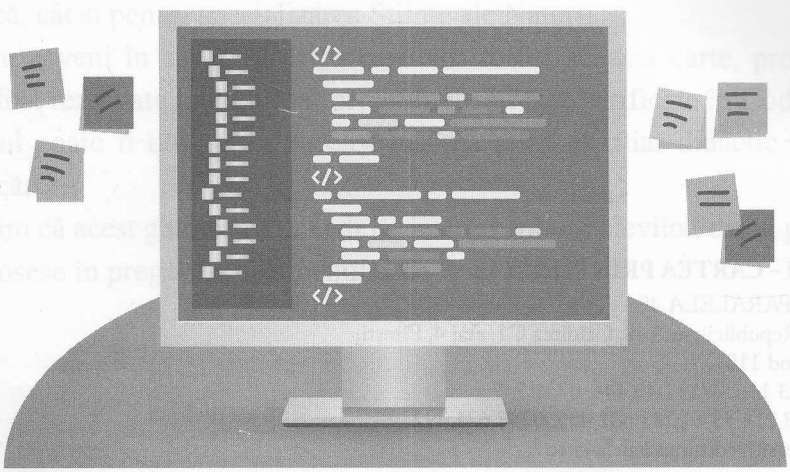
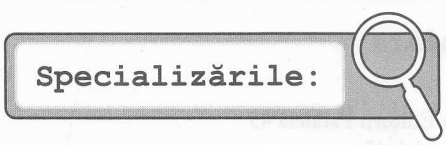
SILVIA GRECU
LUCIA MIRON
MIRELA ȚIBU

BACALAUREAT INFORMATICĂ
LIMBAJUL C++

Ghid complet de pregătire a examenului de Bacalaureat





{ Matematică-Informatică
  Științe ale Naturii
}

Editura Paralela 45

CUPRINS

CAPITOLUL 1

Reprezentarea algoritmilor în pseudocod. Elemente de bază ale limbajului C++

Teorie

Principiul programării structurate: Orice algoritm poate fi descris utilizând trei tipuri de structuri de control: liniară, alternativă și repetitivă.

Structuri de control – reprezentare în pseudocod	Implementare C++
• **Structura liniară** – conține operații de citire/scriere/atribuire. citește var_1, var_2, ..., var_n var_1, var_2, ... sunt identificatori de variabile scrie $expresie_1$, $expresie_2$, ..., $expresie_n$ $expresie_1$, $expresie_2$, var_2, ... sunt expresii de orice tip (numeric, caracter sau șir de caractere) *variabila* ← *expresie* Se evaluează expresia din partea dreaptă, iar valoarea obținută i se atribuie variabilei cu identificatorul *variabila.*	Citirea valorilor variabilelor de la tastatură: cin ≫ var_1 ≫ var_2 ≫ ⋯ ≫ var_n; //valorile variabilelor se citesc în ordinea din listă Afișarea valorilor expresiilor pe ecran: cout ≪ $expresie_1$ ≪ ⋯ ≪ $expresie_n$; //expresiile se vor evalua și afișa în ordinea din listă variabila = expresie; variabila < *op* > = expresie; unde *op* ∈ {+, −, *, /, %, ≫, ≪ , &, |, ^}, este echivalentă cu *variabila = variabila <op> expresie;*
Structura alternativă – conține operații de decizie: dacă <*expresie*> atunci *secvența_A* altfel *secvența_B*	Instrucțiunile *if* și *switch* implementează structura alternativă. if (*expresie*) instrucțiuni_A else instrucțiuni_B **Efect:** Se evaluează *expresie*. Dacă valoarea este diferită de 0, se execută secvența de

Sau

```
dacă <expresie> atunci
    secvența_A
```

Efect: Se evaluează *expresie*. Dacă este adevărată, se execută *secvența_A*, altfel se execută *secvența_B* și se continuă algoritmul cu următoarele structuri.

Secvențele A și/sau B pot conține orice alte structuri liniare, alternative sau repetitive.

Ramura altfel poate lipsi.

instrucțiuni_A, altfel se execută secvența de instrucțiuni_B.

În cazul în care o secvență conține mai mult de o instrucțiune, se va delimita prin { } grupul de instrucțiuni care definesc secvența.

Instrucțiunea *switch* pentru alternative multiple, simplifică modul de scriere pentru if-urile imbricate.

```
switch (expresie) {
    case constantă_1: instrucțiuni_1
                      break;
    case constantă_2: instrucțiuni_2
                      break;
    .....
    case constantă_n: instrucțiuni_n
                      break;
    default: instrucțiuni
    }
```

Efect: Valoarea *expresie* este evaluată la un tip întreg, apoi această valoare este comparată cu fiecare constantă. Este rulat blocul de instrucțiuni al valorii găsite.

