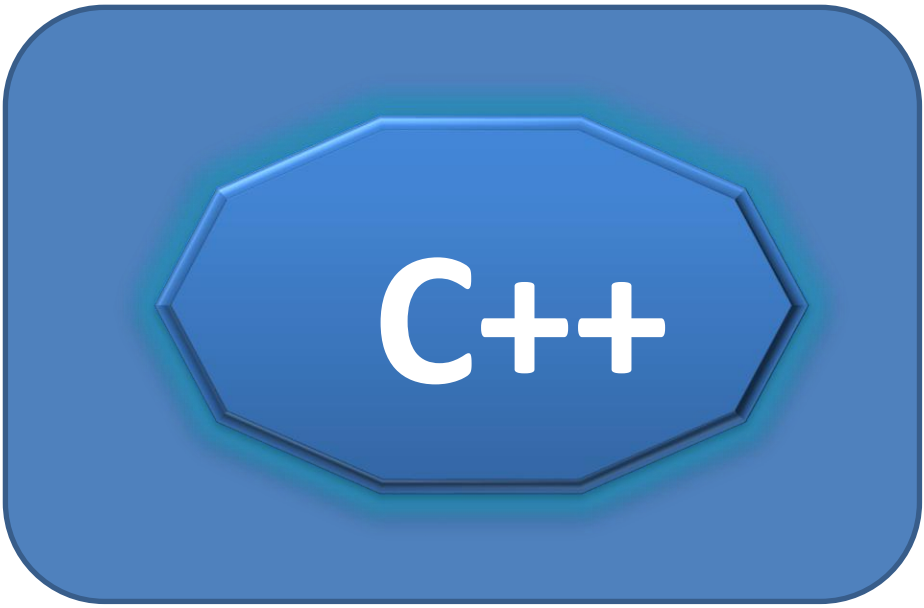
ELEMENTE DE BAZĂ ALE LIMBAJULUI C++



NELI SECITĂ

Bacău, 2023

NELI SECITĂ

ELEMENTE DE BAZĂ ALE LIMBAJULUI C++

Ediția a II-a (online)

Bacău, 2023

CUPRINS

CAPITOLUL 1

GENERALITĂŢI DESPRE AGORITMI 3

1.1. NOŢIUNI INTRODUCTIVE 3
1.2. CLASIFICAREA DATELOR 6
1.3. TIPUL DATEI 7
1.4. OBIECTELE CU CARE LUCREAZĂ ALGORITMII 8

CAPITOLUL 2

REPREZENTAREA ALGORITMILOR 11

2.1. MODURI DE REPREZENTARE A UNUI ALGORITM 11
2.2. VOCABULARUL LIMBAJULUI C++ 13
2.3. INSTRUCŢIUNILE DECLARATIVE 15
2.4. VARIABILE 18
2.5. CONSTANTE 19
2.6. DECLARAREA UNUI SET DE CONSTANTE 21
2.7. OPERAŢIILE DE CITIRE ŞI SCRIERE 21
2.8. EXPRESIA ŞI INSTRUCŢIUNEA EXPRESIE 23
2.9. OPERATORI 24
2.9.1. Operatori aritmetici 24
2.9.2. Operatorul de conversie explicit 25
2.9.3. Operatorii pentru incrementare şi decrementare 27
2.9.4. Operatorii relaţionali 29
2.9.5. Operatorii logici 30
2.9.6. Operatorii de atribuire 31
2.9.7. Operatorul condiţional 34
2.9.8. Operatorul virgulă 38
2.9.9. Operatorul dimensiune 39
2.9.10. Prioritatea operatorilor în C++ 40

CAPITOLUL 3

PRINCIPIILE PROGRAMĂRII STRUCTURATE 42

3.1. STRUCTURA LINIARĂ 42

3.2. STRUCTURA ALTERNATIVĂ (DE SELECŢIE) 43
3.2.1. Structura alternativă simplă 43
3.2.2. Structura alternativă multiplă 47
3.3. STRUCTURI REPETITIVE 51
3.3.1. Instrucțiunea repetitivă cu test inițial 51
3.3.2. Instrucţiunea repetitivă cu test final 53
3.3.3. Instrucţiunea repetitivă cu un număr cunoscut de paşi 55
3.4. TRANSFORMĂRILE STRUCTURILOR REPETITIVE 58

CAPITOLUL 4

ALGORITMI ELEMENTARI 60

4.1. ALGORITMI PENTRU INTERSCHIMBAREA A VALORILOR A DOUĂ NUMERE 60
4.2. ALGORITMI PENTRU DETERMINAREA MAXIMULUI / MINIMULUI 65
4.3. ALGORITMI PENTRU PRELUCRAREA CIFRELOR UNUI NUMĂR 67
4.3.1. Algoritmul pentru extragerea cifrelor unui număr 67
4.3.2. Algoritmul pentru compunerea unui număr din cifrele sale 70
4.3.3. Algoritmul pentru determinarea inversului unui număr 72
4.4. ALGORITMUL PENTRU CALCULAREA CMMDC 75
4.4.1. Algoritmul lui Euclid 76
4.4.2. Algoritmul de scădere repetată 76
4.5. ALGORITMUL PENTRU TESTAREA UNUI NUMĂR PRIM 78
4.6. ALGORITMII PENTRU PRELUCRAREA DIVIZORILOR AI UNUI NUMĂR 79
4.6.1. Algoritmul pentru generarea divizorilor proprii ai unui număr 79
4.6.2. Algoritmul pentru generarea divizorilor primi ai unui număr 80

BIBLIOGRAFIE 81

Capitolul 1

GENERALITĂŢI DESPRE AGORITMI

1.1. Noţiuni introductive

Etapele rezolvării unei probleme sunt:

1. Analiza problemei
2. Formularea modului de rezolvare a problemei
3. Codificarea modului de rezolvare a problemei într-un limbaj de programare
4. Testarea programului şi corectarea erorilor

Succesiunea etapelor rezolvării unei probleme, în ordinea parcurgerii lor, prin care se prelucrează un set de date de intrare, în scopul obţinerii unor date de ieşire se numeşte **algoritmul unei probleme.**

Orice algoritm trebuie să îndeplinească câteva **cerinţe fundamentale**:

1. **să fie cât mai general**, adică să acopere toate situaţiile care pot interveni în rezolvarea problemei respective

2. **să fie bine definit**, adică cerinţele problemei să fie formulate clar, fără ambiguităţi, iar datele de intrare de care dispune algoritmul să fie suficiente pentru rezolvarea problemei

3. **să fie finit**, adică la un moment dat algoritmul va încheia cu succes execuţia lui, obţinându-se toate soluţiile problemei

4. **să fie rezolvabil** adică orice algoritm trebuie să aibă cel puţin o soluţie

5. **să fie universal** – algoritmul trebuie să permită rezolvarea unei clase de probleme, care sunt de acelaşi tip şi care diferă între ele numai prin datele de intrare.

6. **să fie eficient** Operaţiile care compun algoritmul trebuie alese astfel încât soluţia problemei să fie obţinută după un număr minim de paşi

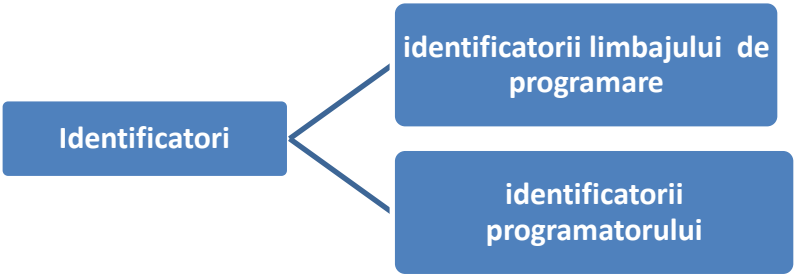
Un algoritm prelucrează **date.**

Data este un model de reprezentare a informaţiei, accesibil calculatorului, cu care poate opera pentru a obţine noi informaţii.

Din punct de vedere logic, data este definită prin tripletul:

Data elementară = (identificator, valoare, atribute)

Identificatorul reprezintă numele dat datei pentru a o putea distinge de alte date şi pentru a putea face referiri la ea în procesul de prelucrare a datelor.



Exemple:

- Identificatori ai limbajului C++: for, int, while, etc.
- Identificatori creaţi de programator: prim, a11, a_1, etc.

Fiecare limbaj de programare are implementat diferit conceptul de identificator al datei:

- **numărul maxim de caractere din nume** (de exemplu 31 de caractere în C++)

- **caractere acceptate în nume** (litere, cifre, caracterul linie de subliniere)
- **caracterul care poate fi folosit la începutul numelui** (majoritatea limbajelor de programare nu acceptă ca numele unei date să înceapă cu o cifră)

Observaţii

1. Limbajul C++ este case – sensitive (face diferenţa între litere mici şi litere mari).
2. Este recomandat utilizarea identificatorilor sugestivi.
3. Identificatorii limbajului C++ se scriu numai cu litere mici.

Valoarea datei reprezintă conţinutul zonei de memorie în care este stocată data.

Atributele sunt proprietăţile datelor care determină modul în care datele vor fi tratate. Cel mai important atribut al unei date este **tipul datei.**

Tipul datei defineşte apartenenţa datei la o clasă de date, căreia îi corespunde un anumit model de reprezentare internă.

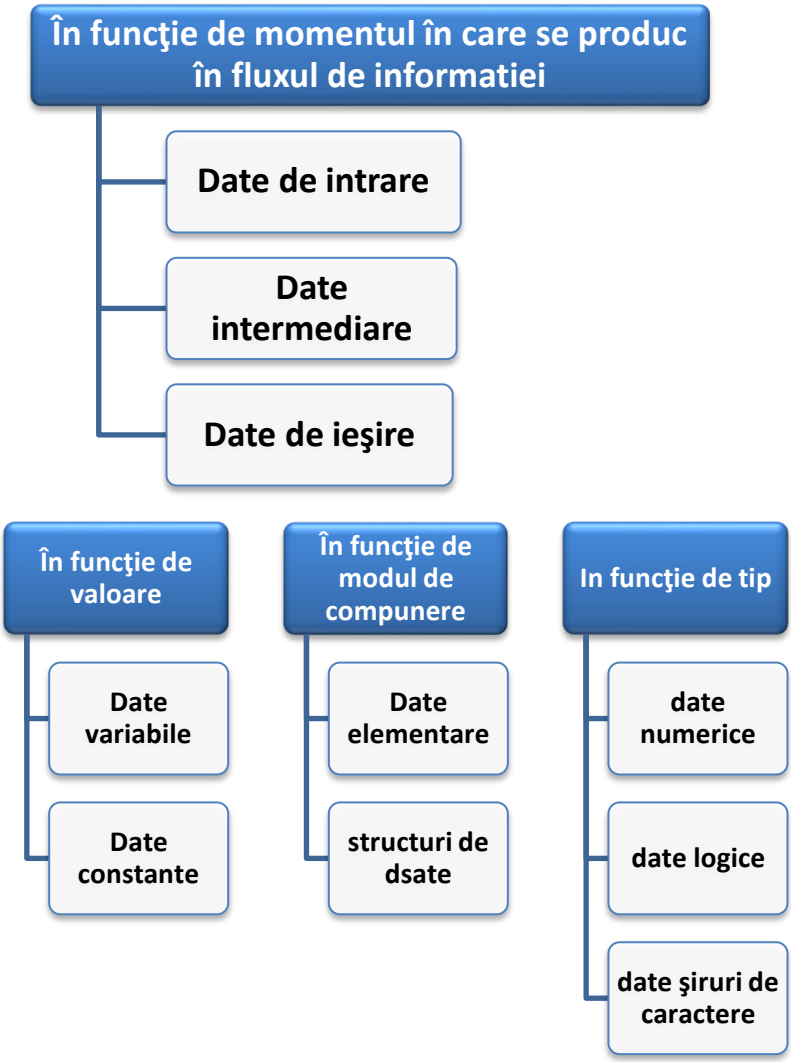
Tipul datei determină:

- **Dimensiunea zonei de memorie alocate datei** (se măsoară în octeţi sau bytes);
- **Operatorii care pot fi aplicaţi** asupra datei respective;
- **Modul în care data este reprezentată în memoria internă** (metoda de codificare a valorii date);

Reprezentarea datei în memoria calculatorului se face printr-un şir de biţi. Fiecare limbaj de programare conţine algoritmi de codificare care asigură corespondenţa dintre tipul de dată şi şirul de biţi, atât la scrierea datei cât şi la citirea lor.

