Editura Paralela 45

Redactare: Daniel Mitran
Tehnoredactare: Iuliana Ene
Pregătire de tipar: Marius Badea
Design copertă: Mirona Pintilie

Sursă foto interior: © Zdenek Sasek | Dreamstime.com

Descrierea CIP a Bibliotecii Naționale a României
GRECU, SILVIA
Memorator de informatică : limbajul C++ / Silvia Grecu, Lucia Miron, Mirela Țibu : clasele 9-12. – Ed. a 2-a. – Pitești : Paralela 45, 2023
ISBN 978-973-47-4033-8

I. Miron, Lucia
II. Țibu, Mirela

004

SILVIA GRECU LUCIA MIRON
MIRELA ŢIBU

MEMORATOR DE INFORMATICĂ

LIMBAJUL C++

Clasele 9-12

Ediţia a II-a

Editura Paralela 45

Cuprins

I. Elemente de bază ale limbajului C++ 7
II. Algoritmi elementari 17
III. Fişiere text 33
IV. Tablouri unidimensionale (Vectori) 35
V. Tablouri bidimensionale (Matrice) 43
VI. Şiruri de caractere 47
VII. Structuri de date neomogene 53
VIII. Subprograme (Funcţii) 57
IX. Funcţii recursive 63
X. Elemente de combinatorică 67
XI. Elemente de teoria grafurilor 73

I. ELEMENTE DE BAZĂ ALE LIMBAJULUI C++

Structura unui program C++

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

Citirea valorilor variabilelor de la tastatură / afişarea valorilor expresiilor pe ecran

```
cin>>var1>>var2>>...>>varn;
cout<<exp1<<' '<<exp2<<' '<<...<<expn;
```

Atribuirea

```
variabila = expresie;
variabila <op> = expresie;
   //unde op ∈ {+,-,*,/,%,≫,≪,&,|,^}, este
   echivalentă cu
   //variabila = variabila <op>(expresie);
```

Instrucţiunea *if*

```
a) if (expresie)
        instrucţiune_A;
   else
        instrucţiune_B;
b) if (expresie)
        instrucţiune_A;
```

Instrucţiunea *switch*

```
   switch (expresie) {
         case constantă_1: instrucţiuni_1
                           break;
         case constantă_2: instrucţiuni_2
                           break;
         ……
         case constantă_n: instrucţiuni_n
                           break;
        default: instrucţiuni
   }
```

Instrucţiunea *while*

```
while (expresie)
  {
    Instrucţiuni
  }
```

Instrucţiunea *do-while*

```
   do {
       Instrucţiuni
      } while (expresie);
```

Instrucţiunea *for*

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

`expresie1:` expresie de iniţializare;
`expresie2:` expresie de test;
`expresie3:` expresie de continuare.

Tipuri de date simple

Tip variabilă:
short int
Nr. octeți (bytes): 2 Domeniul de valori: $[-2^{15}, 2^{15} - 1]$
unsigned short int
Nr. octeți (bytes): 2 Domeniul de valori: $[0, 2^{16} - 1]$
int = long int
Nr. octeți (bytes): 4 Domeniul de valori: $[-2^{31}, 2^{31} - 1]$
unsigned int = unsigned long int
Nr. octeți (bytes): 4 Domeniul de valori: $[0, 2^{32} - 1]$
long long int
Nr. octeți (bytes): 8 Domeniul de valori: $[-2^{63}, 2^{63} - 1]$
unsigned long long int
Nr. octeți (bytes): 8 Domeniul de valori: $[0, 2^{64} - 1]$
Operatori:
Aritmetici: +, −, ∗, /, % Pe biți: ≫, ≪, &, |, ^, ~ Relaționali: <, <=, >=, > De egalitate: ==, !=
Funcții specifice:
<cmath.h> sqrt(x), $x \geq 0$, pentru $\sqrt{x}$ abs(x) pentru $\|x\|$ pow(a,b) pentru a^b

