Programare orientată obiect

Bibliografie

- Liviu Negrescu **- *Limbajele C si C++ pentru începători,*** *Volumul II - Limbajul C++*, Editura Albastră, Cluj-Napoca, 1995.
- Herbert Schildt - ***C++ manual complet***, Bucureşti , Editura Teora, 2000
- Herbert Schildt - ***C Manual Comple***t, Bucureşti, Editura Teora, 1998
- Bjarne Stroustrup - ***The C++ Programming Language***, Adisson-Wesley, 1997
- Robert Lafore - ***C++ Interactive Course*****,** Macmillan Computer Publishing 1996
- Kris Jamsa & Lars Klander - ***Totul despre C si C++ - Manualul fundamental de programare in C si C++*** , Teora, 2007

Resurse electronice

- Bruce Eckel - ***Thinking in C++***, 2nd ed., Volumele 1 si 2

  - http://www.mindview.net

- Peter Müller - ***Introduction to Object-Oriented Programming Using C++***

  - http://gd.tuwien.ac.at/languages/c/c++oop-pmueller/

- Frank B. Brokken - ***C++ Annotations Version 10.1.0***

  - http://www.icce.rug.nl/documents/cplusplus/

- MSDN Library – Visual Studio 2010 - Visual C++

  - http://msdn.microsoft.com/en-us/library/60k1461a%28v=vs.100%29.aspx

Paradigme de programare

Paradigma defineşte un anumit stil fundamental de programare.
Cele mai frecvent utilizate paradigme de programare sunt:

- programarea procedurală (imperativă) – are la bază utilizarea procedurilor (echivalent, a funcţiilor în C). Accentul se pune pe execuţia unor acţiuni, importanţa datelor este subevaluată. Se presupune o separare fermă între structurile de date şi codul funcţiilor care le prelucrează => o modificare a aplicaţiei atrage după sine atât o reproiectare a structurii de date, cât şi o reimplementare a funcţiilor de prelucrare.

- programarea modulară – termenul de modul defineşte o colecţie de funcţii înrudite împreună cu datele pe care le prelucrează şi care se compilează independent. In acest mod, aplicaţiile se rezolvă printr-un program alcătuit din module. Un modul poate proteja o parte din datele şi funcţiile cu care operează, preîntâmpinând utilizarea neautorizată / eronată a funcţiilor şi alterarea nedorită a datelor.

- programarea orientată obiect (POO) – reprezintă etapa actuală, modernă de proiectare a produselor software complexe. Într-o primă aproximare, orientarea obiect înseamnă organizarea resurselor soft sub forma unor colecţii de obiecte, ce înglobează atât structuri de date, cât şi funcţiile de prelucrare a acestora. O clasă (de obiecte) grupează entităţi cu proprietăţi similare. Această similitudine se referă atât la descriere (date sau atribute ale acestora), cât şi la comportare (funcţii), dar în acelaşi timp şi la relaţii posibile cu alte obiecte. Diferenţele dintre obiectele aceleiaşi clase se materializează în diferenţele dintre valorile datelor de descriere. POO se mai caracterizează şi prin aceea că datele şi funcţiile (obiectele) cu care se operează într-o aplicaţie au fost supuse unui proces de clasificare, pe baza unor trăsături identificate ca fiind comune. Acest proces conduce în mod natural la stabilirea de ierarhii. In vârful unei ierarhii se află entitatea care are trăsăturile comune pentru toate celelalte componente ale ierarhiei respective (elementul din vârful ierarhiei se consideră a fi cel mai general). Toate celelalte entităţi sunt particularizări/specializări, ele moştenind trăsăturile elementelor de pe nivelele anterioare ale ierarhiei.