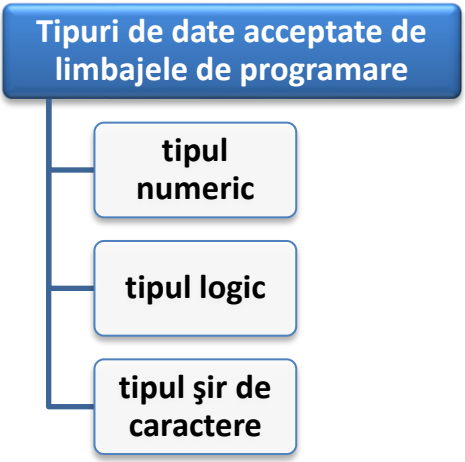
1.2. Clasificarea datelor

Datele pot fi clasificate în funcţie de mai multe criterii.



1.3. Tipul datei

Tipul datei este definit prin dubletul **(D, O)**, unde:
D – domeniul de definiţie al datei
O – mulţimea operatorilor care se pot aplica asupra datei.



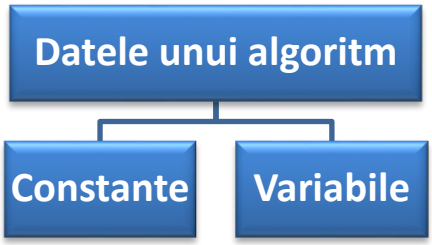
Tipul numeric a fost implementat pentru reprezentarea numerelor întregi sau reale, pozitive sau negative şi pentru a realiza majoritatea operaţiilor matematice.

Tipul logic sau **boolean** a fost implementat pentru reprezentarea datelor care nu pot lua decât două valori: adevărat (true) sau fals (false).

Tipul şir de caractere a fost implementat pentru reprezentarea unei mulţimi ordonate de caractere care este tratată ca un tot unitar.

1.4. Obiectele cu care lucrează algoritmii

Orice algoritm foloseşte două tipuri de obiecte: date şi expresii



Constante

Numim **constantă** o dată a cărei valoare ***nu se modifică*** în timpul execuţiei algoritmului.

Constantele sunt de mai multe tipuri:

1. ***constante întregi*** (exemple: 10, -32, 1332, -1567);
2. ***constante reale*** – virgula zecimală se reprezintă prin caracterul punct (exemple: 5.22, -1.33)
3. ***constante caracter*** sunt caracterele cuprinse între apostrofuri (exemple: ‘a’, ‘B’, ‘*’, ‘5’)
4. ***constante şir de caractere*** sunt şirurile de caractere cuprinse între ghilimele, care pot conţine orice caracter, inclusiv caracterul special spaţiu (exemplu: "program in C++")

Variabile

Numim **variabilă** o dată a cărei valoare ***se modifică*** în timpul execuţiei algoritmului

În limbajul C++, înainte de a utiliza o variabilă, trebuie să o declarăm. La declarare, trebuie să specificăm numele

variabilei, tipul acesteia şi, eventual, o valoare iniţială pe care dorim să o atribuim variabilei.

Observaţii

- ✓ În cadrul unui algoritm nu pot exista două variabile cu acelaşi nume.
- ✓ Tipul unei variabile stabileşte mulţimea valorilor pe care le poate primi variabila respectivă.
- ✓ La nivel de algoritm vom folosi patru tipuri:
  - ➢ Tipul întreg
  - ➢ Tipul real
  - ➢ Tipul caracter
  - ➢ Tipul şir de caractere

Expresii

În scopul efectuării calculelor, algoritmii folosesc expresii.

O **expresie** este alcătuită dintr-unul sau mai mulţi operanzi legaţi între ei prin operatori.

Operanzii pot fi constante sau variabile.

Operatorii desemnează operaţii care se execută în cadrul expresiei. Fiecare limbaj de programare are implementat propriul set de operatori.

Operanzilor dintr-o expresie li se pot aplica următorii **operatori:**

- ✓ **operatori matematici (+, -, *, mod, div)**
- ✓ **operatori relaţionali (<, >, <=, >=, ==, !=)**
- ✓ **operatori logici (negaţia, si, sau)**
- ✓ **operatorul de concatenare (+)**
- ✓ **operatorul de atribuire (←)**

În timpul execuţiei algoritmului expresiile sunt evaluate. Evaluarea unei expresii are loc astfel: se înlocuieşte variabilele cu

valorile lor şi se efectuează calculele, obţinându-se astfel valoarea expresiei.

Expresiile se împart în două categorii:

- ✓ expresii matematice
- ✓ expresii logice

Combinaţiile posibile de operatori şi operanzi

TIPUL OPERANZILOR	TIPUL OPERATORULUI	TIPUL REZULTATULUI
Numeric	Matematic	Numeric
Numeric	Relaţional	Logic
Şir de caractere	Concatenare	Şir de caractere
Şir de caractere	Relaţional	Logic
Logic	Logic	Logic

Capitolul 2

REPREZENTAREA ALGORITMILOR

2.1. Moduri de reprezentare a unui algoritm

Un **limbaj de programare** reprezintă un mijloc de comunicare între programator şi calculator. Pentru a defini un limbaj de programare, trebuie să descriem trei aspecte ale limbajului:

- **sintaxa** – reprezintă totalitatea regulilor pe bază cărora se obţin elementele limbajului
- **semantica** – defineşte semnificaţia construcţiilor sintactic corecte (a elementelor limbajului)
- **pragmatica** – defineşte modul de utilizare a elementelor limbajului

Reprezentarea algoritmilor se poate face cu ajutorul:

- ✓ schemelor logice
- ✓ limbajului pseudocod
- ✓ limbajul C++ sau alte limbaje

Pseudocodul

Ansamblul cuvintelor cheie împreună cu regulile care trebuie respectate în folosirea lor alcătuieşte sintaxa limbajului **pseudocod**.

Există o mare diversitate de limbaje pseudocod. Practic, fiecare programator îşi poate proiecta propriul pseudocod, definind cuvintele cheie ale acestuia şi impunând nişte reguli de sintaxă.

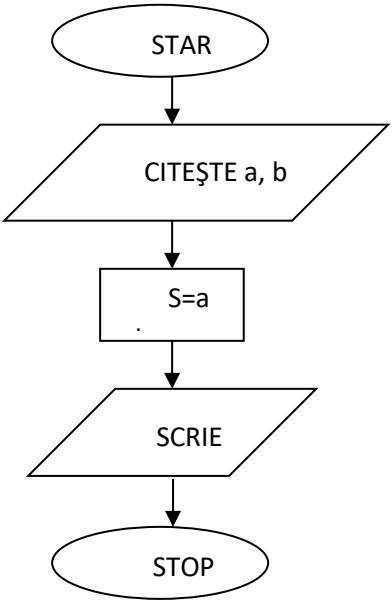
Vom folosi un limbaj pseudocod care are cuvintele cheie în română şi regulile de sintaxă din C++.

Programele C++ vor fi scrise pentru compilatorul Codeblocks.

Exemplu

Fie a şi b două numere naturale citite de la tastatură. Să se afişeze suma celor două numere.

Schema logică



Pseudocodul

```
Citeste a, b
s←a+b
scrie s
```

C++

```
#include<iostream>
using namespace std;
int main()
{
 int a, b, s;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
s=a+b;
cout<<"Suma este"<<s;
return 0;
}
```

2.2. Vocabularul limbajului C++

Vocabularul limbajului C++ este format din:

- setul de caractere
- identificatori
- separatori
- comentarii

Setul de caractere reprezintă totalitatea caracterelor cu ajutorul cărora se poate scrie un program în limbajul C++.

Setul de caractere utilizat pentru scrierea programelor C++ este setul de caractere al codului ASCII.

Setul de caractere este alcătuit din :

- ✓ litere mari şi mici ale alfabetului englez :a,…., y, A, B,…Y;
- ✓ cifrele sistemului zecimal de numeraţie : 0, 1, …, 9;
- ✓ caractere speciale: +, -, *, /, =, etc

Identificatorul este un nume asociat unei constante, variabile, funcţii, etc. Un identificator poate să conţine numai litere, cifre şi caracterul special _ şi trebuie să înceapă obligatoriu cu o literă.

Exemple:

Corect: a1, b, val_medie;

Incorect: 1b, m#t;

Cuvintele cheie (keywords) ale limbajului ***nu pot fi identificatori***. Exemple de cuvinte cheie din limbajul C++: define, else, include, case, for, char, int, return, struct, etc.

Observaţie

În limbajul C++ toate cuvintele cheie se scriu numai cu litere mici.

Separatorii pot fi după caz: blanc (caracterul „spațiu”), caracterul ; (punct şi virgulă). Au rolul de a separa unităţile sintactice.

Comentariile într-un program sunt folosite pentru ca alte persoane să înţeleagă ceea ce a dorit programatorul să transmită. Aceste

comentarii sunt ignorate de către compilator la execuţia programului.

Un **comentariu** poate fi scris în două moduri:

✓ **/* comentariu*/** - când se doreşte introducerea unui comentariu pe mai multe rânduri

✓ **// comentariu** - când se doreşte introducerea unui comentariu pe un singur rând

Exemplu

```
#include <iostream>
using namespace std;
int main()
{/*începutul programului
  programul afişează */
  cout<< "Un exemplu de program " ;
  return 0;
  //sfârşitul programului
}
```

Preprocesorul este un program lansat în execuţie automat înainte de compilare. El ***execută toate directivele*** preprocesor incluse în program, efectuând substituţii de texte.

Toate directivele preprocesor încep cu #.

Directiva **#include** este folosită pentru a include într-un program un fişier antet standard sau creat de utilizator.

Un **fişier antet** conţine declaraţiile funcţiilor, constantelor, variabilelor şi tipurilor definite într-o bibliotecă.

