

Contents

CAPITOLUL 1	2
Aplicații de tip Consola (Console Application)	2
1. Introducere	2
1.1 Sintaxa limbajului C#	2
1.1.1 Nume	2
1.1.2 Identificatori	3
1.1.3 Cuvinte cheie	3
1.1.4 Constante	4
1.1.5 Variabile	4
1.1.6 Expresii și operatori	4
1.1.7 Tipuri de date	8
1.1.8 Tipuri de conversii	8
1.1.9 Opțiuni de afișare și scriere	10
1.1.10 Comentarii.....	12
2. Instalarea mediului de dezvoltare pentru realizarea aplicațiilor C#	13
2.1 Crearea unei aplicații de tip “Console Application”	13
2.1.1 Structura unei aplicații de tip “Console Application”	14
2.1.2 Adăugarea codului în aplicații de tip “Console Application”	15
2.1.3 Compilarea și rularea aplicațiilor de tip “Console Application”	16
3. Exerciții	19

CAPITOLUL 1

Aplicații de tip Consola (Console Application)

Acest capitol prezintă câteva concepte introductive ale limbajului de programare C#, utilizând mediul de dezvoltare Microsoft Visual Studio. Sunt prezentate, de asemenea, câteva aplicații de tip Console Application.

1. Introducere

C# este un limbaj orientat spre obiect, elegant și sigur, care permite dezvoltatorilor să construiască o varietate de aplicații sigure și robuste care rulează pe .NET Framework. Puteți utiliza C# pentru a crea aplicații client Windows, servicii Web XML, componente distribuite, aplicații client-server, aplicații de bază de date și multe altele. Visual C# oferă un editor de coduri avansate, designeri convenabili pentru interfața cu utilizatorii, un debugger integrat și multe alte instrumente pentru a ușura dezvoltarea aplicațiilor bazate pe limbajul C# și pe .NET Framework.

1.1 Sintaxa limbajului C#

Limbajul C# are un alfabet format din litere mari și mici, cifre și alte semne. Pentru scrierea programelor (nume, expresii, separatori, delimitatori și comentarii) se folosesc "simboluri" cu semnificații lexicale.

1.1.1 Nume

Numele reprezintă o succesiune de caractere dat unei variabile, clase, metode, interfete etc și care are câteva reguli de bază:

- Numele începe cu majusculă în cazul numelor pentru clase, metode, proprietăți, enumerări, interfețe, spații de nume;
- Dacă numele variabilelor este compus din mai multe cuvinte, primul începe cu literă mică, urmând ca celelalte nume să fie cu majusculă;
- Numele trebuie să înceapă cu o literă sau cu unul dintre caracterele "_" și "@";
- Numele care reprezintă cuvinte cheie nu pot fi folosite în alt scop decât acela pentru care au fost definite;
- Cuvintele cheie pot fi folosite în alt scop numai dacă sunt precedate de "@";
- Două nume sunt distincte dacă diferă prin cel puțin un caracter;

1.1.2 Identificatori

Identificatorii sunt utilizați pentru a indica variabile, tipuri de date, constante simbolice sau metode. Lungimea maxima a acestora nu poate să depășească 31 de caractere. Primul caracter al unui identificator trebuie să fie o literă sau o subliniere.

1.1.3 Cuvinte cheie

Cuvintele cheie sunt niște identificatori predefiniți, care au o semnificație specială pentru compilator. În tabelul de mai jos sunt prezentate cuvintele cheie existente în C#(Tab.1).

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
volatile	while			

Tab.1 Cuvinte cheie în C#

De asemenea, mai există un set de cuvinte contextuale, care au semnificații specifice, prezentate în tabelul 2.

ascending	by	descending	equals	from
get	group	into	join	let

on	orderby	partial	select	set
value	where	yield		

Tab.2 Cuvinte cheie în C#

În general se păstrează semnificația fiecăruia pentru a evita neclaritățile care pot să apară, însă, dacă dorim să dăm o altă semnificație cuvintelor se poate adăuga ca prefix simbolul “@”.

1.1.4 Constante

O constantă reprezintă o valoare care nu se schimbă și care se folosește în repetate rânduri în codul aplicației. În general, numele constantei este sugestiv în a înțelege scopul acesteia și se poate modifica într-un singur loc. Acestea trebuie inițializate în momentul declarării lor și au un tip, un nume și o valoare.

Exista două modalități de declarare a constantelor, și anume:

- Utilizând modificatorul **const**, acestea sunt constante la compilare și trebuie inițializate la declararea lor;
- Folosind modificatorul **readonly**, acestea sunt constante la execuție și inițializarea acestora se face la runtime.

1.1.5 Variabile

O variabilă este o locație de memorie cu nume, căreia îi poate fi atribuită o valoare. Aceasta face distincție între minuscule și majuscule (ex: *sum* și *Sum* sunt două variabile distincte). Variabilele sunt declarate printr-o instrucțiune de forma: tip nume_var; (*tip* reprezintă tipul variabilei, operațiile premise asupra variabilei, iar *nume_var* numele acesteia). Variabilele trebuie declarate înainte de a fi folosite, tipul variabilei neputând fi modificat ulterior.

Inițializarea unei variabile, după declararea ei se poate face astfel:

```
nume_var = valoare;
```

Valoarea atribuită variabilei poate fi : a = 14, b = true, etc.

Observație!: Declararea și inițializarea unei variabile se poate face în același timp: int a = 14, bool b = true, etc.

1.1.6 Expresii și operatori

Expresie = operatori (acțiunea care se efectuează) + operanzi (execută operația)

Există trei categorii de operatori:

- **Unari** – acționează asupra unui singur operand;
- **Binary** – acționează între doi operanzi;
- **Ternari** – acționează asupra a trei operanzi.

În C# sunt definiți mai mulți operatori. În cazul în care într-o expresie nu intervin paranteze, operațiile se execută conform priorității operatorilor. În cazul în care sunt mai mulți operatori cu

aceeași prioritate, evaluarea expresiei se realizează de la stânga la dreapta. În tabelul alăturat prioritatea descrește de la 0 la 13.