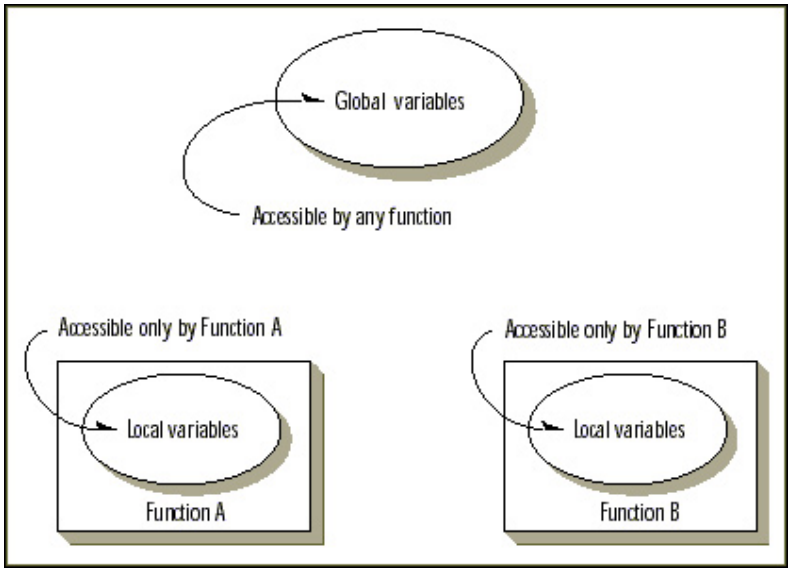

Figura 1.1.Variabile globale şi locale.

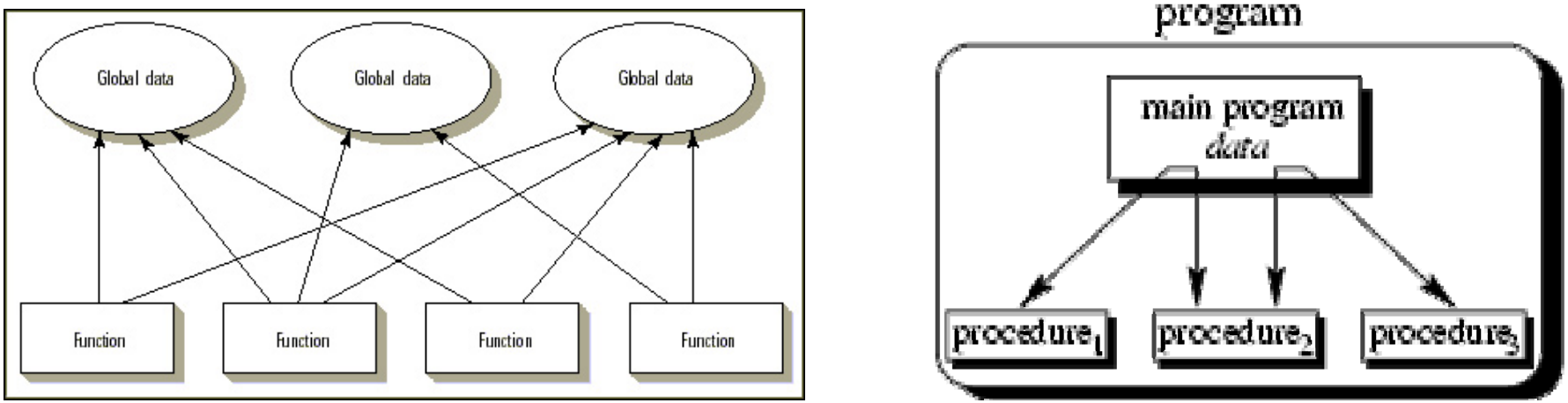

Figura 1.2. Paradigma procedurală.