Fişier antet standard	**Efect**
#include<iostream>	Include un fişier antet al bibliotecii limbajului C++ cu funcţii de intrare/ieşire
#include<cmath>	Include un fişier antet al bibliotecii de funcţii matematice

Fişier antet standard	Efect
#include<cstring>	Include un fişier antet al bibliotecii de funcţii cu şiruri de caractere
#include<fstream>	Include un fişier antet al bibliotecii de funcţii cu fişiere

Structura unui program în C++

#include<iostream>	*Fişiere header (funcţiile predefinite, existente în limbaj)*
int main()	*Antetul funcţiei principale*
{	*Început programului*
int a, b, s;	*Zona declarativă:*
cout<<"a="; cin>>a; **cout<<"b="; cin>>b;** **s=a+b;** **cout<<"Suma este"<<s;** **return 0;**	*Corpul programului* *Se citesc două variabile* *Se calculează suma lor* *Se afişează suma calculată*
}	*Sfârşitul programului*

2.3. Instrucţiunile declarative

Instrucţiunile declarative sunt utilizate pentru declararea obiectelor folosite în algoritm: variabile de memorie, constante simbolice şi tipuri de date

În limbajul C++ sunt următoarele tipuri de date:

- **Tipuri standard** sau **predefinite**, adică implementate în limbaj
- **Tipuri definite** de utilizator

Tipurile standard din limbajul C++ sunt:

- ✓ Tipul întreg
- ✓ Tipul real
- ✓ Tipul caracter
- ✓ Tipul void

Tipul întreg

Numele tipului	*Dimensiunea*	*Intervalul de valori*
bool	1B	True (1), false (0)
int	4B	[-2.147.483.648, 2.147.483.647] **[-2^{31}, 2^{31}-1]**
unsigned int	4B	[0, 4.294.967.295] **[0, 2^{32}-1]**
long	4B	[-2.147.483.648, 2.147.483.647] **[-2^{31}, 2^{31}-1]**
unsigned long	4B	[0, 4.294.967.295] **[0, 2^{32}-1]**
unsigned long long	8B	[0, 10.446.744.073.709.551.615] **[0, 2^{64}-1]**

Tipul real

Numele tipului	*Dimensiunea*	*Intervalul de valori*
float	4B	[$-3{,}4 * 10^{-37}$, $3{,}4 * 10^{37}$] (cu o precizie de 6 cifre)
double	8B	[$-1{,}7 * 10^{-308}$, $1{,}7 * 10^{308}$] (cu o precizie de 10 cifre)
long double	10B	[$-3{,}4 * 10^{-4932}$, $1{,}1 * 10^{4932}$] (cu o precizie de 15 cifre)

Tipul caracter

Numele tipului	*Dimensiunea*	*Intervalul de valori*
char	1B	**[0, 255]** **[0, 2^8-1]**

Constantele de tip ***char (unsigned char)*** pot fi numere întregi din intervalul specificat sau caractere care au codul ASCII în intervalul specificat. De exemplu, caracterul ‘a’ şi constanta 97 au pentru calculator aceeaşi semnificaţie.

Cod ASCII	Caracter
048	**0**
049	1
050	2
051	3
052	4
053	5
054	6
055	7
056	8
057	9
065	**A**
066	B
067	C
068	D
069	E
070	F
071	G
072	H
073	I
074	J
075	K

Cod ASCII	Caracter
076	L
077	M
078	N
079	O
080	P
081	Q
082	R
083	S
084	T
085	U
086	V
087	W
088	X
089	Y
090	Z
097	**a**
098	b
099	c
100	d
101	e
102	f

Cod ASCII	Caracter
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z

Tipul void este un tip special, pentru care mulţimea valorilor este vidă. Acest tip se utilizează atunci când este necesar să specificăm absenţa oricărei valori (de exemplu, pentru tipul funcţiilor care nu întorc nici un rezultat)

Observaţie

În limbajul C++ **nu este implementat tipul logic**. Pentru acest tip se poate folosi orice tip numeric.
Interpretarea este următoarea:

- Unei expresii al cărei rezultat este de tip logic i se atribuie valoarea **0** pentru **False** şi **1** pentru **True**
- Într-o expresie, un operand numeric este interpretat ca un **operand logic** astfel: dacă are valoarea **0** este considerat **False** iar dacă are valoare **diferită de 0**, este considerat **True**

2.4. Variabile

Variabila este o dată care îşi modifică valoarea pe parcursul execuţiei programului

O variabilă este caracterizată de următoarele atribute:

- tip
- valoare
- adresă
- identificator

Declararea unei variabile

Sintaxa

<tip> <identificator1>;

La declarare, variabilei de memorie i se alocă o zonă de memorie cu dimensiunea corespunzătoare tipului de dată, iar tipul de dată va determina modul în care va putea fi prelucrată acea variabilă de memorie în program.

Observaţii

1. Sintaxa declarării mai multor variabile de acelaşi tip:

<tip> <identificator1>, <identificator2>;

2. O variabilă poate fi iniţializată la declararea ei sau în corpul funcţiei.

Sintaxa iniţializării unei variabile la declarare

<tip> <identificator_variabila>=<valoare>;

3. O instrucţiune declarativă poate să apără oriunde în blocul funcţiei (nu este obligatoriu să fie scrisă la începutul blocului). **Regulă**: ***declararea unei date trebuie să se facă înainte ca ea să fie folosită într-o instrucţiune.***

4. O variabilă poate fi declarată înaintea funcţiei principale **int main ()** şi se va numi **variabilă globală** sau în corpul funcţiei în zona declarativă şi se va numi **variabilă locală.**

5. Tipul unei variabile reprezintă intervalul de valori de unde variabila poate lua valori.

2.5. Constante

Constanta este o dată care nu se modifică pe parcursul unui algoritm.

Tipuri de constante:

1. **Constante întregi**
2. **Constante reale**
   - **Forma normală:** ***parte întreagă .parte zecimală***
   - **Forma exponenţială** (ştiinţifică): -1.2e-23 =$-1.2*10^{-23}$
3. **Constante caracter** sunt exprimate printr-un caracter delimitat cu apostrofuri sau prin codul ASCII al caracterului.

Orice limbaj de programare foloseşte o codificare a caracterelor cu ajutorul căreia lucrează.

Programul C++ foloseşte **codul ASCII** conform căruia fiecărui caracter îi corespunde un cod ASCII unic, un număr cuprins între 0 şi 255.

- caractere litere mari: ‚A'→65 ….'Z'→90
- caractere litere mici: ‚a'→97 ….'z'→122
- caractere cifre: ‚0'→48 ….'9'→57
- caractere speciale sunt intercalate în tot intervalul [0,255]

Observaţie

Nu confundaţi caracterul ‘9’ cu cifra 9.

4. Constante şir de caractere. O constantă de tip şir de caractere este formată dintr-o succesiune de caractere cuprinsă între ghilimele.

Exemplu: ”Informatică”

5. Constante simbolice

Declararea unei constante simbolice se poate realiza în două moduri :

I. **#define <nume_constanta> <valoare>**

- ✓ ***declararea*** se face *înaintea funcţiei int main()*

II. **const <tip> <nume_constanta>=<valoare>;**

- ✓ **declararea** se face *în zona declarativă* a funcţiei *int main()*

În varianta Borland C++ sunt câteva ***constante simbolice predefinite***:

Exemplu

```
#include <iostream>
#include<values.h>
using namespace std;
# define a INT_MAX
# define b INT_MIN
# define c LONG_MAX
#define d 15.45
int main()
{  const float e=7.15;
   cout <<a<<'\n';
   cout <<b<<'\n';
   cout<<c<<endl;
   cout<<d<<endl;
   cout<<e<<endl;
   return 0;
}
```

*Pe ecran se va afişa***:**

2147483647
-2147483648
2147483647
15.45
7.15

2.6. Declararea unui set de constante

Un grup de constante întregi din domeniul [0, 32767] poate fi definit printr-o singură instrucţiune declarativă, sub forma unui set de constante, astfel:

enum nume {lista_constante};

Setul de constante se mai numeşte şi **constantă de tip enumerare**. Lista conţine elemente de forma **nume=[valoare]** separate prin virgulă. Dacă se precizează valoarea, constanta va avea valoarea precizată. Dacă nu se precizează valoarea, constanta ca lua valoarea constantei precedente în listă, incrementată cu 1, valoare primei constante din listă este 0.

Exemple:

1. ***enum logic {Nu, Da}*** s-a definit setul de constante logic care conţine constantele Nu, care are valoare 0 şi Da care are valoarea 1.
2. ***enum set {A, B, C}*** s-a definit setul de constante set care conţine constantele A=0, B=1, C=2.
3. **enum set1 {A=2, B=5, C}** s-a definit setul de constante set1 care conţine constantele A=2, B=5, C=6
4. ***enum set2 {A=5, B, C=B, D}*** s-a definit setul de constante set care conţine constantele A=5, B=6, C=6, D=7.

2.7. Operaţiile de citire şi scriere

Introducerea datelor se poate realiza cu ajutorul tastaturii sau se pot introduce date dintr-un fişier.

Sintaxa pentru citirea *de la tastatură*:

1. Pentru citirea valorii unei singure variabile de memorie identificată prin numele ei:

cin>>nume_variabila;

Exemplu:

int a;

cin>>a;

2. Pentru citirea valorilor a n variabile de memorie identificate prin numele lor:

cin>>nume_1>>nume_2>>…>>nume_n;

Exemplu:

int a,b,c,d;

cin>>a>>b>>c>>d;

Observaţii

1. Introducerea valorii unei variabile de memorie se termină prin apăsarea tastei **ENTER**

2. Dacă de la tastatură nu se introduce o valoare ce corespunde tipurilor de variabile dintr-o listă atunci operaţia de citire se blochează, nu se mai citesc valorile nici pentru celelalte variabile din listă şi se trece la execuţia următoarei instrucţiuni din program.

Afişarea datelor se poate realiza pe ecran sau se pot afişa într-un fişier.

Sintaxa pentru scrierea pe ecran

1. Pentru afişarea valorii unei singure variabile sau constante simbolice identificate prin numele lor:

cout<<nume_variabila;

Exemplu:

int a;

cout<<a;

2. Pentru afişarea valorilor a n variabile sau constante simbolice

cout<<nume_variabila1<<…<<nume_variabila_n;

Exemplu:

int a,b,c,d;

cout<<a<<b<<c<<d;

Observaţie

Valorile precizate în listă se afişează unele după altele, fără spaţiu între ele.

Exemplu

```
int a=10, b=20;
cout<<a<<b;
```

se va afişa pe ecran: ***1020***

Pentru evidenţierea fiecărei valori se pot folosi două metode:

1. Valorile se scriu pe acelaşi rând dar separate prin spaţii:

   cout<<a<<“ ”<<b;

   se afişează pe ecran: ***10 20***

2. Valorile se scriu pe rânduri diferite, pentru separarea lor folosindu-se **endl** (care este echivalent **‘\n’**)care corespunde apăsării tastei Enter :

   cout<<a<<endl<<b;

Se afişează pe ecran ***10***
20

2.8. Expresia şi instrucţiunea expresie

Expresia este o succesiune de operatori şi operanzi legaţi între ei şi se caracterizează printr-un rezultat numit ***valoare expresiei***.

Operanzii pot fi:

- ***Nume de variabile sau de constante***
- ***Constante***
- ***Funcţii*** care furnizează un rezultat prin numele lor (ex. sqrt(x)= $\sqrt{x}$)

Operatorii sunt simboluri care determină executarea anumitor operaţii.

Operatorii pot fi:

- ***Operator unar*** – se aplică pe un singur operand
- ***Operator binar*** – se aplică pe doi operanzi
- ***Operatori ternar*** – se aplică pe trei operanzi

Tipuri de operatori în C++:

- ✓ **Operatori aritmetici** (+, -, *, /, %)
- ✓ **Operatorul de conversie explicită**
- ✓ **Operatori de incrementare şi decrementare**
- ✓ **Operatori relaţionali** (<, >, <=, >=, = =. !=)
- ✓ **Operatori logici** (&& - şi, || - sau, ! - negaţia)
- ✓ **Operatori logici pe biţi**
- ✓ **Operatorul de atribuire** =
- ✓ **Operatorul virgulă**
- ✓ **Operatorul condiţional**
- ✓ **Operatorul dimensiune**
- ✓ **Operatorul de concatenare** +

2.9. Operatori

2.9.1. Operatori aritmetici

Operatorii aritmetici implementaţi în limbajul C++ sunt:

1. **Operatori unari:**
   - ✓ operatorul minus (-)
   - ✓ operatorul plus (+)

Observaţii

Operatorii unari preced operandul. Ei se folosesc pentru a stabili semnul unui operand numeric.

Operatorii unari au prioritate mai mare decât operatorii binari.

2. **Operatori binari:**

- ✓ ***de adunare***
  - ➢ operatorul de adunare **(+)**
  - ➢ operatorul de scădere **(-)**
- ✓ ***de multiplicare***
  - ➢ operatorul de înmulţire **(*)**
  - ➢ / operatorul de împărţire **(/)**
  - ➢ peratorul modulo (**%** - restul împărţirii)

Observaţii

1. **Operatorul /** este folosit pentru ***împărţirea întreagă*** dar şi pentru ***împărţirea reală***.