Dacă este prezentă instrucțiunea *break* după blocul executat, se va părăsi instrucțiunea *switch*, altfel se vor executa în continuare toate blocurile de instrucțiuni ale constantelor care urmează, până la întâlnirea primului *break* sau până la sfârșitul instrucțiunii *switch*.

În caz că numărul nu este egal cu niciuna dintre constante, este executat blocul aflat după *default*.

Structura repetitivă

- **Cu condiție inițială**

```
cât timp <expresie> execută
    secvența_A
```

Efect:

Pas 1. Se evaluează *expresie*.

Pas 2. Dacă este diferită de 0, se execută *secvența_A* și se reia Pas 1. Dacă este 0 se părăsește structura repetitivă și se continuă algoritmul cu următoarele structuri.

Instrucțiunea **while** implementează structura repetitivă cu condiție inițială.

```
while (expresie)
{
    //declarări de date locale
    Instrucțiuni
}
```

Variabilele declarate în blocul instrucțiunii *while* sunt vizibile doar în acest bloc și sunt valabile doar pe durata execuției instrucțiunii.

Obs. 1) Dacă la prima evaluare, valoarea *expresiei* este 0, *secvența_A* nu se execută deloc.

2) În *secventa_A* trebuie să se modifice valoarea cel puțin a unei variabile care apare în *expresie* astfel încât, după un număr finit de pași, execuția structurii *cât timp* să se încheie.

- **Cu condiție finală**

repetă
    secvența_A
■ până când *<expresie>*

Efect:

Pas 1. Se execută *secvența_A*.

Pas 2. Se evaluează *expresie*.

Pas 3. Dacă valoarea este diferită de 0 (este adevărată), se părăsește structura repetitivă și se continuă algoritmul cu următoarele structuri. Dacă valoarea este egală cu 0, se reia Pasul 1.

Obs. 1) *secvența_A* se execută cel puțin o dată, indiferent de valoarea inițială a *expresiei*.

2) În *secvența_A* trebuie să se modifice valoarea cel puțin a unei variabile care apare în *expresie* astfel încât, după un număr finit de pași, execuția structurii *cât timp* să se încheie.

- **Cu număr cunoscut de pași**

pentru contor ← start, stop, pas execută
    secvența_A
■

Obs. Dacă *pas* nu este precizat, implicit va avea valoarea 1.

Efect:

Pas 1. Se atribuie variabilei *contor* valoarea de start (evaluată ca număr întreg)

Pas 2. Se compară valoarea *contor* cu valoarea stop, în funcție de semnul pasului. Dacă pasul este 1 (parcurgere crescătoare) atunci compararea va fi *contor≤stop*, iar dacă pasul este –1 (parcurgere descrescătoare), atunci compararea va fi *contor≥stop*

Instrucțiuni pot fi orice instrucțiuni C++, inclusiv dintre cele care implementează structuri repetitive.

Instrucțiunea **do-while** implementează structura repetitivă cu condiție finală.

```
do {
        //declarări de date locale
        Instrucțiuni
} while (not expresie);
```

Atenție! Condiția de oprire a buclei, exprimată de *expresie* din descrierea în pseudocod, se transformă în condiție de repetare a buclei, în instrucțiunea din C++. De aceea apare negația (not *expresie*).

Variabilele declarate în blocul instrucțiunii *do-while* sunt vizibile doar în acest bloc și sunt valabile doar pe durata execuției instrucțiunii.

Instrucțiuni pot fi orice instrucțiuni C++, inclusiv dintre cele care implementează structuri repetitive.

Instrucțiunea *for* implementează structura repetitivă cu număr cunoscut de pași *pentru*.

Dacă *pas* este 1

```
for (contor=start; contor<=stop;
contor++)
{
    Instrucțiuni
}
```

Pas 3. Dacă este adevărată evaluarea expresiei corespunzătoare *contor≤stop* (pentru pas = 1) sau *contor≥stop* (pentru pas = –1) se execută *secvența_A* și se trece la Pas 4.
Dacă este falsă, se părăsește structura repetitivă *pentru* și se continuă algoritmul cu următoarele structuri.
Pas 4. La valoarea *contor* se adună valoarea *pas* și se reia Pas 2.