Tip variabilă:
float
Nr. octeți (bytes): 4 Domeniul de valori: $[-3.2 \cdot 10^{38}, 3.2 \cdot 10^{38}]$
double
Nr. octeți (bytes): 8 Domeniul de valori: $[-1.7 \cdot 10^{308}, 1.7 \cdot 10^{388}]$
Operatori:
Aritmetici: +, −, ∗, / Relaţionali: <, <=, >=, > De egalitate: ==, !=
Funcţii specifice:
<cmath.h> sqrt(x), x ≥ 0, pentru $\sqrt{x}$ abs(x) pentru |x| pow(a,b) pentru a^b

Tip variabilă:
char
Nr. octeți (bytes): 1 Domeniul de valori: $[-3.2 \cdot 10^{38}, 3.2 \cdot 10^{38}]$
unsigned char
Nr. octeți (bytes): 1 Domeniul de valori:
Operatori:
Aritmetici: +, −, ∗, /, % Relaţionali: <, <=, >=, > De egalitate: ==, !=

Funcţii specifice:
islower(car) = {1, dacă car e literă mică 0, altfel
isupper(car) = {1, car e literă majusculă 0, altfel
tolower(car) = *car* în minusculă = car+'a'–'A'
toupper(car) = *car* în majusculă = car–'a'+'A'

Operatori

1. Prioritate maximă:		
Operatori	*Semnificaţie*	*Asociativitate*
( ), [ ], → .	Apel de funcţie Expresie cu indici Selectori de membru la structuri	St-Dr
2. Operatori unari:		
Operatori	*Semnificaţie*	*Asociativitate*
! ~, +, - ++, --, &, * sizeof(tip) (cast)	Negare logică Negare bit cu bit (complementare cu 1) Plus şi minus unari Incrementare/decrementare (pre şi post) Obţinerea adresei/indirectare Dimensiune operand (în octeţi) Conversie explicită de tip – cast	Dr-St

3. Operatori aritmetici multiplicativi:		
Operatori	*Semnificație*	*Asociativitate*
*, /, %	Înmulțire/împărțire/restul împărțirii întregi	St-Dr
4. Adunare, scădere:		
Operatori	*Semnificație*	*Asociativitate*
+, -	Plus și minus binari	St-Dr
5. Deplasări:		
Operatori	*Semnificație*	*Asociativitate*
<<, >>	Deplasare stânga/dreapta pe biți	St-Dr
6. Relaționali:		
Operatori	*Semnificație*	*Asociativitate*
<, <=, >, >=	Mai mic/mai mic sau egal/Mai mare/ mai mare sau egal	St-Dr
7. Egalitate:		
Operatori	*Semnificație*	*Asociativitate*
==, !=	Egal/Diferit	St-Dr

8. 9. 10. Operatori logici pe biți:		
Operatori	*Semnificație*	*Asociativitate*
&, ^, |	SI/SAU EXCLUSIV/SAU logic bit cu bit	St-Dr
11. 12. Operatori logici:		
Operatori	*Semnificație*	*Asociativitate*
&&, ||	SI/SAU logic	St-Dr
13. Operatori de atribuire compuși:		
Operatori	*Semnificație*	*Asociativitate*
= <op>=	• Atribuire simplă: *variabila* = *expresie*; • Atribuire multiplă: *variabila1* = *variabila2* = ... = *expresie*; • Atribuire compusă: *variabila* <*op*> = *expresie*; unde *op* ∈ {+, – , *, /, %, >>, <<, &, |, ^}, este echivalentă cu *variabila* = *variabila* <*op*> *expresie*;	Dr-St
14. Operator condiţional:		
Operator	*Semnificație*	*Asociativitate*
?:	Operatorul condițional (ternar)	Dr-St

15. Virgula:		
Operator	*Semnificație*	*Asociativitate*
,	Evaluare *expresie1*, *expresie2*; Valoarea rezultatului este valoarea *expresie2*.	St-Dr

II. ALGORITMI ELEMENTARI

Algoritmi care prelucrează cifrele unui număr

Construirea oglinditului şi verificarea lui *n* dacă este palindrom

```
copie ← n
oglindit ← 0
┌cât timp n ≠ 0 execută
│   cif ← n%10
│   oglindit ← oglindit * 10 + cif
│   n ← [n/10]
└■
┌dacă copie=oglindit atunci
│   prelucrare_cifra(palindrom)
└■
```

