la infinit.

Instrucțiuni de iterație - do



- ▶ Execuție bazată pe valoarea unei expresii booleane
- ▶ Expresia este evaluată la sfârșitul fiecărei iterații
- ▶ Iterația se execută cât timp expresia este adevărată

```
int i = 0;
do
{
    Console.Write(i.ToString() + " ");
    i++;
} while (i < 10);
```

0 1 2 3 4 5 6 7 8 9

14

Instrucțiunea **do** este întotdeauna cuplată cu o instrucțiune **while**. Este similar cu o instrucțiunea **while**, cu excepția faptului că evaluarea expresiei, pentru a decide dacă se va executa un ciclu sau se va ieși, se face la finalul ciclului, nu la început. Acest lucru înseamnă că instrucțiunea **do** va itera cel puțin o dată.

Execuția instrucțiunii:

- 1) Controlul este transferat blocului de instrucțiuni
- 2) Când se ajunge la sfârșitul blocului, se evaluează expresia booleană
- 3) Dacă rezultatul este **true**, controlul este transferat la începutul instrucțiunii **do**
- 4) Dacă rezultatul este **false**, controlul este transferat la finalul instrucțiunii **do**

Instrucțiuni de iterație - for



- ▶ Instrucțiune foarte flexibilă
- ▶ Sintaxa:

```
for (int i = 0; i < 10; i++)
{
    Console.Write(i);
}
```

- ▶ Sunt definite 3 zone:
 - ▶ Operații executate la intrarea în buclă
 - ▶ int i = 0;
 - ▶ Condiții de ieșire
 - ▶ i < 10;
 - ▶ Operații executate la fiecare pas al iterației
 - ▶ i ++;

15

Atunci când se folosește instrucțiunea **while**, programatorii uită adesea să actualizeze variabila de control. Acest lucru se întâmplă pentru că atenția programatorului este concentrată pe corpul instrucțiunii **while** și nu pe zona de actualizare.

Aceste erori pot fi minimizate folosind instrucțiunea **for**. Instrucțiunea **for** are partea de actualizare la începutul buclei, unde este mai greu de omis.

Sintaxa pentru instrucțiunea **for** este:

```
for (initializare; conditie; actualizare)
{
    bloc instructiuni
}
```

Obs: Deși codul de actualizare este scris înainte de blocul de instrucțiuni, în execuție este rulat întâi blocul de instrucțiuni și apoi codul din zona de actualizare.

Ca și pentru celelalte instrucțiuni de ciclare, expresia din zona de condiție trebuie să fie o expresie booleană. Dacă ea se evaluează la **true**, ciclul este continuat (condiție de continuare, nu de terminare).

Declararea variabilelor

O diferență subtilă între instrucțiunea **while** și **for** este că, în cazul celei de-a doua, o variabilă definită în zona de inițializare este definită doar până la terminarea instrucțiunii.

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
Console.WriteLine(i);    //Eroare: i nu mai este definit aici
```

Atenție: în momentul declarării unei variabile în zona de inițializare a instrucțiunii, ea nu trebuie să mai fie definită în blocul parinte.

```
int i;
for (int i = 0; i < 10; i++)    //Eroare: i este deja declarat
```



Instrucțiuni de iterație - foreach



- Clasă specială – clasa Collection
- O serie de clase moștenesc aceasta clasă
- foreach permite iterarea pe fiecare element al unei clase de acest tip

```
ArrayList numbers = new ArrayList();
for (int i = 0; i < 10; i++)
{
    numbers.Add(i);
}

foreach (int number in numbers)
{
    Console.WriteLine(number);
}
```

16

Colecțiile sunt entități software ale căror rol este de a colecta alte entități software, așa cum o clasă poate fi considerată o colecție de camere.

Platforma Microsoft .NET oferă o clasă colecție simplă numită **ArrayList**. Aceasta poate fi folosită pentru a crea variabile de tip colecție și pentru a adăuga elemente la colecție. Exemplu:

```
using System.Collections;
...
ArrayList numbers = new ArrayList();
for (int i = 0; i < 10; i++)
{
    numbers.Add(i);
}
```


Codul pentru a accesa elementele unei colecții și pentru a le afișa este:

```
for (int i = 0; i < numbers.Count; i++)
{
    int number = (int)numbers[i];
    Console.WriteLine(number);
}
```

După cum observați, metoda de a itera pe elementele unei colecții nu este deloc trivială. Pentru a rezolva această problemă, C# oferă instrucțiunea **foreach**, care permite iterarea pe o colecție printr-o singură instrucțiune.


