Introducere în C și C++

Săptămâna 1 — de la cod la program executabil

Alexandru Ioniță

October 1, 2026

Cuprins

1. Introducere

2. Sintaxă și structură

3. Variabile și tipuri

4. Operatori și expresii

5. Interacțiune și control

Introducere

Ce sunt C și C++?

C	C++
General-purpose	General-purpose
Procedural	Procedural
Compilat	Compilat
Statically typed	Statically typed
	Clase, Șabloane

- **Procedural**: programul este organizat în funcții care descriu pași de execuție.
- **Compilat**: compilatorul transformă codul sursă în cod mașină înainte ca programul să ruleze.
- **Statically typed**: tipul fiecărei variabile este cunoscut și verificat la compilare.
- **Clase**: tipuri care grupează date și operații; **șabloane**: cod generic adaptat tipurilor folosite.
- Codul sursă este text scris de programator; procesorul execută instrucțiuni mașină.

Ce înseamnă „compilat”?

Un compilator transformă codul sursă într-un program executabil înainte de rulare.

$$\text{Cod sursă} \longrightarrow \text{compilator} \longrightarrow \text{executabil} \longrightarrow \text{rulare}$$

- erorile de sintaxă și de tip sunt descoperite, de regulă, la compilare;
- executabilul poate rula fără compilatorul prezent pe fiecare execuție;
- executabilul este dependent de platformă și arhitectură.

Primul program în C

main.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, world!\\n");
    return 0;
}
```

- #include aduce declarații dintr-o bibliotecă;
- main este punctul de pornire al programului;
- acoladele delimitează un bloc;
- ; încheie o instrucțiune.

Compilarea pe Linux, macOS și Windows

Linux / macOS

```
gcc -std=c17 -Wall -Wextra main.c
↪  -o main
./main

g++ -std=c++17 -Wall -Wextra
↪  main.cpp -o main
./main
```

Windows — MinGW / GCC

```
gcc -std=c17 -Wall -Wextra main.c
↪  -o main.exe
main.exe

g++ -std=c++17 -Wall -Wextra
↪  main.cpp -o main.exe
main.exe
```

Ce se schimbă?

Comenzile și extensia executabilului diferă, dar ideea este aceeași: compilatorul transformă sursa într-un executabil pentru platforma curentă.

Ce ascunde un IDE?

Butonul **Run** sau **Build** nu elimină compilarea; doar o pornește automat.

1. IDE-ul salvează fișierele sursă;
2. apelează compilatorul cu anumite opțiuni;
3. afișează erorile și avertismentele;
4. creează sau actualizează executabilul;
5. pornește programul și afișează ieșirea.

Exercitiu

In IDE-ul pe care il folosiți, identificați ce comenzi se execută pentru a compila fișierul sursă, care este fișierul executabil rezultat și cum se rulează acesta.

Versiuni și standarde C++

C++ evoluează prin standarde succesive. Codul vechi rămâne important, dar proiectele noi aleg de obicei un standard explicit.

Standard	C++98	C++03	C++11	C++14	C++17	C++20/
Ideea	bază	corecții	modernizare	rafinare	biblioteci	funcții n

- C++11 introduce multe facilități moderne: `auto`, lambda, `range-based for`;
- C++17 și C++20 sunt alegeri frecvente în cursuri și proiecte;
- C++23 este standard publicat, iar următoarea versiune este încă în dezvoltare.

Alegerea standardului la compilare

GCC / MinGW

```
g++ -std=c++17 -Wall -Wextra main.cpp -o main
g++ -std=c++20 -Wall -Wextra main.cpp -o main
```

- –std=c++17 alege versiunea de limbaj;
- –Wall –Wextra activează avertismente utile;
- fără –std, compilatorul folosește o valoare implicită care poate diferi între sisteme;

Primul program în C++

main.cpp

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!\\n";
    return 0;
}
```

```
$ g++ main.cpp -o main
$ ./main
Hello, world!
```

Sintaxă și structură

Reguli de sintaxă

- C și C++ sunt case-sensitive: `var` și `Var` sunt identificatori diferiți.
- Instrucțiunile se termină cu punct și virgulă.
- Blocurile sunt delimitate de `{` și `}`.
- Spațiile și indentarea ajută cititorul, dar nu înlocuiesc acoladele.
- Comentarii: `// până la sfârșitul liniei` și `/* pe mai multe linii */`.

Convenție utilă

O instrucțiune pe linie și indentare consecventă: compilatorul verifică sintaxa, oamenii verifică lizibilitatea.