Dacă *pas* este –1

```
for (contor = start; contor>=stop;
contor - -)
{
    Instrucțiuni
}
```

Sintaxa C++ a instrucțiunii **for** este

```
for (expresie1; expresie2; expresie3)
{
        Instrucțiuni
}
```

și este echivalentă cu

```
expresie1
while (expresie2)
{
        Instrucțiuni
        expresie3
}
```

Echivalența structurilor de control repetitive

Structura de control	Structuri de control echivalente
cât timp *<expresie>* execută *secvența_A*	dacă *<expresie>* atunci repetă *secvența_A* până când ***<not expresie>***
	Atenție! Condiția de repetare exprimată prin *expresie* în structura *cât timp* devine condiție de repetare în structura *repetă-până când*, de aceea apare negația.
repetă *secvența_A* până când *<expresie>*	*secvența_A* cât timp ***<not expresie>*** execută *secvența_A*
	Atenție! Condiția de oprire exprimată prin *expresie* în structura *repetă-până când* devine condiție de repetare în structura *cât timp*, de aceea apare negația.

pentru contor ← start, stop execută *secvența_A*	contor←start cât timp *contor ≤ stop* execută *secvența_A* contor←contor+1
pentru contor ← start, stop execută *secvența_A*	contor←start dacă *contor ≤ stop* atunci repetă *secvența_A* contor←contor+1 până când *contor > stop*
pentru contor ← start, stop, –1 execută *secvența_A* pentru contor ← start, stop, –1 execută *secvența_A*	contor←start cât timp *contor ≥ stop* execută *secvența_A* contor←contor–1 contor←start dacă *contor ≥ stop* atunci repetă *secvența_A* contor←contor–1 până când *contor < stop*
cât timp *val>0* execută *secvența_A* val←val–1	pentru val ← val, 1, –1 execută *secvența_A* sau pentru contor ← val, 1, –1 execută *secvența_A*

Atenție la a doua implementare! În cazul în care *secvența_A* conține instrucțiuni care utilizează valoarea *val*, trebuie înlocuite toate referirile la val din *secvența_A* cu *contor*.

Elemente de bază ale limbajului C/C++

Datele prelucrate de un program sunt *constante* și *variabile*. Unei variabile i se poate modifica valoarea pe parcursul execuției blocului în care este declarată, în timp ce unei constante nu i se poate schimba valoarea.

O *variabilă* este caracterizată prin nume (identificator de variabilă), tip, spațiu de memorie alocat și valoare. Spațiul de memorie alocat (numărul de octeți) și domeniul valorilor posibile depind de tipul variabilei.

Identificatorul unei variabile poate fi format din litere 'a'...'z', 'A'...'Z', cifre '0'...'9' și simbolul '_' și nu poate începe cu o cifră.

Corect: _3ab, max2, M_aux, p123 **Incorect:** 1aB, 5_sum

În limbajul C/C++ pot fi reprezentate tipuri de date **simple** și **structurate** (omogene și neomogene).

Tipuri de date simple

Tip variabilă	Nr. octeți (bytes)	Domeniu de valori	Operatori	Funcții specifice
short int unsigned short int int = long int unsigned int = unsigned long int long long int unsigned long long	2 2 4 4 8 8	$[-2^{15}, 2^{15}-1]$ $[0, 2^{16}-1]$ $[-2^{31}, 2^{31}-1]$ $[0, 2^{32}-1]$ $[-2^{63}, 2^{63}-1]$ $[0, 2^{64}-1]$	Operatori: Aritmetici: +, −,*,/, % Pe biți: ≫, ≪, &, |, ^, ~ Relaționali: <,<=,>=, > De egalitate: ==, !=	<cmath.h> sqrt(x), x≥0, pentru $\sqrt{x}$ abs(x) pentru |x| pow(a,b) pentru a^b
float double	4 8	$[-3.2 \cdot 10^{38}, 3.2 \cdot 10^{38}]$ $[-1.7 \cdot 10^{308}, 1.7 \cdot 10^{308}]$	Operatori: Aritmetici: +, −,*,/ Relaționali: <,<=,>=, > De egalitate: ==, !=	<cmath.h> sqrt(x), x≥0, pentru $\sqrt{x}$ abs(x) pentru |x| pow(a,b) pentru a^b
char unsigned char	1 1	−128, 127] [0, 255]	Operatori: Aritmetici: +, −,*,/,% Relaționali: <,<=,>=, > De egalitate: ==, !=	<ctype.h> islower(car) = { *1, car e literă mică*; *0, altfel* } isupper(car) = { *1, car e majusculă*; *0, altfel* } tolower(car) = *car*
				în minusculă = car +'a'-'A' toupper(car) = *car* în majusculă = car -'a'+'A'

Limbajul C/C++ permite realizarea de conversii implicite sau explicite de tip a varibilelor și constantelor.