```
foreach (int number in numbers)
{
    Console.WriteLine(number);
}
```

Elementele unei colecții nu pot fi modificate în timpul iterației pentru că ele sunt **read-only**

 itacad
you@technology

Microsoft
.NET

Identificați greșeala



- 1)

```
for (int i = 0, i < 10, i++)
    Console.Write(i);
```
- 2)

```
int i = 0;
while (i < 10)
    Console.Write(i);
```
- 3)

```
for (int i = 0; i >= 10; i++)
    Console.Write(i);
```
- 4)

```
do
    ...
    string line = Console.ReadLine();
    guess = int.Parse(line);
while (guess != answer);
```

17

- 1) Elementele instrucțiunii **for** trebuie să fie separate prin ';' și nu prin ','
- 2) Zona de update a contorului *i* a fost uitată, se va cicla la infinit
- 3) Instrucțiunea **for** nu se va executa, condiția de continuare nu este respectată niciodată
- 4) Instrucțiunile incluse în instrucțiunea **do** trebuie închise într-un bloc, între acolade

Instrucțiuni de salt – break



► Instrucțiunea **break** întrerupe o iterație

```
while(a < 10)
{
    Console.WriteLine ("Begin");
    while(b < 100)
    {
        Console.WriteLine ("Before break");
        break;
        Console.WriteLine ("After break");
    }
    Console.WriteLine ("End");
}
```

18

- Instrucțiunea **break** trimite controlul programului la ieșirea din cea mai apropiată instrucțiune de tip **switch**, **while**, **do**, **for** sau **foreach** care o conține.
- Dacă instrucțiunea nu se află într-un bloc de tip **switch**, **while**, **do**, **for** sau **foreach**, va fi generată o eroare la compilare.
- În cazul în care instrucțiunea se află într-un bloc imbricat într-un alt bloc de tipul enumerat mai sus, **break** se aplică doar pentru cel mai aproape bloc:

```
while(a < 10)
{
    Console.WriteLine ("Begin");
    while(b < 100)
    {
        Console.WriteLine ("Before break");
        break;
        Console.WriteLine ("After break");
    }
    Console.WriteLine ("End");
}
```

- În exemplul de mai sus, se vor afișa următoarele mesaje:
Begin
Before break
End

Instrucțiuni de salt – continue



- Instrucțiunea **continue** sare la iterația următoare

```
int i = 0;
while (i<10)
{
    i++;
    if (i==2)
        continue;
    Console.Write(i);
}
```

19

- Instrucțiunea **continue** începe o nouă iterație pentru cea mai apropiată instrucțiune de tipul while, do, for sau foreach
- Dacă instrucțiunea nu este cuprinsă într-un astfel de bloc, va apărea o eroare la compilare.
- La fel ca la break, dacă instrucțiunea continue se află într-un bloc nested, ea va începe o nouă iterație doar pentru cel mai apropiat bloc.

```
int i = 0;
while (i<10)
{
    i++;
    if (i==2)
        continue;

    Console.Write(i);
}
```

- După execuția exemplului de mai sus se va afișa: 1,3,4,5,6,7,8,9,10.
La prima iterație i=0, dar după ce intră în while, vom avea i=1. Această valoare va fi afișată.
La a doua iterație i=1, va fi incrementat, condiția din if este adevărată, ceea ce înseamnă că va fi executată instrucțiunea continue. Aceasta transferă controlul la începutul instrucțiunii while.
Execuția programului continuă până când condiția i<10 devine falsă.



Sumar



- ▶ Tipuri definite de utilizator
- ▶ Conversia tipurilor
- ▶ Tipuri de instrucțiuni

20

➤ Întrebări:

- Poate o variabilă tip valoare să fie **null**?
- Se pot folosi variabile neinițializate? De ce?
- Se pierde exactitatea valorii variabilei în conversia implicită?