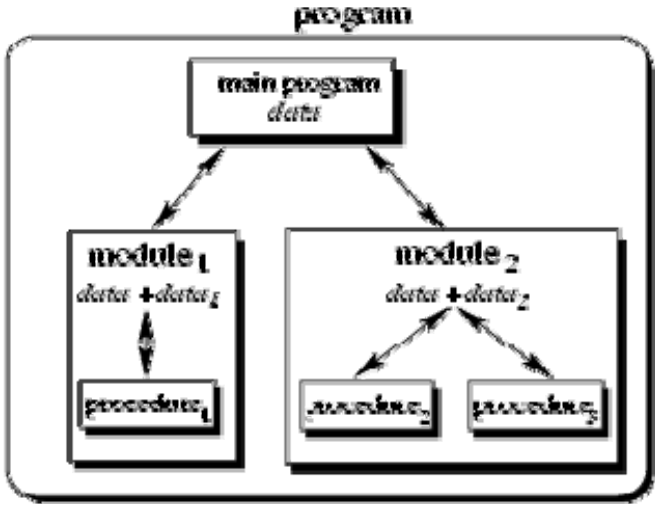

Figura 1.3. Paradigma modulară

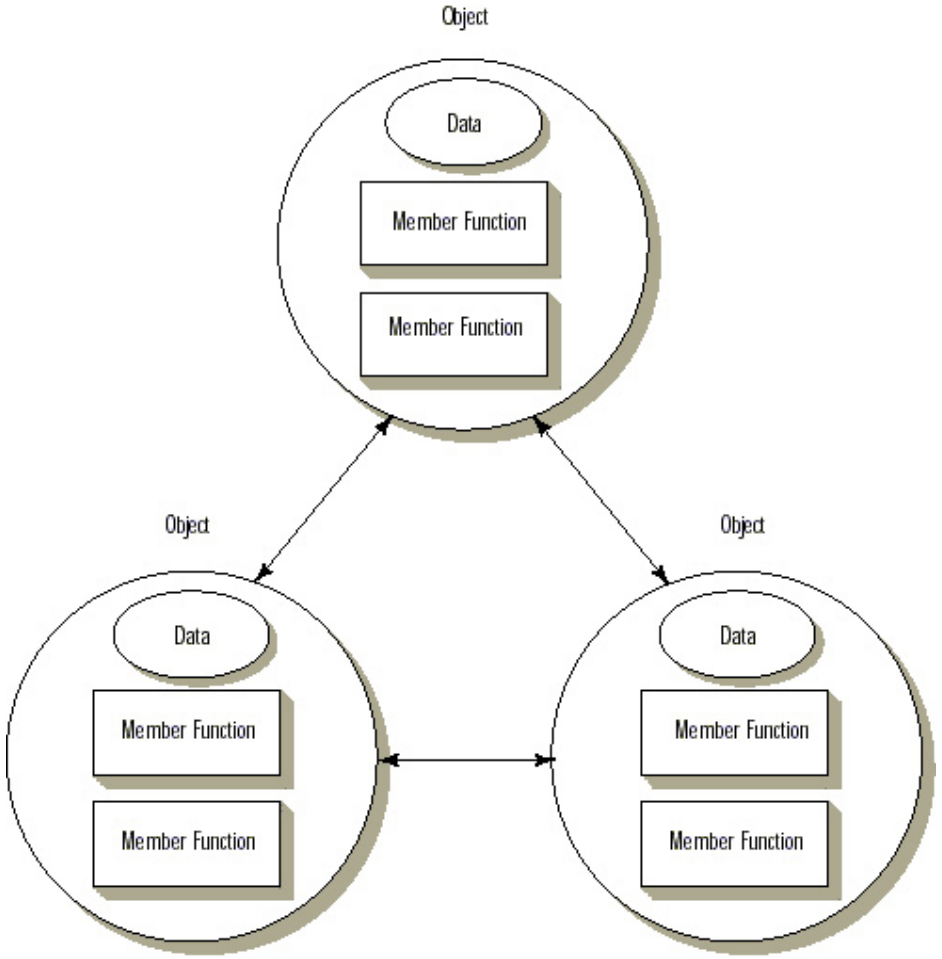

Figura 1.4. Paradigma orientată obiect

Un tip de date descrie un set de entităţi ce au o aceeaşi reprezentare. Mai precis, caracterizarea unui tip de date impune două aspecte : unul legat de modul efectiv de reprezentare a datelor, iar celălalt, de operaţiile ce se pot executa asupra datelor respective.

Tipurile de date ce au drept componente atât date, cât şi funcţii care le prelucrează se numesc **tipuri de date abstracte**. Ele sunt o generalizare a tipurilor de date definite de utilizator. Un tip de date abstract este, în acest context, reprezentarea unui concept. O definiţie echivalentă : **un tip de date abstract este un tip definit de comportamentul/ proprietăţile sale şi nu de reprezentare**.

Stilul de programare care permite definirea unei ierarhii între tipuri abstracte de date pe baza conceptului de moştenire se numeşte POO.

Principiile programării orientate obiect sunt :

- existenţa **tipurilor de date abstracte (abstract data types)** – ce extind setul de tipuri predefinite(standard) ale limbajului;
- **încapsularea (encapsulation)** – posibilitatea de a grupa date şi funcţii înrudite (corelate) într-o entitate autonomă, denumită clasă;
- **ascunderea informaţiei (information hiding)** – posibilitatea oferită de a restricţiona accesul la anumite date şi funcţii (de obicei se ascund detaliile de implementare ce sunt mult mai probabil supuse modificărilor ulterioare, rămânând accesibile doar componentele ce constituie interfaţa);
- **moştenirea (inheritance)** – conferă posibilitatea ca o clasă sa preia funcţionalităţile unei alte clase, ceea ce permite reutilizarea codului;
- **polimorfismul (polymorphism)** – semnificaţia exactă a termenului înseamnă “forme multiple”, în POO el referă abilitatea unui obiect de a răspunde într-o manieră specifică unui anumit mesaj.