Fie **c←a/b**

Dacă **a** şi **b** sunt de tip **întreg**, **c** va reprezenta câtul împărţirii lui a la b, care este tot de tip **întreg**.

Exemplu: Dacă a=9 şi b=5 atunci c=a/b=9/5=1

Dacă cel puţin unul dintre a şi b este de tip real, atunci c este de tip real şi va reprezenta împărţirea reală.

2. Operatorul **%** nu ***se*** poate ***aplica*** pe valori reale, ci doar ***pe valori întregi***.

3. Fie a şi b de tip întreg
   - ✓ **a/b** – câtul împărţirii lui a la b
   - ✓ **a%b** – restul împărţirii lui a la b
   - ✓ **a%10** – ultima cifră a numărului a
- ✓ **a/10** – elimină ultima cifră din numărul a

4. Pentru a schimba ordinea implicită de executare a operatorilor matematici se pot folosi parantezele rotunde () care au rol de separatori.

5. O expresie aritmetică poate conţine operanzi numerici de mai multe tipuri. De exemplu dacă a este de tip int şi b este de tip float atunci tipul rezultatului c se va obţine prin conversie aritmetică implicită.

Conversia implicită s-a efectuat astfel: ***tipul inferior int a fost promovat la tipul float***, iar ***rezultatul*** este de tip ***float***.

2.9.2. Operatorul de conversie explicit

Conversia unui întreg la tipul float se poate face în două moduri:

1. ***Cu ajutorul operatorului „punct”*** ce se ataşează unui **operand constantă** de ***tip întreg*** şi care capătă rolul de punct zecimal, transformând întregul în număr real.

Exemplul 1

```
int x=4, y=13;
float z;
z=(x+y)/2;
cout<<z ;
```

Pe ecran se va afisa : **8**

Exemplul 2

```
int x=4, y=13;
float z;
z=(x+y)/2. ;
cout<<z ;
```

Pe ecran se va afisa : **8.5**

Atenţie !!

Operatorul “punct” nu se poate aplica unui operand – variabilă.

2. **“Operatorul de conversie explicit” (float)** poate fi ***aplicat*** atât unui ***operand – constantă*** cât şi unui ***operand – variabilă***.

Exemplu :

```
z=((float)x+y)/2;
z=(x+(float)y)/2;
z=(float) (x+y)/2;
z=(x+y)/ (float)2;
```

2.9.3. Operatorii pentru incrementare şi decrementare

Operatorii pentru incrementare şi decrementare sunt operatori unari care se aplică pe un operand numeric.

Operatorul de incrementare ++ adună 1 la valoarea operandului.

a++ ⇔ a=a+1

Operatorul de decrementare -- scade 1 la valoarea operandului.

a-- ⇔ a=a-1

Operatori de incrementare şi decrementare pot fi:

- **Prefixaţi,** adică aplicaţi înaintea operandului **(++a sau - -a**). ***Incrementarea, respectiv decrementarea operandului se face înainte ca valoarea operandului să intre în calcul*** (înainte să fie evaluată expresia).
- **Postfixaţi,** adică aplicaţi după operandului (**a++ sau a--**). ***Incrementarea, respectiv decrementarea operandului se face după ce valoarea operandului a intrat în calcul*** (înainte să fie evaluată expresia).

Exemplu

```
#include<iostream>
using namespace std;
int main()
{int a=1,b=2;
cout<<a+++b++<<endl;          //se afiseaza 3
cout<<a<<" "<<b<<endl;        //se afiseaza 2 3
cout<<++a+(++b)<<endl;        //se afiseaza 7
cout<<a<<" "<<b<<endl;        //se afiseaza 3 4
cout<<a+b++<<endl;            //se afiseaza 7
```

```
cout<<a<<" "<<b<<endl;          //se afiseaza 3 5
cout<<a+++b--<<endl;            //se afiseaza 8
cout<<a<<" "<<b<<endl;          //se afiseaza 4 4
cout<<a+++--b--<<endl;          //se afiseaza 7
cout<<a<<" "<<b<<endl;          //se afiseaza  5 3
cout<<++a+b++<<endl;            //se afiseaza 9
cout<<a<<" "<<b<<endl;          //se afiseaza 6 4
return 0;
}
```

Operatorii de incrementare se pot aplica pe orice tip numeric.

Exemplu

```
#include <iostream>
using namespace std;
int main ()
{float a=1.5, b=-2.5;
char x='a';
cout<<a+++b++<<endl;            //se afiseaza -1
cout<<a<<" "<<b<<endl;          //se afiseaza 2.5  -1.5
cout<<a--+b--<<endl;            //se afiseaza 1
cout<<a<<" "<<b<<endl;          //se afiseaza 1.5  -2.5
cout<<++(++x)<<endl;            // afiseaza c
return 0;
}
```

Observaţii

Evitaţi astfel de situaţii.

```
int a=2
cout<<a+++a++
```

Cazul 1. Incrementarea se face după evaluarea expresiei şi rezultatul expresiei va fi 4, iar valoarea operandului a va fi 4
Cazul 2. Incrementarea se face înaintea de evaluarea expresiei şi rezultatul expresiei va fi 5, iar valoarea operandului a va fi 4.

2.9.4. Operatorii relaţionali

Toţi operatorii relaţionali sunt operatori **binari**.
Operatorii relaţionali se împart în două grupe:

- **Operatori relaţionali pentru inegalităţi (<, >, <=, >=)**
- **Operatori relaţionali pentru egalitate (= =, !=)**

Rezultatul operatorilor relaţionali sunt de tip numeric (logic)

- **0** pentru **false**
- **1** pentru **true**

Observaţii
1. Operatori relaţionali pentru inegalităţi au prioritate mai mare decât operatori relaţionali pentru egalitate.
2. Operatorii relaţionali au prioritate mai mică decât operatorii aritmetici.
Exemplu

```
#include<iostream>
using namespace std;
int main()
{int a=2, b=3;
  cout<<(a>b)<<endl;          se afiseaza 0
  cout<<(a<=b)<<endl;         se afiseaza 1
  cout<<(a= =b)<<endl;        se afiseaza 0
  cout<<(a!=b)<<endl;         se afiseaza 1
  return 0;
```

}

Observaţie

Operatorul de ieşire << are prioritate mai mică decât operatorii unari şi matematici, dar mai mare decât operatorii relaţionali.

Deoarece cout<< are prioritate şi evaluarea se face de la stânga la dreapta, pentru ca întreaga expresie să fie evaluată corect, scrieţi expresia logică între paranteze rotunde.

2.9.5. Operatorii logici

Există trei operatori logici:

- **Operatorul unar**: **!** operatorul pentru negaţia logică
- **Operatori binari**:
  - ✓ **&&** operatorul ***ŞI logic***
  - ✓ || operatorul ***SAU logic***

Se aplică pe orice variabilă sau constantă de tip numeric (0 pentru fals şi orice valoare diferită de 0 pentru adevărat) şi produc un rezultat numeric: 0 pentru false şi 1 pentru true

Precedenţa operatorilor logici este:

- Operatorul unar are prioritate mai mare decât operatorii binari.
- Operatorul binar && are prioritate mai mare decât operatorul ||.

Operatorii logici au prioritate mai mică decât operatorii relaţionali.

Exemplu:

```
#include <iostream>
using namespace std;
int main()
{ float a=2.5;
    int b=0,c=2,d=3;
    cout<<(a>0)<<endl;                    se afișează 1
    cout<<(!a)<<endl;                     se afișează 0
```

```
cout<<(c&&d)<<endl;                    se afișează 1
cout<<(b&&c)<<endl;                    se afișează 0
cout<<(b||c)<<endl;                    se afișează 1
cout<<(a&&d)<<endl;                    se afișează 1
cout<<(b&&a)<<endl;                    se afișează 0
cout<<(b||a)<<endl;                    se afișează 1
return 0;
}
```

a	b	!a	a&&b	a||b
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

Observaţii

1. În cazul operatorilor logici binari al doilea operand nu mai este evaluat, dacă prin evaluarea primului operand se poate decide valoarea rezultatului:
2. **Operatorul &&.** Daca primul operand are valoarea 0, rezultatul este 0 oricare ar fi valoarea celui de al doilea operand, iar dacă primul are valoarea 1, rezultatul depinde de al doilea operand.
3. **Operatorul ||.** Daca primul operand are valoarea 1, rezultatul este 1 oricare ar fi valoarea celui de al doilea operand, iar dacă primul are valoarea 0, rezultatul depinde de al doilea operand.

2.9.6. Operatorii de atribuire

Operatorul de atribuire = este un operator binar.

Există trei moduri de a folosi acest operator:

- Atribuire simplă
- Atribuire multiplă
- Atribuirea cu operator

Atribuirea simplă

Sintaxa

nume_variabilă=expresie

Exemplu: a=5

Observaţii

1. Tipul expresiei a=b este dat de tipul operandului a.
2. Dacă unei variabile de tip int **i** se atribuie valoarea unui variabilei reale, valoarea acesteia este trunchiată.

Exemplu:

```
#include<iostream>
using namespace std;
int main()
{ int a;
  float b=1.8, c=-2.7;
  a=b;
  cout<<a<<endl;       // se afişează 1
  a=c;
  cout<<a<<endl;       // se afişează -2
  return 0;
}
```

Atribuirea multiplă

În C++ putem atribui aceeaşi valoare mai multor variabile:

<nume_var_1>=<nume_var_2>=…..=<valoare>

Într-o astfel de expresie, atribuirea se efectuează de la dreapta la stânga.

Exemplu:

```
int x, z, y;
x=y=z=2;
```

Compilatorul va atribui valoarea variabilelor astfel:

```
z=2;
y=z;
x=y;
```

Observaţie

O iniţializare prin atribuire multiplă nu se poate.
Nu este corect int x=z=z=2;
Corect este int x=2, y=2, z=2;

Atribuirea cu operatori: „+=”, „-=”, „*=”, „/=”

<var>+=<expr> este echivalent cu **<var>=<var>+<expr>**
<var>-=<expr> este echivalent cu **<var>=<var> - <expr>**
<var>*=<expr> este echivalent cu **<var>=<var>*<expr>**
<var>/=<expr> este echivalent cu **<var>=<var>/<expr>**

Exemplu:

```
#include<iostream>
using namespace std;
int main()
{int a=10, b=20, c=30;
a+=b+c;
cout<<a;        // se afişează 60
return 0;
```

}

Observaţie
Se pot folosi atribuiri multiple cu operatori.