Situații care determină efectuarea implicită a unei conversii într-o expresie:

– dacă operanzii sunt de același tip real sau întreg, dar se memorează în locații cu dimensiuni diferite, conversia se efectuează spre locația cu dimensiune maximă;
– dacă operanzii sunt de tip diferit, întreg și real, conversia se va aplica numerelor întregi care devin numere reale;
– dacă un operand de tip **char** intervine într-o expresie aritmetică se va realiza conversia către tipul întreg, iar valoarea cu care se va evalua expresia va fi codul ASCII al caracterului;
– dacă valoarea reală a unei variabile sau a unei expresii este atribuită unei variabile întregi, se va realiza o conversie implicită spre tipul întreg care va determina „trunchierea" părții zecimale a numărului real.

Exemplul 1

```
.....
int a=12, b, ok ;
long long int n;
char ch1='A', ch2;
ch2=ch1+3;
//codul ASCII('A')=65, 65+3=68 si ch2='D', codul ASCII('D')=68

b=a%7*1.5;
//se efectuează 12%7=5, apoi 5*1.5=7.5, care va fi trunchiat la partea întreagă 7, apoi atribuit lui b

ok=(a<b);
//se vor compara cele două valori ale variabilelor a și b (12<7) și se va atribui lui ok valoarea 0
cout <<ch2<<' '<<b<<' '<<ok;          //se vor afișa: D 7 0
```

Exemplul 2

```
.....
float x, y;
int a=7, b=2;
x=a/2+5.0;
          //se efectuează 7/2=3 (operanzi întregi) apoi 3+5.0=8.0 care se va atribui lui x
y=(float)a/2+3.0;      //se efectuează 7.0/2=3.5 apoi 3.5+3.0=6.5 care se va atribui lui y
cout <<x<<' '<<y;      //se vor afișa: 8   6.5
.....
```

Categorie	Operatori	Semnificație	Asociativitate
1. Prioritate unari de prioritate maximă	(), [], →, .	Apel de funcție Expresie cu indici Selectori de membru la structuri	St-Dr
2. Operatori unari	! ~, +, - ++, --, &, * sizeof(tip) (cast)	Negare logică Negare bit cu bit (complementare cu 1) Plus și minus unari Incrementare/decrementare (pre și post) Obținerea adresei/indirectare Dimensiune operand (în octeți) Conversie explicită de tip – cast	Dr-St
3. Operatori aritmetici multiplicativi	*, /, %	Înmulțire/împărțire/restul împărțirii întregi	St-Dr
4. Adunare, scădere	+, -	Plus și minus binari	St-Dr
5. Deplasări	<<, >>	Deplasare stânga/dreapta pe biți	St-Dr
6. Relaționali	<, <=, >, >=	Mai mic/ Mai mic sau egal/ Mai mare/ Mai mare sau egal	St-Dr
7. Egalitate	==, !=	Egal/ Diferit	St-Dr
8-10. Operatori logici pe biți	&, ^, |	ȘI/ SAU EXCLUSIV/ SAU logic bit cu bit	St-Dr
11-12. Operatori logici	&&, ||	ȘI/ SAU logic	St-Dr
13. Operator de atribuire compus	= <op>=	Atribuire simplă: variabila = expresie; Atribuire multiplă variabila1 = variabila2 = ... = expresie; Atribuire compusa: variabila < *op* >= expresie unde *op* ∈ {+, −, *,/, %, ≫, ≪, &, |, ^}, este echivalentă cu *variabila = variabila < op > (expresie);*	Dr-St
14. Operator condițional	?:	*expresieTest? variantaDa : variantaNu;* Se evaluează *expresieTest*. Dacă este adevărată (diferită de 0), se evaluează doar *expresieDa*, altfel, se evaluează doar *expresieNu*.	Dr-St
15. Virgula	,	*expresie1, expresie2,...expresieN;* Se evaluează expresiile în ordine iar valoarea finală este cea a *expresieN*	St-Dr

Expresiile sunt succesiuni de operatori și operanzi, corecte din punct de vedere sintactic. Orice expresie are o valoare, obținută la evaluarea acesteia.
În funcție de tipul operanzilor și al operatorilor, expresiile pot fi:
– aritmetice: valoarea obținută la evaluare este întreagă sau reală.
– logice: valoarea obținută la evaluare este 0 (fals) sau 1 (adevărat).
La evaluarea expresiilor logice se utilizează tabelele de valori ale operatorilor logici.