Instrucțiuni și expresii

Exemplu

```
int x = 2 + 3;        // declarație + inițializare
x = x * 4;            // atribuire
printf("%d\\n", x);   // apel de funcție
```

- o **expresie** produce o valoare;
- o **instrucțiune** descrie o acțiune și se termină, de obicei, cu ;;
- o secvență de instrucțiuni formează un bloc executat în ordine.

Variabile și tipuri

Variabile: declarație și inițializare

O variabilă are un tip, un nume și o zonă de memorie.

Exemplu

```
int age;              // declarație
age = 19;             // atribuire ulterioară
double price = 12.5; // declarație + inițializare
char grade = 'A';
```

- în C/C++, variabilele locale trebuie inițializate înainte de a fi citite;
- numele trebuie să înceapă cu literă sau _ și poate conține cifre;
- cuvintele rezervate, de exemplu int și return, nu pot fi nume de variabile.

Declarație, atribuire, inițializare

Acțiune	Exemplu	Ce se întâmplă
Declarație	`int n;`	se stabilește tipul lui n
Inițializare	`int n = 5;`	declarație și prima valoare
Atribuire	`n = 8;`	valoarea veche este înlocuită
Expresie	`n = n + 1;`	se calculează partea dreaptă

Într-o atribuire, expresia din dreapta se evaluează înainte ca rezultatul să fie stocat în stânga.

Tipuri numerice și caractere

Tipuri uzuale

```
char letter = 'A';          // un caracter
int count = 42;             // întreg
long long population = 8000000000LL;
float ratio = 0.5f;         // precizie simplă
double distance = 12.75; // precizie dublă
```

- dimensiunea exactă depinde de platformă; sizeof o poate măsura;
- char este un tip numeric mic, folosit frecvent pentru caractere;
- pentru calcule cu zecimale, double este alegerea uzuală.

Dimensiunea tipurilor uzuale

Tip	Dimensiune uzuală	Observații
char	1 byte	întotdeauna un byte C/C++
short	2 bytes	
int	4 bytes	
long	4 sau 8 bytes	Windows 64-bit sau Linux/macOS 64-bit
long long	8 bytes	
float	4 bytes	
double	8 bytes	
long double	8, 12 sau 16 bytes	depinde de compilator și platformă
bool	1 byte	de obicei

Valorile sunt uzuale, nu garantate. `sizeof(T)` măsoară dimensiunea tipului `T` în bytes pe sistemul curent; tipurile întregi signed și unsigned au aceeași dimensiune.

Tipuri logice și constante

C

```c
#include <stdbool.h>
bool ready = true;
const int DAYS = 7;
```

În C++ putem scrie direct aceleași declarații:

C++

```cpp
bool ready = true;
const int DAYS = 7;
```

- o variabilă const nu poate fi modificată după inițializare;
- true și false reprezintă valori logice.

Conversii de tip

Conversie implicită și explicită

```
int a = 5, b = 2;
double exact = (double)a / b; // 2.5
int truncated = a / b;        // 2
int rounded_down = (int)3.9;  // 3
```

- operația a / b este întreagă dacă ambele valori sunt întregi;
- o conversie poate schimba precizia sau valoarea;

Operatori și expresii

Operatori aritmetici

Exemplu

```
int a = 17, b = 5;
a + b;   // 22
a - b;   // 12
a * b;   // 85
a / b;   // 3
a % b;   // 2
-a;      // -17
```

- % este restul împărțirii întregi;
- ++x incrementează înainte de folosire, x++ după folosire;
- parantezele fac intenția explicită: (a + b) ∗ c.

Operatori relaționali și logici

Exemplu

```
int age = 20;
bool adult = age >= 18;
bool valid = age >= 0 && age <= 120;
bool special = age == 18 || age == 21;
bool not_adult = !adult;
```

- relaționali: < <= > >= == !=;
- logici: && (și), || (sau), ! (negație);
- atenție: = atribuie, iar == compară.

Operatorul condițional ?:

Operatorul ternar alege una dintre două expresii în funcție de o condiție:

```
condiție ? valoare_dacă_adevărat : valoare_dacă_fals
```

Exemplu: valoarea mai mare

```c
int a = 7, b = 4;
int maxim = (a > b) ? a : b;
printf("%d\\n", maxim); // 7
```

Este o formă compactă de alegere; pentru mai multe instrucțiuni, folosește `if` și `else`.

Atribuiri compuse și precedență

Exemplu

```
int x = 10;
x += 5;     // x = x + 5
x *= 2;     // x = x * 2
x -= 4;
x /= 2;
```

Ordinea uzuală: paranteze → unari → multiplicare/împărțire → adunare/scădere → comparații → operatori logici → atribuire.