Exemplu:
```
#include<iostream>
using namespace std;
int main()
{int a=10, b=20, c=10;
a+=b+=c;                    //b=b+c   a=a+b
cout<<a<<" "<<b;            // se afişează  40  30
return 0;
    }
```

2.9.7. Operatorul condiţional

Operatorul **?:** este singurul **operator ternar** (compus din trei elemente):

expresie_1 ? expresie_2 : expresie_3

Executarea acestui operator se face astfel:
Se evaluează **expresia_1;**

✓ Dacă valoarea obţinută este diferită de 1 atunci se evaluează **expresia_2**, iar **expresia_3** este ignorată. Valoarea furnizată de operator va fi valoarea **expresiei_2.**

✓ Dacă valoarea obţinută este 0 atunci se evaluează **expresia_3**, iar **expresia_2** este ignorată. Valoarea furnizată de operator va fi valoarea **expresiei_3.**

Exemplu
1. Afişarea modulului unui număr citit de la tastatură.

```
#include<iostream>
using namespace std;
int main ()
{int x;
cout<<”x=” ;
cin>>x ;
cout<< ”modulul =” << (x>=0 ? x : -x) ;
return 0;
}
```

Observaţie

Tipul rezultatului va fi determinat prin conversie implicită. De exemplu, dacă tipul **expresiei_3** este superior tipului **expresiei_2**, tipul rezultatului este dat de **expresie_3**, chiar dacă se evaluează **expresie_2.**

Exemplu

```
#include<iostream>
using namespace std;
int main ()
{int a=3;
float b=2 ;
cout<<(a>0 ? a : b)/2<<endl;  // se va afişa 1.5
return 0;
}
```

Operatorul condiţional poate fi folosit în locul unei **structuri alternative simplă.**

Exemple

1. Se citeşte un număr de la tastatură un număr. Să se afişeze dacă este par sau impar.

```
#include<iostream>
using namespace std;
int main()
{int a;
cout<<”a=”; cin>>a;
cout<<(a%2==0 ? ”numar par” : ”numar impar” ) ;
return 0;
}
```

2. Se citeşte un număr de la tastatură un număr. Să se afişeze dacă este număr întreg sau nu.

```
#include<iostream>
using namespace std;
int main()
{float a;
cout<<”a=”; cin>>a;
cout<<(int(a)==a ? ”numar intreg” : ”numar real” ) ;
return 0;
}
```

3. Se citesc trei numere intregi de la tastatură. Să se afişeze maximul dintre numere.

```
#include<iostream>
using namespace std;
int main()
{int a,b,c,max;
cout<<”a=”; cin>>a;
cout<<”b=”; cin>>b;
```

```
cout<<"c="; cin>>c;
max=a ;
max=max<b ?b :max ;
max=max<c ?c :max
cout<<max;
return 0;
}
```

4. Se citeşte un număr de la tastatură un număr. Să se afişeze dacă este pătrat perfect sau nu.

```
#include<iostream>
using namespace std;
int main()
{int a;
cout<<"a="; cin>>a;
cout<<(sqrt(a)*sqrt(a)==int(a)?"numar patrat perfect"
: "numar nu este patrat perfect" ) ;
return 0;
}
```

5. Să se scrie un programcare transformă litera mare în literă mică şi invers.

```
#include<iostream>
using namespace std;
   int main()
   {char a;
    cout<<"a="; cin>>a;
cout<<(a>='a'&&a<='z'  ?  (char)(a-32):(a>='A'&&a<='Z')    ?
(char)(a+32) : a);
   cout<<a;
```

return 0;
}

2.9.8. Operatorul virgulă

Operatorul virgulă poate fi folosit pentru construirea unei expresii compuse din mai multe expresii:

expresie_1, expresie_2, expresie_3, …, expresie_n

Executarea acestui operator se face astfel:

se evaluează **expresia_1**
se evaluează **expresia_2**
….
se evaluează **expresia_n**

Observaţii

- Rezultatul şi tipul expresiei este dat de valoarea ultimei expresii.
- Evaluarea expresiilor se face de la stânga la dreapta.
- Este operatorul cu cea mai mică prioritate.
- Folosirea operatorului virgulă este utilă acolo unde sintaxa nu permite decât o singură expresie şi trebuie evaluate mai multe expresii. Expresiile vor fi legate prin operatorul virgulă, obţinându-se o singură expresie.

Exemplu

```
#include<iostream>
int main()
{int a=2,b=3;
float c;
c=a=++b, a++, --b;
cout<<a<<" "<<b<<" "<<c; se afişează   5        3        4
return 0;
}
```

Nu confundaţi caracterul virgulă folosit ca separator în zona declarativă cu operatorul virgulă.

2.9.9. Operatorul dimensiune

Operatorul dimensiune **sizeof** este un **operator unar**. Operandul poate fi o expresie sau un tip de dată:

sizeof (expresie)
sizeof (tip_data)

Rezultatul acestui operator este **numărul de octeţi** utilizaţi pentru a memora valoarea expresiei sau tipul de dată.

Aplicarea acestui operator pe o expresie nu are ca efect şi evaluarea expresiei.

Exemplu:

```
#include<iostream>
using namespace std;
int main()
{int a=2,b=3;
float c;
cout<<sizeof(a)<<" "<<sizeof(b)<<" "<<sizeof(c);
return 0;
    }
```

Se va afişa pe ecran: 2 2 4

2.9.10. Prioritatea operatorilor în C++

Nr. crt.	Categorie	Operatori	Semnificaţie
1	Prioritate maximă	**( )**	Apel de funcţie (exemplu: sqrt(x))
2	Operatori unari	+ -	plus minus
3		++ -- **!** **sizeof** **(tip)**	Incrementare Decrementare Negare logică Dimensiune (în octeţi) Conversie explicită de tip
4	Operatori aritmetici de multiplicare	*** / **%**	Înmulţirea împărţirea restul
	a%b = restul împărţirii lui a la b (a, b întregi) **a/b = câtul împărţirii lui a la b (a, b întregi)** **a%10 = ultima cifră a lui a (a întreg)** **a/10 = numărul a fără ultima cifră (a întreg)**		
5	Operatori aritmetici adiţionali	+ -	Adunare Scădere
6	Operatorii relaţionali pentru inegalităţi	< <= > >=	Mai mic Mai mic sau egal Mai mare Mai mare sau egal

Nr. crt.	Categorie	Operatori	Semnificaţie
7	Operatori relaţionali pentru egalităţi	== !=	Egal Diferit
8	Operatori logici	&&	**Si logic** (adevărat (1) dacă ambii operanzi sunt adevăraţi)
9	Operatori logici	||	**Sau logic** (adevărat (1) dacă cel puţin un operand este adevărat)
10	Operatorul condiţional	? :	**condiţie ? expresie 1 : expresie**
11	Operatori de atribuire	= += -= *= /=	Atribuire simplă Atribuire cu operatori matematici
	var += valoare ⇔ var =var+valoare **var -= valoare ⇔ var =var-valoare** **var *= valoare ⇔ var =var*valoare** **var /= valoare ⇔ var =var/valoare**		
12	Operatorul virgulă	,	Evaluare multiplă

Capitolul 3

PRINCIPIILE PROGRAMĂRII STRUCTURATE

Prin structură înţelegem o formă de îmbinare a operaţiilor cu care lucrează un algoritm.

Pentru realizarea unui algoritm sunt suficiente trei tipuri de structuri:

- Structură liniară
- Structură alternativă (de selecţie)
- Structură repetitivă

3.1. Structura liniară

Orice operaţie (citire/ scriere, atribuire, decizională în ansamblul ei) constituie o **structură liniară**.

Exemplu

Să se determine ultimele două cifre ale produsului a*b, pentru a, b numere întregi ce sunt citite de la tastatură.

Pseudocod	**C++**

citeşte a, b;
p←a*b;
u←p%100;
scrie u;
	#include<iostream>
	using namespace std;
	int main ()
	{int a,b,p,u;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	p=a*b;
	u=p%100;
	cout<<"Ultimile 2 cifre sunt "<<u;
	return 0;}

3.2. Structura alternativă (de selecţie)

Structura alternativă are două forme:

- Structura alternativă simplă
- Structura alternativă multiplă

3.2.1. Structura alternativă simplă

Sintaxa

Pseudocod	C++
dacă (<condiţie>) **atunci** secventa1; **altfel** secventa2;	**if** (<conditie>) secventa1; **else** secventa2;

Observaţia 1

Condiţia din structura alternativă trebuie întotdeauna trecută între paranteze.

Mecanismul de executare a structurii alternative simple

Se testează condiţia

- Dacă condiţia este adevărată se execută ***secvenţa1*** de instrucţiuni;
- Dacă condiţia este falsă se executa ***secvenţa2*** de instrucţiuni

Forma simplificată a structurii alternative simple

Pseudocod	C++
dacă (<condiţie>) **atunci** secventa1;	**if** (<conditie>) secventa1;

Exemplu 1

Se citesc două numere întregi de la tastatură. Să se afişeze pe ecran cel mai mare număr.

Pseudocod	C++

citeste a, b;
dacă (a>b) **atunci**
max←a;
altfel
max←b;
scrie max;
	#include<iostream>
	using namespace std;
	int main ()
	{int a,b,max;
	cout<<"a=";
	cin>>a;
	cout<<"b=";
	cin>>b
	if (a>b)
	max=a;
	else
	max=b;
	cout<<"maximul este
	"<<max ;
	return 0;
	}

Observaţia 2

Dacă condiţia ataşată instrucţiunii alternative este formata din mai multe condiţii atunci şi aceste sunt trecute între paranteze.

Observaţia 3

Dacă secvenţa de instrucţiuni ***secventa1*** sau ***secventa2*** conţine mai mult de o instrucțiune aceasta trebuie cuprinsă între acolade.

Exemplu 2

Se citesc de la tastatură latura unui triunghi b şi înălţimea corespunzătoare laturii h. Să se afişeze pe ecran aria triunghiului.

Pseudocod	C++

citeşte b,h;
dacă ((b>0)&&(h>0)) **atunci**
A←(b*h)/2;
scrie A
altfel
scrie "date incorecte"
	#include<iostream>
	using namespace std;
	int main ()
	{**float** b,h,A;
	cout<<"b=";
	cin>>b;
	cout<<"h=";
	cin>>h;
	if ((b>0)&&(h>0))
	{
	A=(b*h)/2;
	cout<<"Aria este"<<A;
	}
	else
	cout<<"date incorecte";
	return 0;
	}

Exemplul 3

Să se calculeze valoarea funcţiei f(x) pentru o valoare x introdusă de la tastatură.

f:R→R $$f(x)=\begin{cases} x^2+2 \text{ pentru } x \le -5 \\ x-4 \text{ pentru } x > -5 \end{cases}$$

Pseudocod	C++

citeşte x;
dacă (x<=-5) **atunci**
f←x*x+2;
else
f←x-4;
scrie f;
	#include<iostream>
	using namespace std;
	int main ()
	{float x, f;
	cout<<"x=";
	cin>>x;
	if (x<=-5)
	f=x*x+2;
	else
	f=x-4;
	cout<<"valoarea funcție este"<<f;
	return 0;
	}

Observaţia 4

Într-o instrucţiune alternativă **if - else**, în corpul oricărei ramuri poate fi la rândul său introdusă o altă instrucţiune alternativă **if – else.** Structura se va numi ***structură alternativă imbricată***.

Exemplu 4

Să se afişeze pe ecran valoarea funcţiei f(x) pentru o valoare x introdusă de la tastatură.

$$f:R\rightarrow R \qquad f(x)=\begin{cases} x^2+1 \text{ pentru } x \le -3 \\ x-2 \text{ pentru } x \in (-3,3) \\ x^2-4x+5 \text{ pentru } x \ge 3 \end{cases}$$

Pseudocod	C++

citeşte x;
dacă (x<=-3) **atunci**
f←x*x+1;
atunci
dacă (x<3) **atunci**
f←x-2;
atunci
f←x*x-4*x+5;
scrie f;
	#include<iostream>
	using namespace std;
	int main ()
	{float x, f;
	cout<<"x=";
	cin>>x;
	if (x<=-3) f=x*x+1;
	else
	if (x<3) f=x-2;
	else
	f=x*x-4*x+5;
	cout<<"valoarea functiei este "<<f;
	return 0;
	}

3.2.2. Structura alternativă multiplă

Pseudocod	C++

alege (<selector>)
început
cazul<v1>: <secventa_1>;
iesire;
cazul<v2>: <secventa_2>;
iesire;
……
cazul<vn>:<secventa_n>;
iesire;
altfel:
<secventa_0>;
sfârşit
	switch(<expresie>)
	{
	case <exp_1>:<secventa_1>;
	break;
	case <exp_2>:<secventa_2>;
	break;
	……
	case <exp_n>:<secventa_n>;
	break;
	default:
	<secventa_0>;
	}

Observaţie

Expresia ***expresie*** din cadrul structurii alternative multiple trebuie să furnizeze un rezultat numeric ***întreg***, iar expresiile care se evaluează pentru fiecare caz ***(exp_1, exp_2, …)*** trebuie să fie ***constante întregi*** sau ***expresii constante cu valoare întreagă***.