Mai concis POO înseamnă deci :

- gruparea datelor şi funcţiilor
- separarea între interfaţă şi implementare
- definirea unor operaţii dependente de tipul operanzilor
- stabilirea unei ierarhii între entităţile cu care se operează
- reutilizarea codului

Aceasta se realizează prin :

- suport pentru tipurile abstracte de date
- trimiterea de mesaje în locul apelurilor de funcţii
- compilatoare care permit implementarea conceptului de moştenire

Actualmente există limbaje de programare care oferă suport pentru POO (C++, C#, Java, Pascal, Modula-2, Smalltalk).

Introducere în C++

- Autorul limbajului: ***Bjarne Stroustrup***
- Limbajul C++ este construit pe structura limbajului C, fiind un superset al acestuia (o versiune dezvoltată)
- Motivaţie : pentru a beneficia de forţa şi flexibilitatea celui mai popular şi mai important limbaj de programare :
  - flexibilitate - domenii de aplicabilitate foarte diverse, se pot utiliza o varietate extrem de largă de tehnici de programare;
  - eficacitate - apropierea de limbajele de nivel coborât oferă utilizatorului posibilitatea de a folosi la maximum resursele hardware ale maşinii de calcul;
  - disponibilitate - compilatoarele de C standard şi bibliotecile asociate sunt extrem de frecvente (de la minicalculatoare la supercalculatoare)
  - portabilitate - aproape întotdeauna asigurată.
- În concluzie crearea unui limbaj C mai bun permitea corijarea micilor imperfecţiuni, fără pierderea avantajelor deja existente.
- Obiectivele proiectării limbajului C++
  - să ofere suport pentru abstractizarea datelor - clase
  - să ofere suport pentru programarea orientată obiect – moştenire (inheritance), polimorfism (funcţii virtuale, supraîncărcarea operatorilor)
  - să ofere suport pentru programarea generică – tipuri parametrizate (templates)
  - să fie un C mai bun - "As close to C as possible, but no closer" : execuţie eficientă, compatibilitate cu bibliotecile limbajului C

Scurt istoric

O primă descriere a "C-ului cu clase" a fost publicată ca raport la Bell Labs în aprilie 1980 de Bjarne Stroustrup. Obiectivul a fost acela de a facilita organizarea programelor. Se introduc : conceptul de clasă, clasă derivată, controlul accesului public şi privat, constructori şi destructori.

In 1982 a devenit evident că succesul C-ului cu clase e limitat şi s-a căutat un succesor. Astfel s-a născut C++.

Manualul de referinţa al limbajului s-a publicat în 1984. Prima reuniune a comitetului de standardizare ANSI C++ a avut loc în decembrie 1989. Primul proiect de standard a fost propus în 1994, iar pe 28 IX 1998 a fost aprobat limbajul C++ ca standard ANSI/ISO.

C++ Humor

Q: How many C++ programmers does it take to change a light bulb?
A: You're still thinking procedurally. A properly designed light bulb object would inherit a change method from a generic light bulb class, so all you would have to do is send a light-bulb-change message.

"Fifty years of programming language research, and we end up with C++ ???."*(Richard A. O'Keefe)*

"C++: Hard to learn and built to stay that way."

"The evolution of languages: FORTRAN is a non-typed language. C is a weakly typed language. Ada is a strongly typed language. C++ is a strongly hyped (overvalued) language." (*Ron Sercely)*

"C(++) is a write-only, high-level assembler language." (*Stefan Van Baelen)*

"C++ would make a decent teaching language if we could teach the ++ part without the C part." (*Michael B. Feldman)*

"The great thing about Object Oriented code is that it can make small, simple problems look like large, complex ones."

"C++ in Cantonese is pronounced 'C ga ga'. Need I say more?" (Mark Glewwe)

"When your hammer is C++, everything begins to look like a thumb."

THE PROGRAMMER'S QUICK GUIDE TO THE LANGUAGES (An old gem)

The proliferation of modern programming languages (all of which seem to have stolen countless features from one another) sometimes makes it difficult to remember what language you're currently using. This handy reference is offered as a public service to help programmers who find themselves in such a dilemma.