Operatorii sunt utilizați pentru a specifica care operații ar trebui să fie efectuate într-o expresie (Tab. 3).

Prioritate	Tip	Operatori	Asociativitate
0	Primar	() [] f() . x++ x-- new typeof sizeof checked unchecked ->	->
1	Unar	+ - ! ~ ++x --x (tip) true false & sizeof	->
2	Multiplicativ	* / %	->
3	Aditiv	+ -	->
4	De delasare	<< >>	->
5	Relational	< > <= >= is as	->
6	De egalitate	== !=	->
7	And (SI) logic	&	->
8	XOR (SAU exclusive) logic	^	->
9	OR (SAU) logic		->
10	AND (SI) conditional	&&	->
11	OR (SAU) conditional		->
12	Conditional (ternar)	?:	<-
13	Atribuire simpla Atribuire compusa	= *= /= %= += -= ^= &= <<= >>= =	<-

Tab.3 Tabelul de priorități



Operatori de egalitate:

=	Atribuire o valoare
==	Teste pentru egalitate



Operatori de incrementare și decrementare:

i++	Incrementarea variabilei i cu 1
i+=n	Incrementarea variabilei i cu n i=i+n
i*=n	Multiplicarea variabilei i cu n
i-=n	Scaderea variabilei i cu n



Operatori matematici și logici:

+	Adunare
---	---------

-	Scadere
*	Inmultire
/	Impartire
%	Modul
&	ORI (Logical AND)
^	SAU-EXCLUSIV (Logical XOR)
	SAU (Logical OR)

Evaluarea expresiei aritmetice este efectuată din partea stângă spre dreapta, luând în considerare ordinea evaluării periodice.



Operatori logici:

&&	SI (Conditional AND)
	SAU (Conditional OR)
!	Inegalitate (Conditional NOT)

Operatorul **&&** are o prioritate mai mare decât operatorul **||** și o prioritate mai mică decât operatorii de comparare. În operațiile **&&** și **||**, termenii utilizați reprezintă valori de tip *bool*.

Observație!: Regula de la C, în care o valoare numerică diferită de 0 este adevărată și o valoare egală cu 0 este falsă, nu este validă în C#.



Operații relationale:

==	Verifica egalitatea
!=	Verifica inegalitatea
>	Mai mare decat
<	Mai mic decat
>=	Mai mare sau egal decat
<=	Mai mic sau egal decat

Exemplu 1.) Să se verifice dacă un număr citit de la tastatură este pozitiv sau negativ, utilizând operatorul ternar(?).

```

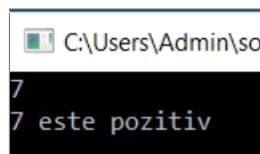
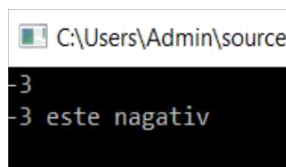
class Program
{
    static void Main(string[] args)
    {
        int a;
        string rez;
        a = int.Parse(Console.ReadLine());
        Console.Write(a);
        rez = (a > 0) ? "este pozitiv" : "este negativ";
        Console.Write(rez);
        Console.ReadLine();
    }
}

```

Nota: Sintaxa operatorului ? este: **(condiție) ? (expr_1) : (expr_2)**, se evaluează condiția, dacă ea este adevărată se execută expr_1, altfel se execută expr_2.

Observație!: `int.Parse` convertește un șir de caractere în integer (int).

În urma rulării programului obținem:



Exemplu2.) Să se verifice dacă un număr este par sau impar, folosind operatorul %.

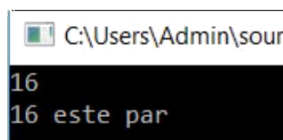
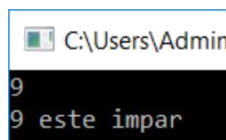
```

class Program
{
    static void Main(string[] args)
    {
        int a;
        a = Convert.ToInt32(Console.ReadLine());
        if (a % 2 == 0) Console.WriteLine("{0} este par", a);
        else Console.WriteLine("{0} este impar", a);
        Console.ReadLine();
    }
}

```

Observație!: `Convert.ToInt32` este folosit pentru a face conversie dintr-un șir de caractere introdus de la tastatură în tipul int.

În urma rulării programului obținem :



1.1.7 Tipuri de date

În limbajul C# există două categorii de tipuri generale de date, care sunt derivate din tipul **System.Object**. Acestea sunt:

- tipuri valorice (în cazul acestor tipuri de date, variabilele conțin valoarea implicită specifică tipului):
 - tipul simplu predefinit: byte, char, int, float etc.;
 - tipul enumerare – enum;
 - tipul structură – struct;
- tipuri referință (în cazul acestor tipuri de date, la declararea unei variabile nu are loc automat alocarea de spațiu) :
 - tipul clasă – class;
 - tipul interfață – interface;
 - tipul delegat – delegate;
 - tipul tablou – array;

1.1.8 Tipuri de conversii

1.1.1.1 Conversia implicită și explicită

O *conversie* permite ca o expresie de un anumit tip, să fie tratată ca o expresie de un alt tip.

Conversia implicită se efectuează automat fără a fi nevoie de un operator de cast și se poate realiza doar dacă valoarea copiată într-o locație de memorie este compatibilă cu tipul.

Există o regulă după care se efectuează conversiile implicite. În tabelul de mai jos sunt prezentate aceste reguli (Tab.4):

Conversie din	În
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal

float	double
ulong	float, double, decimal

Tab.4 Reguli de conversii implicite

Uneori, conversiile implicite duc la pierderi de precizie a valorilor, lucru semnalat de compilator. Atunci când nu există posibilitatea conversiei implicite, se realizează **conversia explicită** prin intermediul expresiei *cast*.

Tabelul de mai jos prezintă regulile de conversii explicite (Tab.5).