Principiul de funcţionare

Se evaluează expresia ***expresie***.

- Dacă ***rezultatul obţinut*** are valoarea ***exp_1 se execută secvenţa_1*** după care se trece la execuţia instrucţiunii care urmează instrucţiunii switch, astfel se trece la pasul următor
- Dacă ***rezultatul obţinut*** are valoarea ***exp_2 se execută secvenţa_2*** după care se trece la execuţia instrucţiunii care urmează instrucţiunii switch, astfel se trece la pasul următor.

……

- Dacă ***rezultatul obţinut*** are valoarea ***exp_n se execută secvenţa_n*** după care se trece la execuţia instrucţiunii care urmează instrucţiunii switch, ***astfel se execută instrucţiunea care urmează după eticheta default***, dacă aceasta există, după care se trece la execuţia instrucţiunii care urmează instrucţiunii switch.

Observaţii

1. Toate expresiile corespunzătoare cazurilor ***exp_i*** trebuie să fie ***diferite între ele.***
2. ***Eticheta default este opţională***. Instrucţiunea ataşată acestei etichete se execută numai dacă nu a fost îndeplinit nici un caz anterior.
3. Instrucţiunea break este obligatorie deoarece întrerupe execuţia switch…case şi trece la executarea următoarei instrucţiuni din

program. Dacă nu se scrie această instrucţiune se continuă execuţia instrucţiunilor până la întâlnirea separatorului }.

Exemplu:

1. Să se alcătuiască un program care citeşte două numere a şi b, apoi afişează media aritmetică, suma pătratelor sau suma cuburilor celor două numere în funcţie de dorinţa utilizatorului.

Pseudocod

```
citeste a,b;
citeşte x ;
alege (x)
  inceput
      cazul 1: scrie „Media aritmetica”; scrie (a+b)/2.; break;
      cazul 2: scrie suma pătratelor; scrie a*a+b*b; break;
       cazul 3: scrie suma cuburilor; scrie a*a*a+b*b*b; break;
   altfel:
            scrie “eroare”;
 sfarsit
```

C++

```
#include<iostream>
using namespace std;
int main()
{int a, b, x;
cout<<"a=";  cin>>a;
cout<<"b=";  cin>>b;
cout<<"x=";  cin>>x;
switch (x)
{case 1:cout<<"Media aritmetica "<<(a+b)/2.; break;
case 2: cout<<"Suma patratelor "<<a*a+b*b; break;
case 3:cout<<"Suma cuburilor" <<a*a*a+b*b*b; break;
default:
    cout<<"eroare";}
return 0;}
```

2. Să se alcătuiască un program care citeşte două numere a şi b, apoi afişează suma, diferenţa, înmulţirea şi împărţirea acestora în funcţie de dorinţa utilizatorului.

Pseudocod

```
citeste a,b;
citeşte x ;
alege (x)
  inceput
     cazul 1: scrie „Suma  ”; scrie (a+b); break;
     cazul 2: scrie „Diferenţa  ”; scrie a-b; break;
     cazul 3: scrie „Înmulțirea  ”; scrie a*b; break;
     cazul 4: scrie „Împărţirea   ”; scrie (float)a/b; break;
   altfel:
          scrie “eroare”;
 sfarsit
```

C++

```
#include<iostream>
using namespace std;
int main()
{int a, b, x;
cout<<"a=";  cin>>a;
cout<<"b=";  cin>>b;
cout<<"x=";  cin>>x;
switch (x)
{case 1:cout<<"Suma   "<<  a+b; break;
case 2: cout<<"Diferenţa  "<<a-b; break;
case 3:cout<<"Înmulțirea " <<a*b; break;
case 4:cout<<"Împărţirea  " <<(float)a/b; break;
default:
   cout<<"eroare";
}
return 0;}
```

3.3. Structuri repetitive

Structurile repetitive sunt clasificate astfel:

1. Structuri repetitive cu număr necunoscut de paşi
    a. Structura repetitivă cu test iniţial
    b. Structura repetitivă cu test final
2. Structura repetitivă cu un număr cunoscut de paşi

3.3.1. Instrucţiunea repetitivă cu test iniţial

Sintaxa

Pseudocod	C++
cât timp (<*condiţie*>) execută <secventa>	**while (<*condiţie*>)** <secventa>

Principiul de funcţionare

Atâta timp cât este îndeplinită condiţia se execută secvenţa.
Se ***evaluează*** **condiţia logică**

- dacă condiţia este adevărată atunci se execută secvenţa, apoi se revine la pasul 1
- în caz contrar se trece la prima instrucţiune după ciclu

Observaţii

1. Condiţia logică va fi scrisă întotdeauna între paranteze rotunde
2. În cazul în care secvenţa conţine cel puţin două instrucţiuni, acestea vor fi cuprinse între acolade.
3. Structura repetitivă cu test iniţial este cunoscută şi sub numele structură repetitivă condiţionată anterior.

Exemple:

1. Se dă un şir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din şir). Să se afişeze numerele citite pe ecran cu spaţiu între ele.

Pseudocod	**C++**

citeste x;
cât timp (x!=0) execută
scrie x;
citeste x;
	#include<iostream>
	using namespace std;
	int main()
	{int x;
	cout<<"x=", cin>>x;
	while (x!=0)
	{cout<<x<<" ";
	cout<<"x="; cin>>x;}
	return 0;
	}

2. Se citeşte un număr natural de la tastatură. Să se afişeze pe ecran suma cifrelor numărului dat.

Pseudocod	**C++**

citeste x;
s←0;
cât timp (x!=0) execută
s ←s+x%10;
x←x/10;
scrie s
	#include<iostream >
	using namespace std;
	int main()
	{int x,s=0;
	cout<<"x=", cin>>x;
	while (x!=0)
	{s=s+x%10;
	s=s/10;
	}
	cout<<"Suma cifrelor este "<<s;
	return 0;
	}

3.3.2. Instrucţiunea repetitivă cu test final

Sintaxa

Pseudocod	C++
execută secvenţă **cât timp (condiţie)**	**do** secvenţă **while (condiţie);**
repetă secvenţă **până când (condiţie)**	**do** secvenţă **while (!condiţie);**

Principul de funcţionare

1. Se execută secvenţa
2. Se evaluează condiţia.

➢ Dacă condiţia este adevărată se revine la pasul 1

➢ Altfel se trece la următoarea instrucţiune care urmează după instrucţiunea do …while

Observaţii

1. Condiţia logică va fi scrisă întotdeauna între paranteze rotunde
2. După condiția logică se adaugă punct şi virgulă.
3. În cazul în care secvenţa conţine cel puţin două instrucţiuni, acestea vor fi cuprinse între acolade.
4. Structura repetitivă cu test final este cunoscută şi sub numele structură repetitivă condiţionată posterior.

Exemplu

1. Se dă un şir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din şir). Să se afişeze numerele citite pe ecran cu spaţiu între ele.

Pseudocod	**C++**

execută
  citeste x;
  scrie x;
cât timp (x!=0)
	#include<iostream>
	using namespace std;
	int main()
	{int x;
	do
	{
	cout<<"x="; cin>>x;
	cout<<x<<" ";
	}
	while (x!=0);
	return 0;
	}

2. Se citeşte un număr natural de la tastatură. Să se afişeze pe ecran suma cifrelor numărului dat.

Pseudocod	**C++**

citeste x;
s←0;
execută
  s ←s+x%10;
  x←x/10;
cât timp (x!=0)
scrie s
	#include<iostream>
	using namespace std;
	int main()
	{int x,s=0;
	cout<<"x=", cin>>x;
	do
	{s=s+x%10;
	s=s/10; }
	while (x!=0);
	cout<<"Suma cifrelor este "<<s;
	return 0;
	}

3.3.3. Instrucţiunea repetitivă cu un număr cunoscut de paşi

Sintaxa

Pseudocod
pentru (<contor> ← <v1>, <v2>, …, <vn>) **execută** secvenţă

Principiul de funcţionare

Contorul ciclului <contor> parcurge pe rând valorile <v1>, <v2>, …, <vn> şi pentru fiecare dintre aceste valori se execută secvenţa de instrucţiuni secvenţa ce reprezintă corpul ciclului.

Sintaxa

C++
for (<expresie1> ; <expresie2> ; <expresie3>) secvența

unde:

- ***<expresie1>*** este expresia de iniţializare a ciclului adică valoarea cu care va intra în ciclu şi va executa primul pas
- ***<expresie2>*** reprezintă condiţia de continuare a ciclului. Această expresie ne asigură că ciclul este unul finit, acesta se va încheia în momentul în care condiţia dată nu mai este adevărată;
- ***<expresie3>*** asigură trecerea la pasul următor al ciclului

Observaţii

1. În cazul în care corpul ciclului are cel puţin două instrucţiuni, acesta trebuie cuprins între acolade
2. În cele mai multe situaţii structura repetitivă cu număr cunoscut de paşi o regăsim astfel:

Pseudocod
pentru i ← 1,n,+1 **execută** secvenţă
C++
for (i=1;i<=n;i++) **s**ecvenţă

Exemple

1. Se citeşte de la tastatură un număr natural n şi apoi un şir de n numere întregi.. Afişaţi pe ecran suma celor n numere întregi.

Pseudocod	**C++**

citeste n;
s←0;
pentru i ←1,n execută
citeste x;
s←s+x;
scrie s;
	#include<iostream>
	using namespace std;
	int main()
	{int n, x, s=0,i;
	cout<<"n="; cin>>n;
	for (i=1; i<=n; i++)
	{
	cout<<"x=";cin>>x;
	s=s+x;
	}
	cout<<"suma numerelor este "<<s;
	return 0;
	}

2. Scrieţi un algoritm care pentru o valoare n citită de la tastatură, calculează valoarea expresiei:

S=1*3+2*5+3*7+…+n*(2n+1)

Pseudocod	C++

citeste n;
s←0;
pentru i ←1,n execută
s←s+i*(2*i+1);
scrie s;
	#include<iostream>
	using namespace std;
	int main()
	{int n,i,s=0;
	cout<<"n="; cin>>n;
	for (i=1; i<=n; i++)
	s=s+i*(2*i+1);
	cout<<"suma este "<<s;
	return 0;
	}

3. Să se afişeze primii n termeni ai şirului lui Fibonacci. Şirul are primii doi termeni egali cu 1 şi fiecare din următorii termeni este egal cu suma dintre termenul precedent şi termenul anteprecedent.