Media aritmetică a cifrelor nenule

```
sum ← 0
nrcif ← 0
┌cât timp n ≠ 0 execută
│   cif ← n%10
│   ┌dacă cif>0 atunci
│   │       sum ← sum + cif
│   │       nrcif ← nrcif + 1
│   └■
│   n ← [n/10]
└■
```

```
┌dacă nrcif>0 atunci
│    medie ← sum/nrcif
└■
```

Cifra maximă din *n*

```
cifmax ← 0
┌cât timp n ≠ 0 execută
│   cif ← n%10
│   ┌dacă cif>cifmax
│   │    atunci
│   │      cifmax ← cif
│   └■
│   n ← [n/10]
└■
prelucrare(cifmax)
```

Eliminarea cifrelor impare din *n*

```
nrNou ← 0
p ← 1
┌cât timp n ≠ 0 execută
│  cif ← n%10
│  ┌dacă cif % 2 = 0 atunci
│  │     nrNou ← nrNou + cif *p
│  │     p ← p * 10
│  └■
│  n ← [n/10]
└■
prelucrare(nrNou)
```

Dublarea aparițiilor cifrelor pare

```
nrNou ← 0
p ← 1
┌cât timp n ≠ 0 execută
│   cif ← n%10
│  ┌dacă  cif % 2 = 0 atunci
│  │ nrNou ← nrNou + cif *p
│  │ p ← p * 10
│  └■
│   nrNou ← nrNou + cif *p
│   p ← p * 10
│   n ← [n/10]
└■
prelucrare(nrNou)
```

Numărarea cifrelor pare din *n*

```
nrPare ← 0
┌repetă
│   cif ← n%10
│  ┌dacă  cif % 2 = 0
│  │ atunci
│  │ nrPare ← nrPare + 1
│  └■
│   n ← [n/10]
└ până când n=0
prelucrare(nrPare)
```

Cifra de control a unui număr *n*

Cifra de control a unui număr *n* se obţine calculând suma cifrelor lui *n*, apoi repetând procesul cu cifrele sumei obţinute anterior până când se obţine un număr format dintr-o singură cifră, numită cifră de control. De exemplu, pentru *n* = 7912 se obţin pe rând sumele 7 + 9 + 1 + 2 = 19, 1 + 9 = 10, 1 + 0 = 1, iar 1 este cifra de control a lui 7912.

```
┌cât timp n > 9 execută
│   sumcif ← 0
│   ┌cât timp n ≠ 0 atunci
│   │     cif ← n % 10
│   │     sumcif ← sumcif + cif
│   │     n ← [n/10]
│   └■
│   n ← sumcif
└■
prelucrare(n)
```

Algoritmul eficient ca timp de execuţie se bazează pe observaţia că cifra de control a unui număr este periodică şi respectă relaţia:

$$cifControl(n) = \begin{cases} 0, \text{dacă } n = 0 \\ 9, \text{dacă } n\%9 = 0 \text{ şi } n \neq 0 \\ n\%9, \text{altfel} \end{cases}.$$

Divizibilitate. Algoritmi care prelucrează divizorii proprii/improprii/primi ai unui număr

Fie n un număr natural. Definim mulţimile:

Divizorii **proprii** ai lui n: $d \in \left\{2, 3, 4, ..., \left[\frac{n}{2}\right]\right\}$, $n \vdots d$.

Divizorii **primi** ai lui n: $d \in \left\{2, 3, 5, 7, ..., \left[\sqrt{n}\right]\right\}$, $n \vdots d$.

Divizorii **improprii** ai lui n: $d \in \{1, n\}$.

Divizorii proprii ai lui *n*

```
p ← 1
pentru d←2,[n/2],1 execută
    dacă  n % d = 0 atunci
        prelucrare (d)
```

Divizorii proprii ai lui *n* – optimizat

```
p ← 1
pentru d←2,[ √n]–1,1 execută
    dacă  n % d = 0 atunci
        prelucrare (d)
        prelucrare (n/d)

dacă  d * d = n  atunci
    prelucrare (d)
```