Conversie din	În
sbyte	byte, ushort, uint, ulong, char
byte	sbyte, char
short	sbyte, byte, ushort, uint, ulong, char
ushort	sbyte, byte, short, char
int	sbyte, byte, short, ushort, uint, ulong, char
uint	sbyte, byte, short, ushort, int, char
long	sbyte, byte, short, ushort, int, uint, ulong, char
ulong	sbyte, byte, short, ushort, int, uint, long, char
char	sbyte, byte, short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double

Tab.5 Reguli de conversii explicite

Exemplu3.) Conversia implicită și explicită

```

class Program
{
    static void Main(string[] args)
    {
        int a = 5;
        float b = 0;
        b = a; //conversie implicita. Datele nu vor fi pierdute.
        b = 0.9F;
        a = (int)b; //conversie explicita. Datele vor fi pierdute.
    }
}

```

1.1.1.1 Conversia între numere întregi și șiruri de caractere

O expresie poate fi convertită de la un tip la altul utilizând metodele din clasa `Convert`. De exemplu:

```
class Program
{
    static void Main(string[] args)
    {
        int m = Convert.ToInt32(123.23);
    }
}
```

Pentru a converti șiruri de caractere la alte tipuri, se poate folosi `Parse`. Există șiruri care conțin cifre. Pentru a utiliza valoarea numerică, șirul ar trebui convertit la un număr întreg. De exemplu:

```
class Program
{
    static void Main(string[] args)
    {
        string myText = "50";
        int myNumber = int.Parse(myText);
        //Conversie din string in integer
    }
}
```

O altă metodă C# pentru parsarea numerelor întregi este `int.TryParse`. Diferența dintre `int.TryParse` și `int.Parse` este faptul că prima poate fi folosită când datele sunt corupte. De exemplu, atunci când datele conțin non-numerice sau caractere nevalide, ar trebui să utilizăm `int.TryParse`. Dacă datele sunt valide și constau numai din numere, `int.Parse` este o soluție bună.

1.1.9 Opțiuni de afișare și scriere

Metodele din clasa `Console`, care aparține spațiului de nume `System` sunt utilizate în aplicațiile `Console`, pentru a putea scrie și a afișa informații în consolă. Printre cele mai utilizate metode de scriere și afișare pe scară largă, avem:

- `Write` – scrie un mesaj către consolă;
- `WriteLine` – scrie un mesaj către consolă și se mută către o linie nouă;
- `ReadLine` – citește o linie de text de la tastatură și se mută la o linie nouă.

Există două modalități de utilizare a metodelor `Write` și `WriteLine`:

- pentru tipurile de bază: `Console.WriteLine(valoare)` – permite afișarea directă, iar valoarea poate fi de orice tip (`int`, `string`, `char`, etc).
- cu formatare: `Console.WriteLine(format,parametrii)` – șirul de formatare este compus din textul de afișat în care sunt introduse elemente de forma `{i}` în locul unde trebuie inserate valorile parametrilor (`i` – începe de la 0 și reprezintă poziția parametrului în listă) și se comportă ca și funcția `printf` în C.

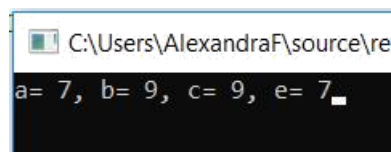
Exemplu4.) Afișarea cu formatare

```

static void Main(string[] args)
{
    int a, b, c, e, d = 7;
    a = 0;
    a = d++;
    b = ++d;
    c = d--;
    e = --d;
    Console.WriteLine("a={0}, b={1}, c={2}, e={3}", a, b, c, e);
    Console.ReadKey();
}

```

Rezultatul rulării acestui program este:



Observație!: Diferența dintre d++ și ++d este că expresia d++ incrementează valoarea lui d și returnează valoarea ne-incrementată pe când expresia ++d incrementează valoarea lui d și returnează valoarea incrementată.

Observație!: Unde avem {0} între ghilimele, se înlocuiește cu prima variabilă de după virgulă.

Exemplu 5.) Concatenare

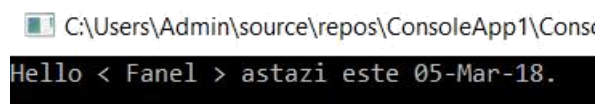
Observație!: O altă metodă de afișare a datelor este utilizarea operatorul + care înseamnă concatenare de string-uri.

```

class Program
{
    static void Main(string[] args)
    {
        string name = "< Fanel >";
        string date = DateTime.Today.ToShortDateString();
        string str = "Hello " + name + " astazi este " + date + ".";
        //folosim operatorul + pentru concatenare
        System.Console.WriteLine(str);
        Console.ReadKey();
    }
}

```

Rezultatul rulării programului este:



Pentru a introduce caractere de la tastatură se face fie utilizând metoda Console.Read(), aceasta returnează un întreg care trebuie convertit în tipul de date char, fie folosind metoda Console.ReadLine(), aceasta returnând un string.

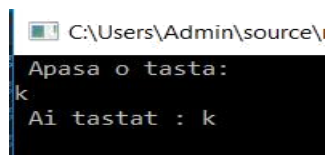
Exemplu 6.) Să se citească un caracter introdus de la tastatură.

```

class Program
{
    static void Main(string[] args)
    {
        char chr;
        Console.WriteLine("Apasa o tasta:");
        chr = (char)Console.Read();
        Console.Read();
        Console.WriteLine("Ai tastat: " + chr);
        Console.ReadKey();
    }
}

```

Rezultatul rulării programului este:



```

C:\Users\Admin\source\l
Apasa o tasta:
k
Ai tastat : k

```

1.1.10 Comentarii

Există trei tipuri de comentarii admise de limbajul C#, și anume:

- Comentariul pe un rand utilizând caracterele "// ". Tot ce urmează după acestea sunt considerate comentarii.
- Comentariul pe mai multe rânduri prin folosirea "/* " (începutul comentariului) și "*/" (sfârșitul comentariului). Tot textul cuprins între aceste simboluri se consideră comentariu.
- Creare document în format XML utilizând "///"; eXtensible Markup Language este un model de stocare a datelor nestructurate și semi-structurate, utilizat pentru transferul de date între aplicații pe Internet.