Pseudocod	C++

citeste n;
prec ←1;
anteprec ←1;
scrie prec;
scrie anteprec;
pentru i ←3,n execută
p←prec+anteprec;
scrie p;
anteprec←prec;
prec←p
	#include<iostream>
	using namespace std;
	int main ()
	{int p,prec, anteprec, i,n;
	cout<<"n="; cin>>n;
	prec=1;
	anteprec=1;
	cout<<prec<<" ";
	cout<<anteprec<<" ";
	for (i=3; i<=n; i++)
	{p=prec+anteprec;
	cout<<p<<" ";
	anteprec=prec;
	prec=p;}
	return 0;}

3.4. Transformările structurilor repetitive

Transformarea unei structuri repetitive cu **test iniţial** într-o structură repetitivă cu **test final**

Structura repetitivă cu **test iniţial**	⇔	Structura repetitivă cu **test final**
Varianta 1		
cât timp (***condiţie***) **execută** secvenţă	⇔	**daca (condiţie) atunci** **execută** secvența **cât timp** (***condiție***)
Varianta 2		
cât timp (***condiţie***) **execută** secvenţă	⇔	**dacă (*condiție*) atunci** **repeta** secvența **până când** (***! condiție***)

Transformarea unei structuri repetitive cu **test final** într-o structură repetitivă cu **test iniţial**

Structura repetitivă cu **test final**	⇔	Structura repetitivă cu **test iniţial**
Varianta 1		
repetă secvență **până când** (***condiție***)	⇔	secvență **cât timp** (!condiţie) **execută** secvenţă
Varianta 2		
execută secvență **cât timp** (***condiție***)	⇔	secvenţă **cât timp** (***condiţie***) **execută** secvenţă

Transformarea unei structuri repetitive cu **test iniţial** într-o structura repetitivă cu **un număr cunoscut de paşi**

Structura repetitivă cu **test iniţial**	⇔	Structura repetitivă cu **număr cunoscut de paşi**
cât timp (***a<=b***) **execută** secvenţă	⇔	**pentru i←a,b,+1 execută** secvenţă

Transformarea unei structuri repetitive cu **un număr cunoscut de paşi** într-o structura repetitivă cu **test iniţial**

Structura repetitivă cu **număr cunoscut de paşi**	⇔	Structura repetitivă cu **test iniţial**
pentru i←a,b,+1 execută secvenţă	⇔	**i←a** **cât timp** (***i<=b***) **execută** secvenţă ***i←i+1***

Transformarea unei structuri repetitive cu **test final** într-o structura repetitivă cu **un număr cunoscut de paşi**

Structura repetitivă cu **test final**	⇔	Structura repetitivă cu **număr cunoscut de paşi**
repetă secvenţă **pana când** (! a<=b)	⇔	**pentru i←a,b,+1 execută** secvenţă

Transformarea unei structuri repetitive cu **un număr cunoscut de paşi** într-o structura repetitivă cu **test final**

Structura repetitivă cu **număr cunoscut de paşi**	⇔	Structura repetitivă cu **test final**
pentru i←a,b,+1 execută secvenţă	⇔	**i←a** **dacă (i<=b) atunci** **execută** secvenţă ***i←i+1*** **cât timp** (i<=b)

Capitolul 4

ALGORITMI ELEMENTARI

Algoritmi elementari folosiţi în rezolvarea problemelor sunt:

- Algoritmi pentru interschimbarea a valorilor a două numere
- Algoritmi pentru determinarea valorii minime (maxime)
- Algoritmi pentru prelucrarea cifrelor unui număr
- algoritmi pentru determinarea cmmdc dintre două numere
- Algoritmi pentru testarea numerelor prime
- Algoritmi pentru prelucrarea divizorilor unui număr

4.1. Algoritmi pentru interschimbarea a valorilor a două numere

Metoda 1

Interschimbarea valorilor a două variabile de memorie a şi b nu se face prin simpla atribuire a noilor valori, deoarece secvența de atribuiri a←b şi b←a duce la pierderea valorii lui a.
Interschimbarea valorilor a două variabile a şi b se poate face folosind o variabilă intermediară aux.

Paşii algoritmului sunt:

1. Se atribuie valoarea primei variabile **a** în variabila aux;
2. Se atribuie primei variabile **a** valoarea celei de a doua variabile **b**;
3. Se atribuie celei de a doua variabile **b** valoarea care a fost salvată în variabilă **aux**.

Pseudocod	C++

real a,b,aux;
citeşte a, b;
aux←a;
a←b;
b←aux;
scrie a, b;
	#include<iostream>
	using namespace std;
	int main ()
	{float a, b,aux;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	aux=a;
	a=b;
	b=aux;
	cout<<a<<" "<<b;
	return 0;
	}

Metoda 2

Interschimbarea valorilor a două variabile se poate face şi fără a folosi o variabilă intermediară. Interschimbarea se poate realiza folosind următoarele identităţi matematice:

a=(a-b)+b şi b=((a-b)+b)-(a-b)

Pseudocod	C++

real a,b,aux;
citeşte a, b;
a←a-b;
b←a+b;
a←b-a;
scrie a, b;
	#include<iostream>
	using namespace std;
	int main ()
	{int a, b, aux;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	a=a-b;
	b=a+b;
	a=b-a;
	cout<<a<<" "<<b;
	return 0;
	}

Exemple

1. Se citeşte un număr natural ***n*** format din exact două cifre. Să se afişeze numărul minim care se poate forma din cifrele sale.

Pseudocod	**C++**

întreg n,a,b,aux;
citeşte n;
a←n%10;
b←n/10;
dacă (b>a)
  aux=a;
  a=b;
  b=aux;
n←b*10+a;
scrie n;
	#include<iostream>
	using namespace std;
	int main ()
	{int n,a, b,aux;
	  cout<<"n="; cin>>n;
	  a=n%10;
	  b=n/10;
	  if (b>a)
	  {aux=b;
	  b=a;
	  a=aux;}
	  n=b*10+a;
	  cout<<n;
	return 0;
	}

2. Se citeşte un număr natural format din exact trei cifre. Să se afişeze numărul maxim care se poate forma din cifrele sale.

Pseudocod	**C++**

întreg n,a,b,c,aux;
citeşte n;
a←n/100;
b←n/10%10;
c←n%10;
scrie a, b, c;
dacă (a<b) atunci
  aux←a;
  a←b;
  b←aux;
dacă (b<c) atunci
	#include<iostream>
	using namespace std;
	int main ()
	{int n,a, b,c,aux;
	cout<<"n="; cin>>n;
	a=n/100;
	b=n/10%10;
	c=n%10;
	cout<<a<<" "<<b<<"
	"<<c<<endl;
	  if (a<b)

Pseudocod	C++

aux←b;
b←c;
c←aux;
dacă (a<b) atunci
aux←a;
a←b;
b←aux;
n←a*100+b*10+c;
scrie n;
	{
	aux=a;
	a=b;
	b=aux;
	}
	if (b<c)
	{
	aux=b;
	b=c;
	c=aux;
	}
	if (a<b)
	{
	aux=a;
	a=b;
	b=aux;
	}
	n=a*100+b*10+c;
	cout<<n;
	return 0;
	}

3. Se citesc trei numere naturale de la tastatură. Să se afişeze în ordine descrescătoare.

Pseudocod	C++

întreg n,a,b,c,aux;
citeşte a;
citeşte b;
citeşte c;
dacă (a>b) atunci
aux←a;
a←b;
b←aux;
	#include<iostream>
	using namespace std;
	int main ()
	{int n,a, b,c,aux;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	cout<<"c="; cin>>c;
	if (a>b)

Pseudocod	C++

dacă (b>c) atunci
aux←b;
b←c;
c←aux;
dacă (a>b) atunci
aux←a;
a←b;
b←aux;
scrie a, b, c;
	{
	aux=a;
	a=b;
	b=aux;
	}
	if (b>c)
	{
	aux=b;
	b=c;
	c=aux;
	}
	if (a>b)
	{
	aux=a;
	a=b;
	b=aux;
	}
	cout<<a<<" "<<b<<" "<<c;
	return 0;
	}

4.2. Algoritmi pentru determinarea maximului / minimului

Algoritmul constă în atribuirea valorii primului element unei variabile care reprezintă maximului (minimului) şi compararea acestei valori cu elementele din şir.

Paşii algoritmului sunt:

1. se citeşte primul număr (a)
2. se atribuie variabilei ce reprezintă maximul valoarea primului număr (max=a)
3. se citeşte următorul număr (tot în variabila a)
4. dacă valoarea numărului citit (a) este mai mare decât valoarea ce reprezintă maximul (max) atunci acesteia i se atribuie valoarea numărului citit (a)
5. dacă mai sunt numere de citit se revine la pasul 3

Exemple

1. Se introduce un şir de n numere întregi de la tastatură. Să se afişeze maximul dintre aceste numere.

Pseudocod	C++

întreg n,a,i, max;
citeşte n;
citeşte a;
max←a;
pentru i←2, n execută
citeşte a;
dacă (max<a) atunci
max←a;
scrie max;
	#include<iostream>
	using namespace std;
	int main ()
	{int n, i, a,max;
	cout<<"n="; cin>>n;
	cout<<"a="; cin>>a;
	max=a;
	for (i=2;i<=n;i++)
	{cout<<"a="; cin>>a;
	if (max<a)
	max=a;
	}
	cout<<"max="<<max;
	return 0;}

2. Se introduce un şir de numere de la tastatură până la întâlnirea valorii 0. Să se afişeze maximul dintre aceste numere.

Pseudocod	C++

întreg a,max;
citeşte a;
dacă (a!=0) atunci
max←a;
cât timp (a!=0) execută
citeşte a;
dacă (max<a) atunci
max←a;
scrie max;
	#include<iostream>
	using namespace std;
	int main ()
	{int a,max;
	cout<<"a="; cin>>a;
	if(a!=0)
	max=a;
	while (a!=0)
	{cout<<"a="; cin>>a;
	if (max<a)
	max=a;
	}
	cout<<"max="<<max;
	return 0;
	}

3. Se introduc de la tastatură n numere. Să se afişeze valoarea minimă şi de câte ori apare în şir.

Pseudocod	C++

întreg a,min,n, i, nr;
citeşte n;
citeşte a;
nr←1;
min←a;
pentru i←2, n execută
citeşte a;
dacă (a==min) atunci
nr←nr+1;
altfel
dacă (a<min) atunci
	#include<iostream>
	using namespace std;
	int main ()
	{int a,min, n,i,nr;
	cout<<"n="; cin>>n;
	cout<<"a="; cin>>a;
	nr=1;
	min=a;
	for (i=2;i<=n;i++)
	{cout<<"a="; cin>>a;
	if (a==min)

Pseudocod	C++

min←a;
nr←1;
 	
scrie min;
scrie " apare de " nr;
	nr=nr+1;
	else
	if (a<min)
	{min=a;
	nr=1;}
	}
	cout<<"min="<<min;
	cout<<" apare de "<<nr;
	return 0;
	}

4.3. Algoritmi pentru prelucrarea cifrelor unui număr

Pentru prelucrarea cifrelor unui număr puteţi folosi următorii algoritmi:

1. algoritmul pentru extragerea cifrelor unui număr;
2. algoritmul pentru compunerea unui număr din cifrele sale
3. algoritmul pentru determinarea inversului unui număr

4.3.1. Algoritmul pentru extragerea cifrelor unui număr

Algoritmul determină cifrele unui număr n, prin extragerea pe rând a fiecărei cifre c (începând cu cifra unităţilor) cu operaţia n%10 (n mod 10) şi eliminând din număr a cifrei extrase cu

operaţia n/10 (n div 10). Aceste operaţii se execută cât timp mai există cifre de extras din n (n!=0).

Principiul de funcţionare

Cât timp numărul n !=0 se executa următorii paşi:

1. Se extrage ultima cifră cu operaţia n%10
2. Se prelucrează cifra
3. Se elimină ultima cifră din număr prin operaţia n=n/10

Exemple:

1. Se citeşte un număr natural n. Să se afişeze suma cifrelor sale.

Pseudocod	C++

întreg n, s;
citeşte n;
s←0;
cât timp (a!=0) execută
s←s+n%10;
n←n/10;
scrie s;
	#include<iostream>
	using namespace std;
	int main ()
	{int n,s=0;
	cout<<"n="; cin>>n;
	while (n!=0)
	{s=s+n%10;
	n=n/10;}
	cout<<s;
	return 0;
	}

2. Se citeşte un număr natural n. Să se afişeze produsul cifrelor pare ale numărului.

Pseudocod	C++
întreg n, p; citeşte n; p←1; cât timp (a!=0) execută	#include<iostream> using namespace std; int main () {int n,p=1;

Pseudocod	C++

dacă (n%2==0) atunci
p←p+n%10;
n←n/10;
scrie p;
	cout<<"n="; cin>>n;
	while (n!=0)
	{if(n%0==0)
	p=p*n%10;
	n=n/10;}
	cout<<p;
	return 0;
	}

3. Se introduce de la tastatură un şir de numere naturale, până la citirea valorii 0. Să se afişeze toate perechile de numere introduse consecutiv care au proprietatea că al doilea număr este egal cu suma cifrelor primului număr.