TASK: Shoot yourself in the foot.

C: You shoot yourself in the foot.

C++: You accidentally create a dozen instances of yourself and shoot them all in the foot. Providing emergency medical assistance is impossible since you can't tell which are bitwise copies and which are just pointing at others and saying, "That's me, over there."

FORTRAN: You shoot yourself in each toe, iteratively, until you run out of toes, then you read in the next foot and repeat. If you run out of bullets, you continue with the attempts to shoot yourself anyway because you have no exception-handling capability.

Pascal: The compiler won't let you shoot yourself in the foot.

BASIC: Shoot yourself in the foot with a water pistol. On large systems, continue until entire lower body is waterlogged.

Visual Basic: You'll really only _appear_ to have shot yourself in the foot, but you'll have had so much fun doing it that you won't care.

Assembler: You try to shoot yourself in the foot, only to discover you must first invent the gun, the bullet, the trigger, and your foot.

Modula2: After realizing that you can't actually accomplish anything in this language, you shoot yourself in the head.

Quotes: *"C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole leg." - Bjarne Stroustrup*

C în C++

C++ este o extensie a limbajului C ce oferă:

- o verificare mai eficientă a validităţii tipurilor de date (strong typechecking)
- suport pentru abstractizarea datelor (data abstraction)
- suport pentru programarea orientată obiect
- suport pentru tratarea eficientă a erorilor (exception handling)
- funcţii inline
- transferul prin referinţă al parametrilor
- valori implicite pentru parametrii funcţiilor
- tipul de date logice *bool*
- numele structurilor, claselor, uniunilor, enumerărilor ca nume de tip
- un mecanism de control al domeniilor numelor (*namespace)*
- reutilizarea codului (code reuse)

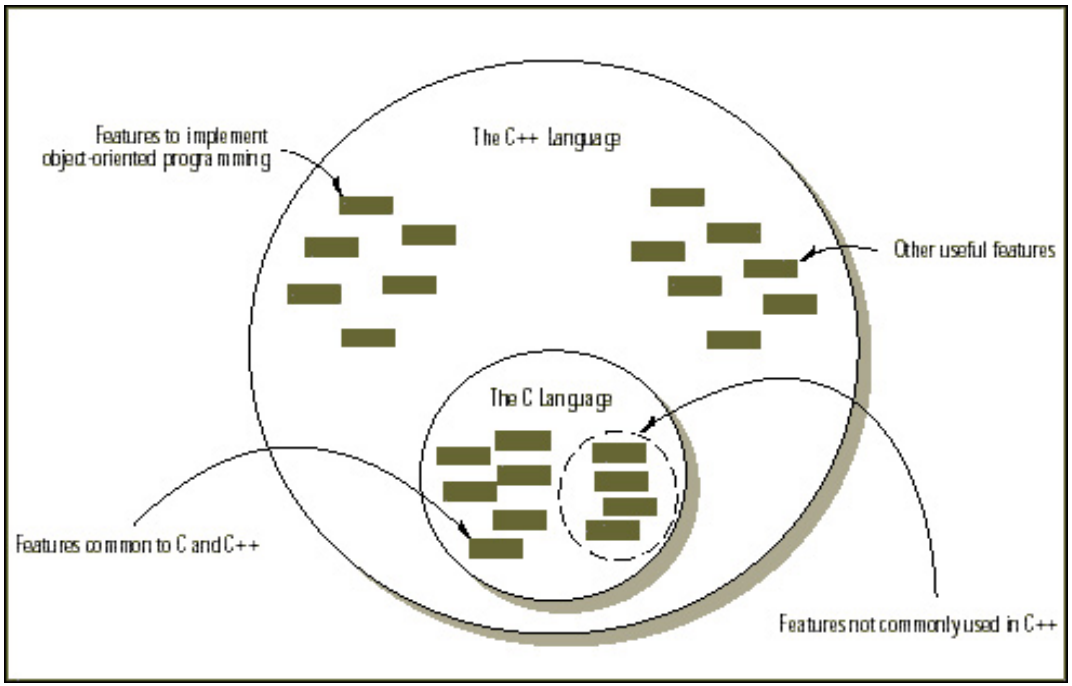

Figura 1.5 Relaţia dintre C şi C++

Tabelul 1.1 Tipuri de date predefinite/standard în C/ C++

Nume	Folosit pentru a stoca	Exemple de valori
char	Caractere	‘a’, ‘B’, ‘$’, ‘3’, ‘?’
short	numere întregi de valori mici	12, -1, 100
int	numere întregi de valori uzuale	730, 0, -22200
long	numere întregi de valori mari	1000000000, -123456789
float	numere reale de valori mici	3.7, 199.99, -16.2, 0.000125
double	numere reale de valori mari	7,553.393.95, 47, -0.048512934
long double*	numere reale de valori foarte mari	779123456789.09012, -345.66678342681234