Exemplu 7.) Utilizarea comentariilor

```

class Program
{
    static void Main(string[] args)
    {
        /* string text = "A fost odată un om" + "și omul ăsta avea trei feciori." +
        "Pe cel mai mic dintre ei îl porecliseră Prostilă și-l luau în râs" +
        "și-l umpleau de ocări ori de câte ori aveau prilejul." +
        "Într-o bună zi," + "cel mai mare dintre frați vru să se ducă în pădure" +
        "să taie lemne și," + "mai înainte de a pleca, maică-sa îi puse n-traistă" +
        "un cozonac bine rumenit și tare gustos și-o sticlă cu vin," + "ca să aibă cu ce-și potoli foamea și setea.";
        Console.WriteLine(text); */
        Console.ReadKey();
    }
}

```

2. Instalarea mediului de dezvoltare pentru realizarea aplicațiilor C#

Pentru a realiza aplicații de tip “Console Application” în mediul de dezvoltare Visual Studio, trebuie să instalăm o versiune a acestuia de la adresa <https://www.visualstudio.com/downloads/> (ex. Visual Studio Community).

2.1 Crearea unei aplicații de tip “Console Application”

După lansarea mediului de programare, selectăm din meniul File -> New -> Project, conform fig.

1. Din această fereastră se alege tipul proiectului Console Application, se denumește proiectul (ex. ConsoleApplication1), se alege locația în care se va salva proiectul și se acționează butonul OK. Acesta este salvat în mod implicit în: C:\Users\Admin\source\repos .

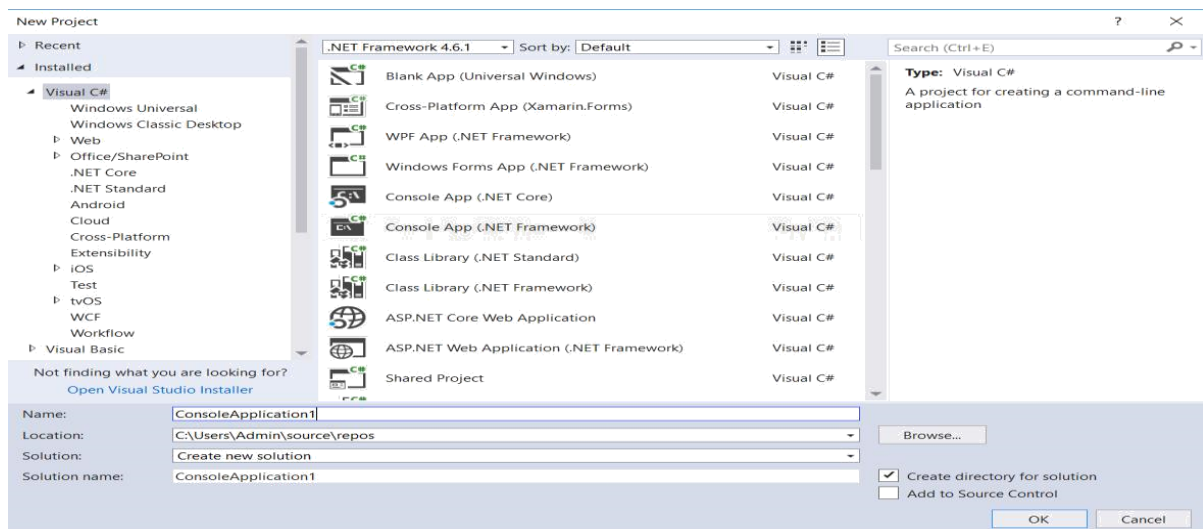


Fig.1 Alegerea șablonului pentru proiect

După crearea aplicației de tip consolă, se generează un fișier cu extensia .cs care provine de la C Sharp. Figura de mai jos (Fig.2) prezintă codul sursă generat automat la crearea unui proiect de tip Console Application.

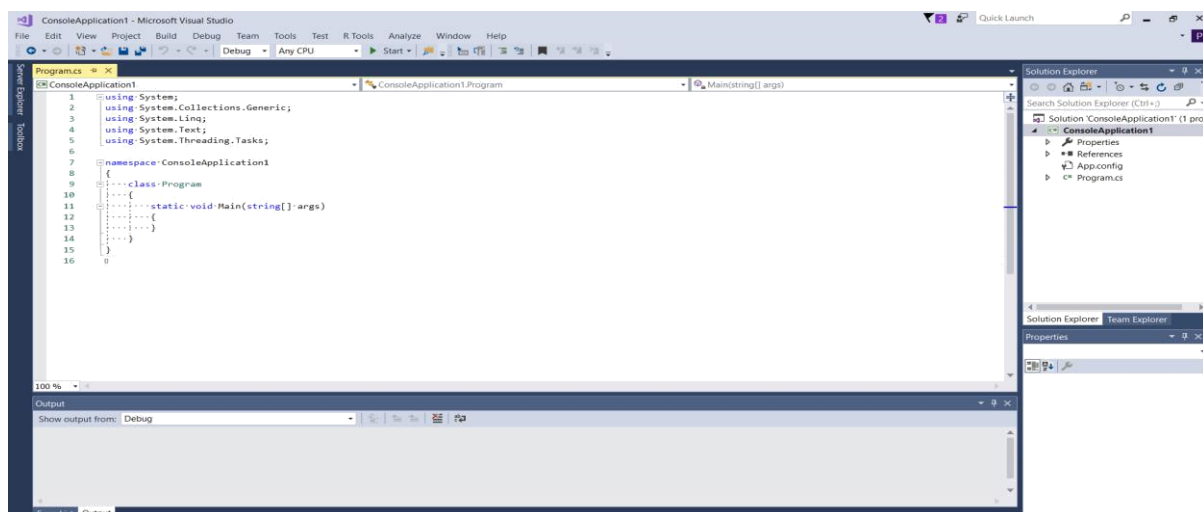


Fig.2 Console Application

2.1.1 Structura unei aplicații de tip “Console Application”

Structura unei aplicații de tip “Console Application”, denumită ConsoleApplication1, este prezentată în fig. 3 și conține:

- Un fișier Program.cs cu clasa principală Program;
- Un fișier ConsoleApplication1.csproj care permite deschiderea proiectului;
- Directoare bin, obj, Properties (doar acesta este vizibil în fereastra Solution Explorer). În directorul bin, subdirectorul Debug se creează în urma rulării fișierul executabil al aplicației.