Regulă practică

Când o expresie devine greu de citit, folosește paranteze sau variabile intermediare.

Precedența operatorilor

Precedență	Operatori (de sus în jos: mare spre mică)	Asociativitate
1	`::`	stânga spre dreapta
2	`a++ a- T(a) T{a} a() a[] a.b a->b`	stânga spre dreapta
3	`++a -a +a -a !a ~a *a &a sizeof new[] delete[] (T)a co_await`	dreapta spre stânga
4	`a.*b a->*b`	stânga spre dreapta
5	`* / %`	stânga spre dreapta
6	`+ -`	stânga spre dreapta
7	`« »`	stânga spre dreapta
8	<=> (C++20)	stânga spre dreapta
9	`< <= > >=`	stânga spre dreapta
10	`== !=`	stânga spre dreapta
11	& (AND pe biți)	stânga spre dreapta
12	^ (XOR pe biți)	stânga spre dreapta
13	| (OR pe biți)	stânga spre dreapta
14	&& (AND logic)	stânga spre dreapta
15	|| (OR logic)	stânga spre dreapta
16	`?:  throw co_yield = += -= *= /= %= «= »= &= ^= |=`	dreapta spre stânga
17	,	stânga spre dreapta

Precedența stabilește gruparea expresiei, nu ordinea evaluării.

Asociativitatea operatorilor

Precedența grupează operatorii cu niveluri diferite. Dacă operatorii au aceeași precedență, asociativitatea arată cum se grupează expresia.

Stânga spre dreapta
`a - b - c`
se grupează ca
`(a - b) - c`

Dreapta spre stânga
`a = b = c`
se grupează ca
`a = (b = c)`

Exemplu: precedența într-o expresie

Expresie

```
int a = 2, b = 3;
int ok1, ok2;
ok1 = ok2 = a+++b*4, ++b*(a+4);
```

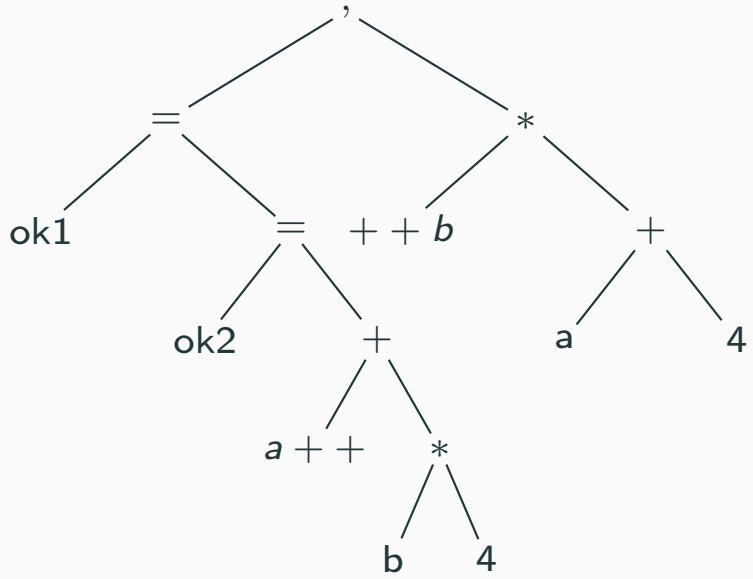
Exemplu: precedența într-o expresie

Expresie

```
int a = 2, b = 3;
int ok1, ok2;
ok1 = ok2 = a+++b*4, ++b*(a+4);
```

Operator	Precedență
postfix a++	2
prefix ++b	3
multiplicare *	5
adunare +	6
atribuire =	16
virgulă ,	17

Arborele de grupare



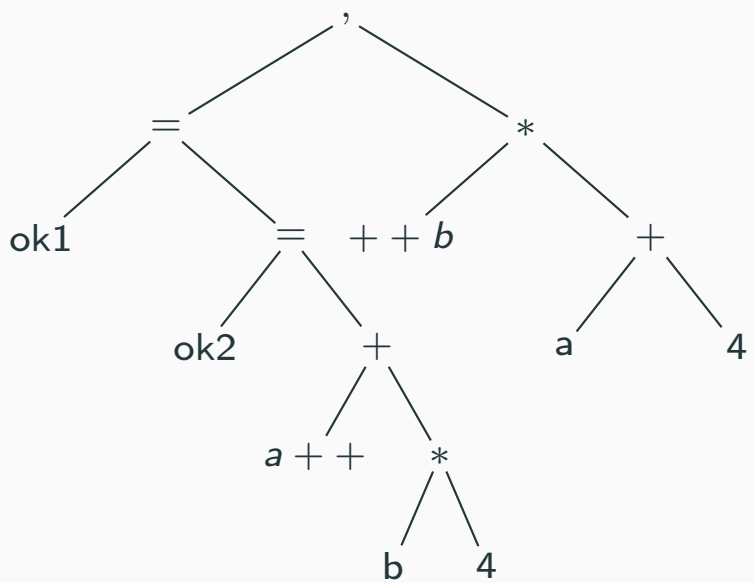
Exemplu: precedența într-o expresie

Expresie

```
int a = 2, b = 3;
int ok1, ok2;
ok1 = ok2 = a+++b*4, ++b*(a+4);
```