! (not), && (and), || (or)

!	0	1
	1	0

&&	0	1
0	0	0
1	0	1

||	0	1
1	0	1
0	1	1

În cazul expresiilor logice compuse se pot aplica regulile lui de Morgan:

!(E1 && E2) = !(E1) || !(E2)

!(E1 || E2) = !(E1) && !(E2)

unde E1 și E2 sunt expresii logice sau aritmetice.

Structura unui program C++

```
#include <iostream.h>
        //includerea altor librării necesare în program
using namespace std;
        //declarare variabile globale
        //declarare funcții utilizator
int main()
{
        //declarare variabile locale
        instrucțiuni
        return 0;
}
```

PROBLEME PROPUSE

1. Variabilele a și b reprezintă numere întregi nenule. Care dintre expresiile C/C++ au valoarea **1** dacă și numai dacă **a** și **b** au același semn și sunt impare?
   1) a+b>0 && a%2!=0 && b%2!=0
   2) !(a%2==0 || b%2==0) && !(a*b<0)
   3) a%2+b%2==2 && a*b>0
   4) (a%2>0 && b%2>0) || !(a%2+b%2<2)
2. Variabila **n** reprezintă un număr întreg nenul, iar variabilele **a** și **b** reprezintă numere reale. Care dintre următoarele expresii C/C++ au valoarea **1** dacă și numai dacă **n** este multiplu de **2019** și **nu** aparține intervalului **(a, b)**?
   1) n%2019==0 && n<a || n>b
   2) !(n%2019 || !(n<=a && n>=b))

3) floor(a)>=abs(n) && !(abs(n)<ceil(b) || n%2019!=0)
4) a>=n || b<=n && (n%3==0 && n%673==0)

3. Un număr natural este palindrom dacă este egal cu oglinditul său (De exemplu: **202**, **1881** sunt numere palindrom, iar **2012 nu** este palindrom). Știind că în variabila **x** este memorat un număr natural de exact 5 cifre, stabilește care dintre următoarele expresii C/C++ are valoarea **1** dacă și numai dacă numărul **x** este palindrom.

a) x%100==x/100/10
b) x/10%10+x%10*10==x/100/10
c) x/1000%10==x/100%10 && x/10000==x%10
d) x/10000==x/10%10 && x/1000==x%100

4. Știind că **x** este o variabilă care memorează un număr întreg nenul, precizează care este cea mai mică valoare ce se poate obține la evaluarea următoarei expresii C/C++: 200%x - x%5.

a) -4 **b)** 0 **c)** -203 **d)** -195

5. Care este rezultatul ce se obține la evaluarea expresiei C/C++ de mai jos?
(5*4%7-19/(7-4%5))/2

a) 1 **b)** 0 **c)** 1.5 **d)** 1.1416

6. Se consideră următoarea secvență de instrucțiuni C-C++. Precizează câte operații de atribuire se vor efectua la execuția acesteia.

```
int n = 1023, k = 2;
while (n>0 || k>0)
{
   n = n / 10;
   k = k - 1;
}
```

a) 10 **b)** 6 **c)** 8 **d)** 12

7. Se consideră **x** și **y** variabile care memorează numere naturale și următoarele 3 instrucțiuni de atribuire în C/C++. Precizează care este ordinea în care trebuie executate acestea, astfel încât să se realizeze interschimbarea valorilor celor două variabile.

1) x = y - x **2)** y = x + y **3)** y = y - x

a) 2, 3, 1 **b)** 1, 2, 3 **c)** 3, 1, 2 **d)** 3, 2, 1

8. Știind că **a**, **x** și **y** sunt variabile reale nenule, scrie instrucțiunea C/C++ corespunzătoare următoarei operații de atribuire: $a \leftarrow \frac{(x+y^2)\cdot(y-1)}{x^2\cdot y}$.

9. Știind că variabilele **x, y, z** memorează numere naturale, nenule și distincte două câte două, care dintre următoarele expresii C/C++ au valoarea **1** dacă și numai dacă numărul memorat în variabila **z** este fie multiplu de **x** și de **y**, fie este divizor al lui **x** și al lui **y**?

a) !(z%x || z%y) && x%z==0 && z%y==0
b) (z%x+z%y==0) || !(x%z || y%z)
c) (x%z==0 || y%z==0) && y%x==0 && z%y==0
d) z%(x*y)==0 || (x*y)%z==0