* neimplementat în Microsoft Visual C++

Tabelul 1.2 Tipuri de date întregi

Numele tipului	Dimensiune	Domeniu
char	1 byte (8 biţi)	-128 până la 127
short	2 bytes (16 biţi)	-32768 pana la 32767
int	coincide cu short la sistemele de 16 biţi sau cu long pentru cele pe 32 biţi	
long	4 bytes (32 biţi)	-2147483648 pana la 2147483647

Tabelul 1.3 Tipuri întregi fără semn (unsigned integers)

Numele tipului	Dimensiune	Domeniu
unsigned char	1 byte (8 biţi)	0 până la 255
unsigned short	2 bytes (16 biţi)	0 pana la 65535
unsigned int or unsigned	coincide cu short la sistemele de 16 biţi sau cu long pentru cele pe 32 biţi	
unsigned long	4 bytes (32 biţi)	0 pana la 4294967295

Tabelul 1.4. Tipuri de date reale (floating-point)

Numele tipului	Dimensiune	Domeniu pentru modul	Precizie
float	4 bytes (32 biţi)	$[1.17*10^{-38}, 3.4*10^{+38}]$	7 cifre
double	8 bytes (64 biţi)	$[2.2*10^{-308}, 1.79*10^{+308}]$	15 cifre
long double	10 bytes (80 biţi)	$[3.4*10^{-4932}, 1.1*10^{+4932}]$	19 cifre

Tipuri de date derivate :

- tipul pointer
- tipul tablou

Tipuri de date definite de utilizator :

- tipul structură
- tipul uniune
- tipul enumerare

Tabelul 1.5. Tipuri de date predefinite/standard în Visual C++ 2010

Tip	Dimensiune	Domeniu de valori
char	1 byte	[-127 , 127] sau [0 , 255]
unsigned char	1 byte	[0 , 255]
signed char	1 byte	[-127 , 127]
int	4 bytes	[-2147483648 , 2147483647]
unsigned int	4 bytes	[0 , 4294967295]
signed int	4 bytes	[-2147483648 , 2147483647]
short int	2 bytes	[-32768 , 32767]
unsigned short int	2 bytes	[0 , 65.535]
signed short int	2 bytes	[-32768 , 32767]
long int	4 bytes	[-2.147.483.647 , 2.147.483.647]
unsigned long int	4 bytes	[0 , 4.294.967.295]
float	4 bytes	[1.17*10-38, 3.4*10+38] (~7 cifre) - domeniu pentru modul
double	8 bytes	[2.2*10-308, 1.79*10+308] (~15 cifre) - domeniu pentru modul
long double	8 bytes	[2.2*10-308, 1.79*10+308] (~15 cifre) - domeniu pentru modul
wchar_t	2 sau 4 bytes	caractere extinse

Tabelul 1.6. Tabelul operatorilor în C++, organizaţi pe categorii :

Operatori aditivi

Addition: + Subtraction: –

Atribuire

Addition Assignment: +=	Assignment: =	Bitwise AND Assignment: &=
Bitwise exclusive OR Assignment: ^=	Bitwise inclusive OR Assignment: |=	Division Assignment: /=
Left shift assignment: <<=	Modulus Assignment: %=	Multiplication Assignment: *=
Right shift assignment: >>=	Subtraction Assignment: –=	

Operatori pe biţi

Bitwise AND: & Bitwise exclusive OR: ^ Bitwise inclusive OR: |

Operatori logici

Logical AND: && Logical OR: ||

Alţi operatori

Comma: , Conditional: ? : Pointer-to-member: .* or –>*
Reference: & Scope resolution: ::

Operatori multiplicativi

Division: / Modulus: % Multiplication: *

Operatori postfixaţi

Cast: ()	Function call: ( )	Member access: . and –>
Postfix decrement: ––	Postfix increment: ++	Subscript: [ ]