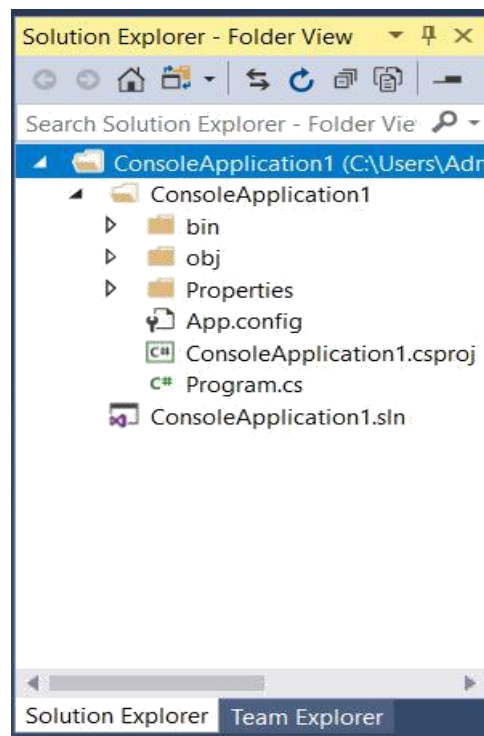


Fig3. Solution Explorer

O aplicație C# este formată din una sau mai multe clase, grupate în spații de nume (namespaces) și este obligatoriu ca doar una din aceste clase să conțină metoda (funcția) Main.

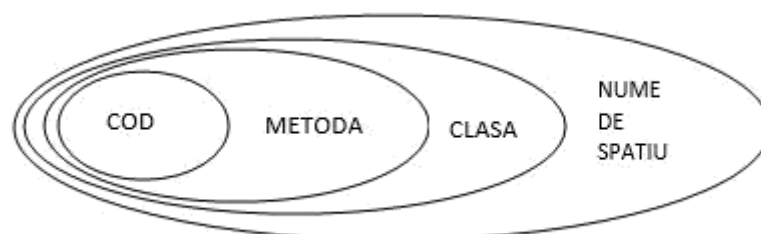


Fig4. Conținutul unei aplicații C#

2.1.2 Adăugarea codului în aplicații de tip “Console Application”

Adăugarea codului se face în interiorul metodei principale Main(). Clasa Console din namespace System este utilizată pentru operațiile de intrare-iesire și oferă o serie de metode, proprietăți și evenimente.

Ultima linie de cod conține Console.ReadKey(), aceasta se folosește pentru a menține deschisă consola până când este apasată o tastă. Dacă această metodă lipsește, consola se va închide imediat după executarea programului(Fig.5).

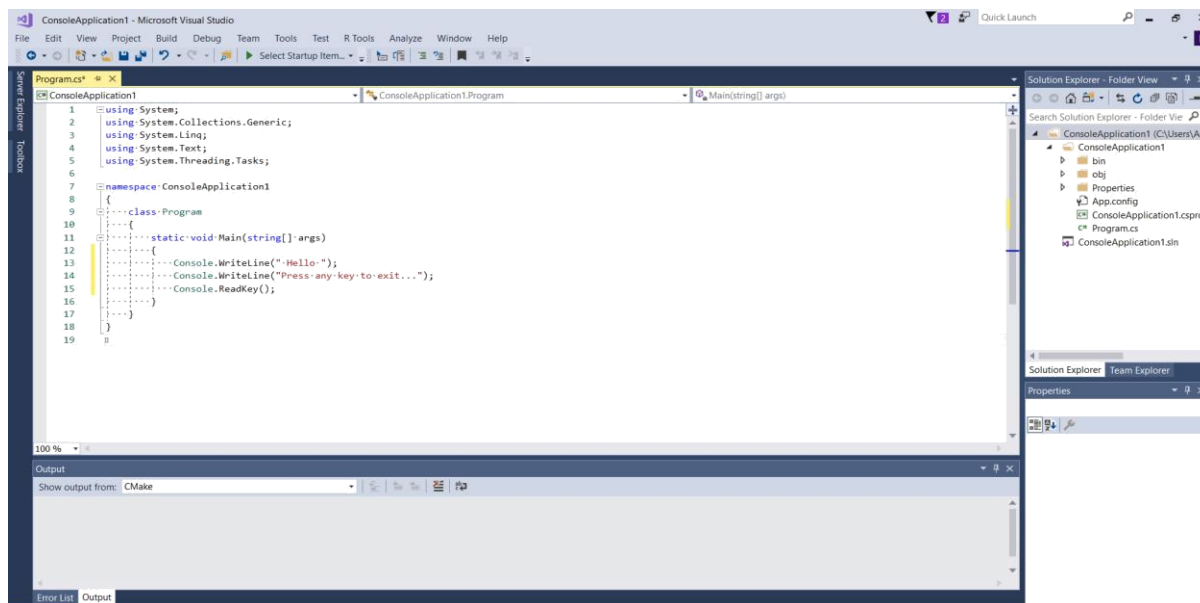


Fig.5 Console.ReadKey()

Datorită unelei IntelliSense, scrierea codului sursă al aplicațiilor este ușoară. În momentul începerii tastării unui cuvânt, IntelliSense oferă o listă cu numele claselor, metodelor, evenimentelor, etc. Așa cum este prezentat în figura 6.