Operator	Precedență
postfix a++	2
prefix ++b	3
multiplicare *	5
adunare +	6
atribuire =	16
virgulă ,	17

Arborele de grupare



Arborele arată gruparea, nu ordinea tuturor evaluărilor. Virgula garantează că ramura stângă se termină prima: produce 14, a devine 3, iar ok2 și apoi ok1 primesc 14. Abia apoi ramura dreaptă crește b la 4 și calculează 28, care se ignoră. Ordinea operanzilor fiecărui + și * nu este garantată.

Interacțiune și control

Afișarea datelor în C

printf

```c
#include <stdio.h>
int n = 12;
double x = 3.14159;
printf("n = %d\\n", n);
printf("x = %.2f\\n", x);
printf("litera = %c\\n", 'A');
```

- %d pentru int, %f pentru double, %c pentru char;
- \\n înseamnă linie nouă;
- formatul trebuie să corespundă tipului valorii.

Opțiuni de formatare pentru `printf`

Un specificator are forma
`%[flags][width][.precision][length]specifier`.

Specificatori uzuali

`%d`, `%i`	`int` cu semn
`%u`	`unsigned int`
`%ld`, `%lld`	`long`, `long long`
`%f`, `%e`, `%g`	număr real (`double`)
`%c`	caracter
`%s`	șir C terminat cu `\0`
`%p`	adresă
`%%`	caracterul %

Lățime, precizie și flaguri

`-`	aliniere la stânga
`+`	afișează semnul numărului
`0`	completează lățimea cu zerouri
`8`	lățime minimă de 8 caractere
`.2`	2 zecimale la `%f`; cel mult 2 bytes la `%s`
`*`	lățimea/precizia vine dintr-un argument `int`
`#`	formă alternativă, de ex. prefix la hexazecimal

Exemplu

```
printf("%-8s | %05d | %.2f\\n", "mere", 7, 3.5);
// mere     | 00007 | 3.50
```

Lățimea este minimă; fiecare specificator trebuie să corespundă argumentului.

Citirea datelor în C și C++

C

```c
int age;
scanf("%d", &age);
printf("%d\\n", age);
```

C++

```cpp
int age;
std::cin >> age;
std::cout << age << '\n';
```

De ce &age în C?

scanf trebuie să primească adresa variabilei în care va scrie valoarea.

Decizii cu if

Structura

```
if (condition) {
    // instrucțiuni dacă este adevărat
} else {
    // instrucțiuni în caz contrar
}
```

Exemplu

```
if (number % 2 == 0) {
    printf("par\\n");
} else {
    printf("impar\\n");
}
```

Condiția este între paranteze, iar blocul este între acolade.

Repetiții: while și for

while

```c
int i = 1;
while (i <= 3) {
    printf("%d ", i);
    i++;
}
```

for

```c
for (int i = 1; i <= 3; i++) {
    printf("%d ", i);
}
```

- while este potrivit când condiția controlează numărul de pași;
- for grupează inițializarea, condiția și actualizarea.

While și for: exemple mai complexe

Doi indici într-un for

Verificare palindrom

```
int pal = 1;
for (int l = 0, r = n - 1;  l < r; ++l, --r) {
    if (t[l] != t[r]) {
        pal = 0;
        break;
    }
}
```

- Căutarea cu doi indici presupune că values este sortat.
- În for, inițializarea și actualizarea pot gestiona mai multe variabile; condiția se verifică înaintea fiecărei iterații.

While și for: descompunerea unui număr în cifre

Pentru fiecare pas, n % 10 extrage ultima cifră, iar n /= 10 o elimină.

Cu while

```
int n;
scanf("%d", &n);
while (n > 0) {
    int cifra = n % 10;
    printf("%d ", cifra);
    n /= 10;
}
```

Cu for

```
int n;
for (scanf("%d", &n); n > 0; n /= 10) {
    int cifra = n % 10;
    printf("%d ", cifra);
}
```

- Dacă citim 2026, ambele variante afișează 6 2 0 2.
- while evidențiază actualizarea în corp; for o pune în antet.