Operatori de relaţie şi egalitate

Equality: ==	Greater than or equal to: >=	Greater than: >
Less than or equal to: <=	Less than: <	Not equal: !=

Operatori de deplasare

Left shift: << Right shift: >>

Operatori unari

Address-of: &	delete	Indirection: *
Logical Negation: !	new	One's Complement: ~
Prefix decrement: ––	Prefix increment: ++	sizeof
Unary Plus Operator: +	Unary Negation Operator: -	

Tabelul 1.7. Tabelul principalilor operatori, în ordinea priorităţilor acestora :

```
: :
(    )    [    ]    .    ->
-                        +    *    &    !    ~    ++    --    sizeof
*    /    %
+    -
<<    >>
<    <=    >=    >
==    !=
&
^
|
&&
||
?    :
=    +=    -=    *=    /=    %=    <<=    >>=    &=    ^=    |=
,
```

Operator	Priority	Description	Order
()	1	Function call operator	from left
[]	1	Subscript operator	from left
->	1	Element selector	from left
!	2	Boolean NOT	from right
~	2	Binary NOT	from right
++	2	Post-/Preincrement	from right
--	2	Post-/Predecrement	from right
-	2	Unary minus	from right
(*type*)	2	Type cast	from right
*	2	Dereference operator	from right
&	2	Address operator	from right
sizeof	2	Size-of operator	from right
*	3	Multiplication operator	from left
/	3	Division operator	from left
%	3	Modulo operator	from left
+	4	Addition operator	from left
-	4	Subtraction operator	from left
<<	5	Left shift operator	from left
>>	5	Right shift operator	from left
<	6	Lower-than operator	from left
<=	6	Lower-or-equal operator	from left
>	6	Greater-than operator	from left
>=	6	Greater-or-equal operator	from left
==	7	Equal operator	from left
!=	7	Not-equal operator	from left
&	8	Binary AND	from left
^	9	Binary XOR	from left
|	10	Binary OR	from left
&&	11	Boolean AND	from left
||	12	Boolean OR	from left
?:	13	Conditional operator	from right
=	14	Assignment operator	from right
op=	14	Operator assignment operator	from right
,	15	Comma operator	from left

Instrucţiuni :

- instrucţiunea vidă
- instrucţiunea compusă
- instrucţiunea expresie
- instrucţiunea *if*
- instrucţiunea *switch*
- instrucţiunea *return*
- instrucţiuni de salt :
  - instrucţiunea *continue*
  - instrucţiunea *break*
  - instrucţiunea *goto*
- instrucţiuni ciclice :
  - instrucţiunea *while*
  - instrucţiunea *for*
  - instrucţiunea *do-while*

Clase de alocare de memorie :

- auto
- static
- registru
- dinamică (în zona *heap*)

Domeniile numelor :

- local
- fişier
- global

Tipuri de legături (linkage) :

- intern
- extern

Tabel cu instrucţiunile limbajului C++

Statement	Description
`break;`	Leave current block. Also used to leave `case` statement in `switch`.
`continue;`	Only used in loops to continue with next loop immediately.

`do`
stmt
`while (`*expr*`);`
	Execute *stmt* as long as *expr* is TRUE.

`for (`[*expr*]`;` [*expr*]`;` [*expr*]`)` *stmt*	This is an abbreviation for a `while` loop where the first *expr* is the initialization, the second *expr* is the condition and the third *expr* is the step.
`goto` *label*`;`	Jumps to position indicated by *label*. The destination is *label* followed by colon ":".
`if (`*expr*`)` *stmt* [`else` *stmt*]	IF-THEN-ELSE in C notation
`return` [*expr*]`;`	Return from function. If function returns `void return` should be used without additional argument. Otherwise the value of *expr* is returned.

`switch (`*expr*`) {`
`case` *const-expr*`:` *stmts*
`case` *const-expr*`:` *stmts*
...
[`default:` *stmts*]
`}`
	After evaluation of *expr* its value is compared with the `case` clauses. Execution continues at the one that matches. BEWARE: You **must** use `break` to leave the `switch` if you don't want execution of following `case` clauses! If no `case` clause matches and a `default` clause exists, the statements of the default clause are executed.

`while (`*expr*`)` *stmt*	Repeat *stmt* as long as *expr* is TRUE.