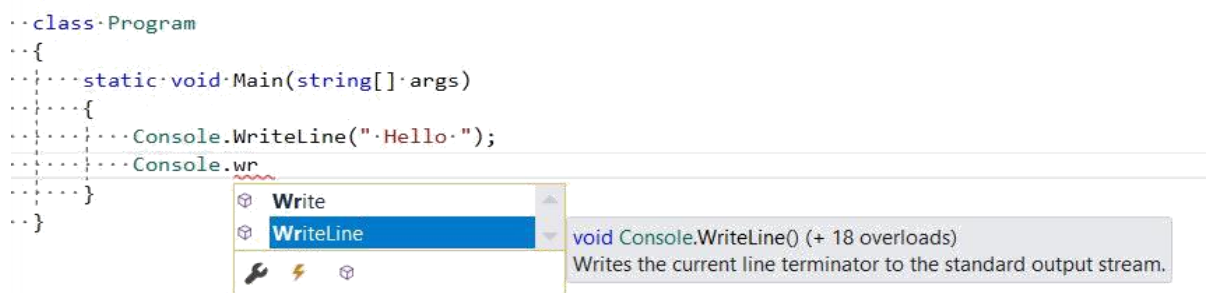






















Fig. 6 Exemplu de utilizare a unelei IntelliSense

Icoanele din IntelliSense cât și semnificația lor sunt prezentate în tabelul de mai jos(Tab. 6):

Simbol	Semnificatie	Simbol	Semnificatie	Simbol	Semnificatie
{ }	Namespace	⚡	Event	📦	Public enumeration
🔑	Interface	📦	Public method	📦	Protected enumeration

	Assembly		Protected method		Private enumeration
	Delegate		Structure		Constant inside enumeration
	Internal class		Public		Private method
	Protected class		Private class		Public property
	Public constant		Protected constant		Protected property
	Private constant		Public variable		Private property
	Protected variable		Private variable		

Tab.6 Icoanele din IntelliSense

2.1.3 Compilarea și rularea aplicațiilor de tip “Console Application”

Din meniul principal al mediului de programare se selectează Build -> Build Solution (F6) și astfel se face compilarea aplicației ConsoleApplication1. În cazul în care există erori, acestea sunt afișate în fereastra Error List. Efectuând dublu click pe fiecare eroare în parte, cursorul din program se poziționează pe linia care conține eroarea.

Rularea programului se poate realiza în mai multe moduri:

- Rapid fără asistență de depanare (Start Without Debugging Ctrl+F5 sau cu butonul din bara de instrumente(Fig. 7));

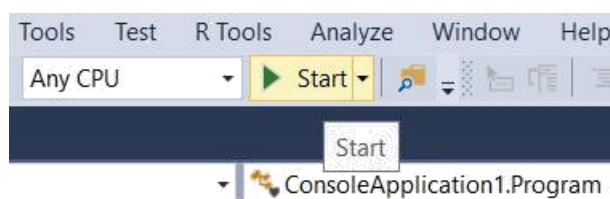


Fig. 7 Rapid fără asistență de depanare

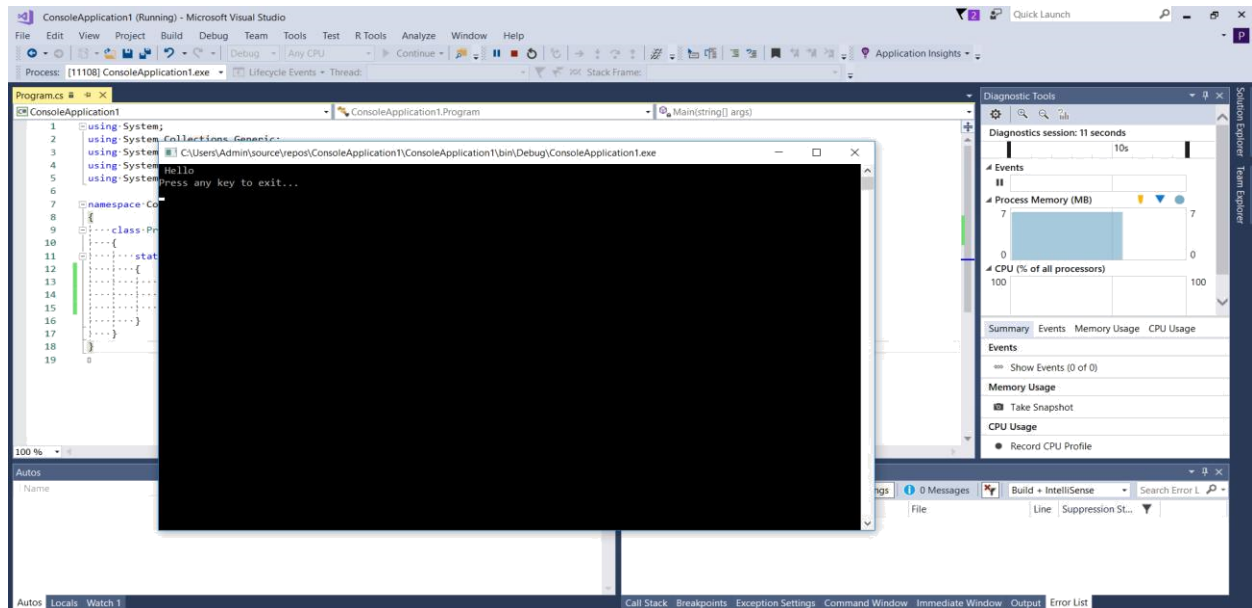


Fig.8 Rezultatul rulării codului



O altă metodă pentru a rula programul este folosirea fișierului .exe din subdirectorul /bin/debug în afara mediului de dezvoltare, din Command Prompt. Rezultatul se poate vedea în figura 11;

Notă: Pentru a putea rula programul din Command Prompt, trebuie să avem calea(path) către fișierul executabil. Deci, primul pas este de a acționa click dreapta pe numele proiectului, pentru a putea deschide directorul în care este salvat proiectul, după care copiem calea (path) ca și în fig. 9, fig. 10:

C:\Users\Admin\source\repos\ConsoleApplication1\ConsoleApplication1\bin\Debug

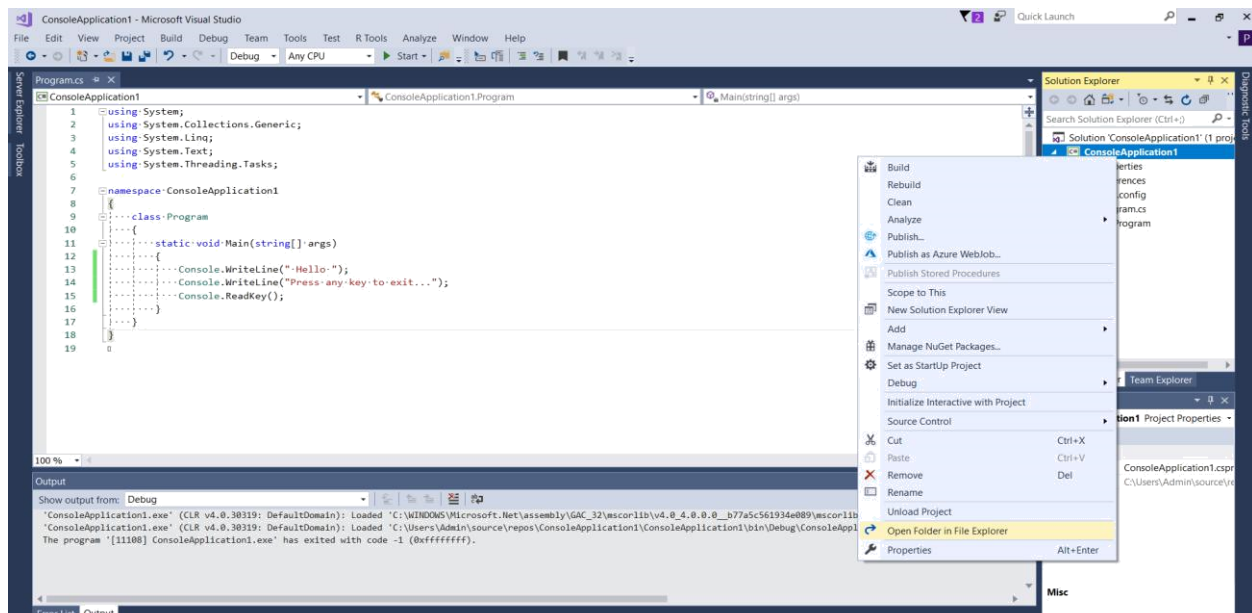


Fig.9 Deschiderea folderului sursă

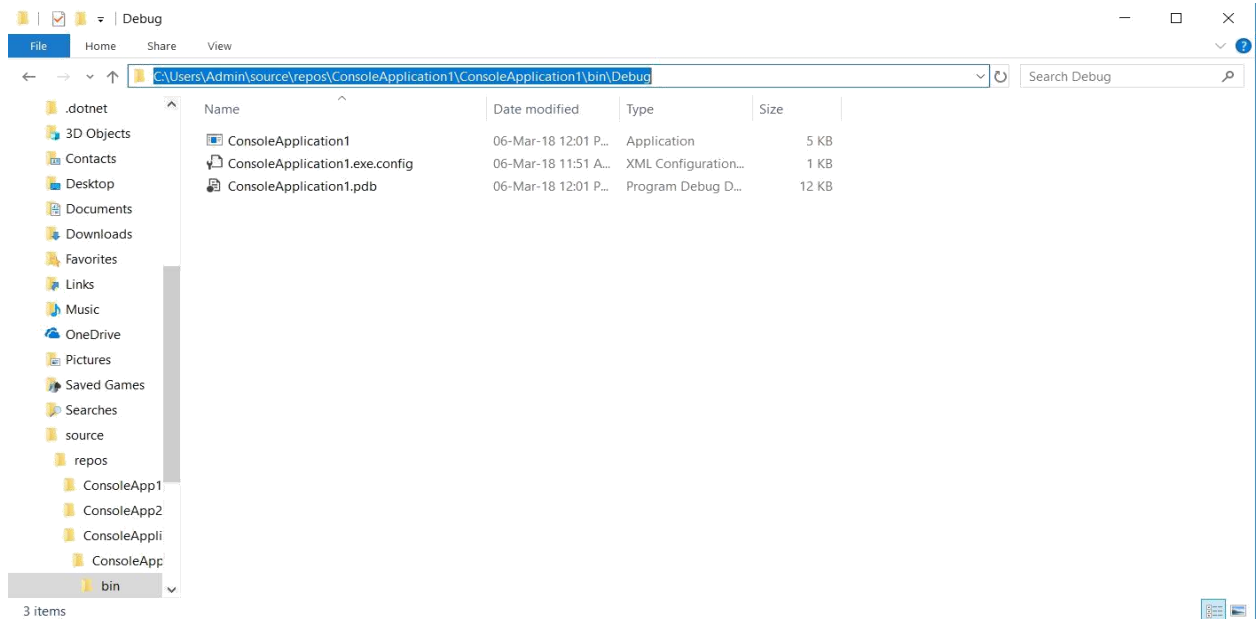


Fig.10 Folderul sursă

Deschidem o fereastră Command și se va tasta comada:

`cd C:\Users\Admin\source\repos\ConsoleApplication1\ConsoleApplication1\bin\Debug`, după care scriem numele fișierului executabil cu extensia .exe: `ConsoleApplication1.exe`, la fel ca și în exemplul de mai jos (Fig.11).

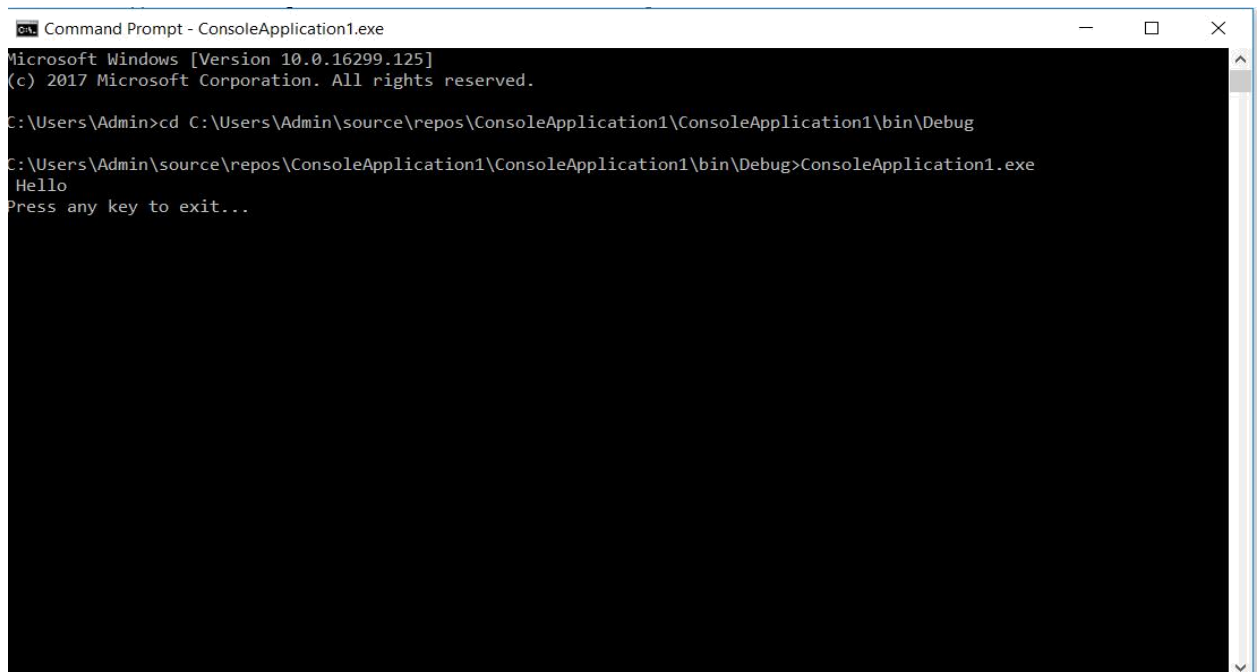


Fig.11 Rularea aplicatiei din Command Prompt

- rapid cu asistență de depanare (Start Debugging F5 sau cu butonul din bara de instrumente);
- rulare pas cu pas (Step Into F11 și Step Over F10) ;

- rulare rapidă până la linia marcată ca punct de întrerupere (Toggle Breakpoint F9 pe linia respectivă și apoi Start Debugging F6). Încetarea urmăririi pas cu pas (Stop Debugging Shift+F5) permite ieșirea din modul depanare și revenirea la modul normal de lucru.

Observatie!: Toate opțiunile de rulare și depanare se găsesc în meniul Debug al mediului de programare (Fig. 12).

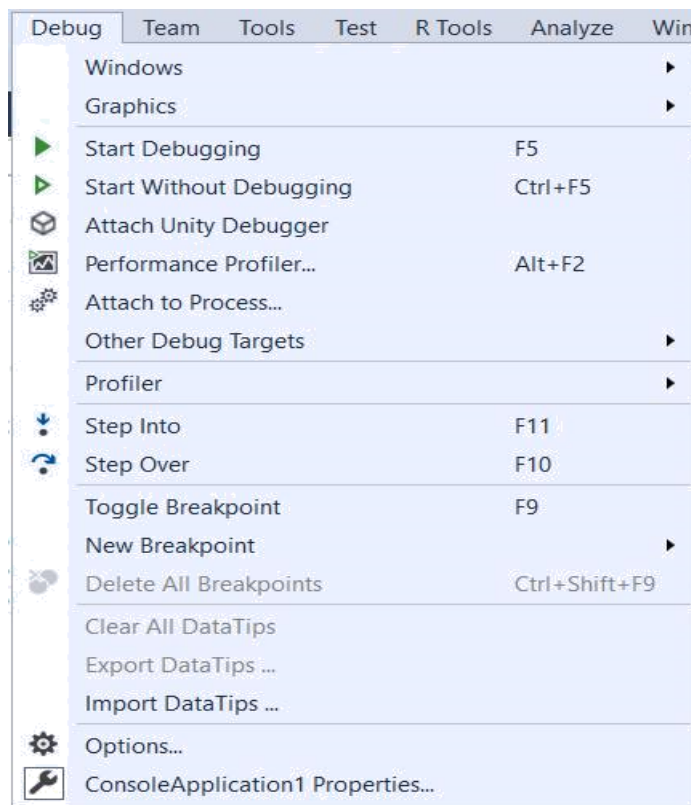


Fig12. Bara de instrumente pentru rularea aplicației

3. Exerciții

- Ex1** Să se determine și să se afișeze primii n termeni ai secvenței Fibonacci. ($F_0 = 0$, $F_1 = 1$, după care se utilizează formula recursivă $F_n = F_{n-1} + F_{n-2}$).
- Ex2** Citiți două numere reale de la tastatură și apoi, utilizând operatorii binari $+$, $-$, $*$, $/$, să se execute câteva calcule. Sa se creeze o clasă care să conțină metode pentru operațiile matematice cât și o metodă pentru afișare.
- Ex3** Să se realizeze o aplicație care să convertească temperaturi din $^{\circ}\text{C}$ în $^{\circ}\text{F}$ și din $^{\circ}\text{F}$ în $^{\circ}\text{C}$, utilizând formulele $F = C \cdot 9/5 + 32$, $C = (F - 32) \cdot 5/9$. Aplicația trebuie să conțină două metode, iar valorile să se introducă de la tastatură.
- Ex4** Să se calculeze greutatea ideală (kg) în funcție de înălțime (cm), vârsta (ani) și sex (f sau m). Să se creeze o clasă nouă care să conțină trei metode: două pentru calculul greutății și una pentru afișarea greutății ideale utilizând formulele de mai jos. Parametrii sunt citiți de la tastatură.
- Bărbați:** $\text{Greutatea ideală} = \lfloor \text{înălțimea (cm)} - 100 - ((\text{înălțimea} - 150) / 4) \rfloor + ((\text{vârsta} - 20) / 4)$;
- Femei:** $\text{Greutatea ideală} = \lfloor \text{înălțimea (cm)} - 100 - ((\text{înălțimea} - 150) / 2.5) \rfloor + ((\text{vârsta} - 20) / 6)$.
- Ex5** Să se introducă un șir de numere întregi de la tastatură și să se calculeze media geometrică și aritmetică a numerelor.