Pseudocod	C++

întreg a, b, x, s;
citeşte a;
citeşte b;
cât timp (b!=0) execută
s←0;
x←a;
cât timp (b!=0) execută
s←s+n%10;
x←x/10;
dacă (s==b) atunci
scrie a, b;
a←b;
citeste b;
	#include<iostream>
	using namespace std;
	int main ()
	{int a, b,x, s;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	while (b!=0)
	{
	s=0;
	x=a;
	while (x!=0)
	{s=s+x%10;
	x=x/10;}
	if (s==b)
	cout<<a<< " "<<b<<endl;
	a=b;
	cout<<"b="; cin>>b;
	}
	return 0;}

4.3.2. Algoritmul pentru compunerea unui număr din cifrele sale

Metoda 1

Algoritmul foloseşte reprezentarea numărului în baza 10:

$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = a_n * 10^n + a_{n-1} * 10^{n-1} + \ldots + a_1 * 10^1 + a_0 * 10^0$

Paşii algoritmului:

1. Se citeşte numărul de n cifre
2. Se calculează p ca fiind 10^{n-1} (p=pow(10,n-1))
3. Se iniţializează numărul nr=0
4. În instrucţiunea repetitivă cu un număr cunoscut de paşi sunt efectuate instrucţiunile:
   - se citeşte cifra **c**
   - se adună la numărul **nr** produsul c * p
   - se decrementează puterea lui 10 prin operaţia p=p / 10
5. se citeşte numărul nr.

Pseudocod	C++

întreg n, i, p, c, nr;
citeşte n;
citeşte b;
nr← 0;
p← pow(10,n-1);
pentru i ← 1, n execută
citeşte c;
nr←nr+c*p;
p←p/10;
scrie nr;
	#include<iostream>
	#include<cmath>
	using namespace std;
	int main ()
	{int nr=0, n, i,p,c;
	cout<<"n=";
	cin>>n;
	p=pow(10,n-1);
	for (i=1;i<=n;i++)
	{
	cout<<"c=";
	cin>>c;
	nr=nr+c*p;
	p=p/10;
	}
	cout<<nr;
	return 0;}

Metoda 2

Algoritmul foloseşte următoarea reprezentare a numărului:
$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = (\ldots((a_n*10+a_{n-1})*10+a_{n-2})*10+\ldots+a_1)*10+a_0$

Paşii algoritmului:

1. Se iniţializează numărul **nr** cu 0
2. Se citește cifra c
3. Cât timp **c** este cifra (c>=0 && c<=9)
   - se adună la nr*10 cifra citită (nr=nr*10+c)
   - se citește cifra c
4. Se afişează numărul **nr.**

Pseudocod	**C++**

întreg c, nr;
citeşte c;
nr← 0;
cât timp(c>=0 && c<=9)
execută
nr←nr*10+c;
citeşte c;
scrie nr;
	#include<iostream>
	using namespace std;
	int main ()
	{int nr, c;
	nr=0;
	cout<<"c=";
	cin>>c;
	while (c>=0 && c<=9)
	{nr=nr*10+c;
	cout<<"c=";
	cin>>c;
	}
	cout<<nr;
	return 0;
	}

4.3.3. Algoritmul pentru determinarea inversului unui număr

Algoritmul pentru determinarea inversul numărului n se realizează prin extragerea pe rând a fiecărei cifre începând cu cifra unităţilor din numărul n şi compunerea unui nou număr cu aceste cifre.

Paşii algoritmului:

1. se citeşte numărul n
2. se iniţializează variabila inv cu 0 (inv=0)
3. cât timp n!=0 sunt executate următoarele instrucţiunile:
   - se extrage ultima cifră din numărul n şi se adună cu numărul inv*10 prin operaţia inv=inv*10+n%10
   - se elimină ultima cifră din număr
4. Se afişează variabila inv care reprezintă numărul invers a numărului citit de la tastatură

Pseudocod	C++

întreg n, inv;
citeşte n;
inv← 0;
cât timp (n!=0) execută
inv←inv*10+n%10;
n←n/10;
scrie inv;
	#include<iostream>
	using namespace std;
	int main ()
	{int n, inv;
	cout<<"n="; cin>>n;
	inv=0;
	while (n!=0)
	{inv=inv*10+n%10;
	n=n/10;}
	cout<<inv;
	return 0;}

Exemple:

1. Se citeşte un număr natural de la tastatură n. Să se verifice dacă numărul este palindrom. (un număr este palindrom dacă,

citit de la stânga la dreapta şi de la dreapta la stânga are aceeaşi valoare).

Pseudocod	C++

întreg n, inv, m;
citeşte n;
m← n;
inv← 0;
cât timp (m!=0) execută
inv←inv*10+m%10;
m←m/10;
dacă (n==inv)
scrie ”este palindrom”
altfel
scrie ”nu este palindrom”
scrie inv;
	#include<iostream>
	using namespace std;
	int main ()
	{int n, inv,m;
	cout<<"n="; cin>>n;
	m=n;
	inv=0;
	while (m!=0)
	{inv=inv*10+m%10;
	m=m/10;}
	if (n==inv)
	cout<<”este palindrom”;
	else
	cout<<”nu este palindrom”;
	return 0;
	}

2. Să se afişeze toate numerele care sunt palindrom şi care aparţin intervalului [a.b]. Valorile a şi b se citesc de la tastatură.

Pseudocod	C++

întreg a,b, inv, x;
citeşte a;
citeşte b;
pentru i←a,b execută
inv← 0;
x← i;
cât timp (x!=0) execută
inv←inv*10+x%10;
x←x/10;
dacă (i==inv)
scrie i
	#include<iostream>
	using namespace std;
	int main ()
	{int a,b,i,inv,x;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	for (i=a;i<=b;i++)
	{inv=0;
	x=i;
	while (x!=0)
	{inv=inv*10+x%10;
	x=x/10;}

	if (i==inv)
	cout<<i<<" ";
	}
	return 0;}

3. Se citeşte un şir de n numere naturale. Să se afişeze cele care sunt palindroame.

Pseudocod	C++

întreg n, i, inv, x, y;
citeşte n;
pentru i←1,n execută
citeşte x;
inv← 0;
y← x;
cât timp (y!=0) execută
inv←inv*10+y%10;
y←y/10;
dacă (x==inv)
scrie x
	#include<iostream>
	using namespace std;
	int main ()
	{int n,i,inv,x,y;
	cout<<"n="; cin>>n;
	for (i=1;i<=n;i++)
	{cout<<"x="; cin>>x;
	inv=0;
	y=x;
	while (y!=0)
	{inv=inv*10+y%10;
	y=y/10;}
	if (x==inv)
	cout<<y<<" ";
	}
	Return 0;}

4. Să se afişeze toate numerele din intervalul [a,b] care au suma cifrelor un număr par. Valorile a şi b sunt citite de la tastatură.

Pseudocod	C++

întreg a,b,i, x, S;
citeşte a;
citeşte b;
pentru i←a,b execută
S← 0;
x← i;
cât timp (x!=0) execută
S←S+x%10;
x←x/10;
	#include<iostream>
	using namespace std;
	int main ()
	{int a,b,i,S,x;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	for (i=a;i<=b;i++)
	{S=0;
	x=i;

dacă (S%2==0)
  scrie i
	while (x!=0)
	  {S=S+x%10;
	  x=x/10;}
	if (S%2==0)
	cout<<i<<" ";
	}
	return 0;}

5. Se citeşte un număr natural. Să se afişeze inversul sumei cifrelor sale.

Pseudocod	C++

întreg n, S, inv;
citeşte n;
S← 0;
cât timp (n!=0) execută
  S←S+n%10;
  n←n/10;
inv← 0;
cât timp (S!=0) execută
  inv←inv+S%10;
  S←S/10;
scrie inv
	#include<iostream>
	using namespace std;
	int main ()
	{int n,S,inv;
	cout<<"n="; cin>>n;
	S=0;
	while (n!=0)
	  {S=S+n%10;
	  n=n/10;}
	inv=0;
	while (S!=0)
	  {inv=inv*10+S%10;
	  S=S/10;}
	cout<<inv;
	return 0;}

4.4. Algoritmul pentru calcularea cmmdc

Pentru calcularea celui mai mare divizor comun dintre două numere naturale nenule, se folosesc următorii algoritmi:

- Algoritmul lui Euclid
- Algoritmul de scădere repetată

4.4.1. Algoritmul lui Euclid

Paşii algoritmului sunt:

1. Se citesc variabilele a şi b;
2. Cât timp b este diferit de 0 se vor executa instrucţiunile:
   - în variabila r se calculează restul împărţirii lui a la b (a%b)
   - se vor face atribuirile a=b şi b=r
3. Se afişează a

Pseudocod	C++

întreg a,b,r;
citeşte a;
citeşte b;
cât timp (b!=0) execută
r←a%b;
a←b;
b← r;
scrie a
	#include<iostream>
	using namespace std;
	int main()
	{int a, b, r;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	while (b!=0)
	{r=a%b;
	a=b;
	b=r;
	}
	cout<<a;
	return 0;}

4.4.2. Algoritmul de scădere repetată

Paşii algoritmului sunt:

1. Se citesc variabilele a şi b;
2. Cât timp a este diferit de b se vor executa instrucţiunile
   a. Dacă a>b atunci a=a-b, altfel b=b-a
3. Se afişează a

Pseudocod	C++

întreg a,b;
citeşte a;
citeşte b;
cât timp (a!=b) execută
dacă (a>b) atunci
a←a-b;
altfel
b←b-a;
scrie a
	#include<iostream>
	using namespace std;
	int main()
	{int a, b;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	while (a!=b)
	if (a>b)
	a=a-b;
	else
	b=b-a;
	cout<<a;
	return 0;}

Exemplu:

Se citesc de la tastatură două numere naturale. Să se afişeze pe ecran cel mai mic multiplu comun

Fie a şi b numerele naturale. Pentru aceste numere are loc următoarea relaţie:

a*b= c.m.m.d.c.(a,b)* c.m.m.m.c.(a,b)

formulă ce permite astfel determinarea şi a celui mai mic multiplu comun a acestor două numere.

Pseudocod	C++

întreg a,b,p;
citeşte a;
citeşte b;
p←a*b;
cât timp (a!=b) execută
dacă (a>b) atunci
a←a-b;
altfel
b←b-a;
scrie p/a
	#include<iostream>
	using namespace std;
	int main()
	{int a, b,p;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	p=a*b;
	while (a!=b)
	if (a>b)
	a=a-b;
	else

	b=b-a;
	cout<<p/a;
	return 0;
	}

4.5. Algoritmul pentru testarea unui număr prim

Un ***număr prim*** este numărul care are ca divizori pe 1 şi pe el însuşi.

Verificarea unui număr dacă este prim sau nu se face astfel:

1. Se citeşte numărul de la tastatură
2. Variabilei prim i se atribuie valoarea 1 (prim =1)
3. Se verifică dacă numărul citit este 0 sau 1
   - în caz afirmativ prim =0
4. Se verifică dacă numărul n are divizori în intervalul [2, n/2]
   - în caz afirmativ prim =0
5. se verifică dacă variabila prim =1
   - în caz afirmativ se afişează ”numărul este prim”
   - în caz contrar se afişează ”numărul nu este prim”

Pseudocod	C++

întreg n, prim,i;
citeşte n;
prim←1;
dacă (n==0 || n==1) atunci
prim←0;
pentru i ←2,n/2 execută
dacă (n%i==0) atunci
prim←0;
dacă (prim==1) atunci
scrie Numarul este prim;
altfel
	#include<iostream>
	using namespace std;
	int main ()
	{int n, prim, i;
	cout<<"n="; cin>>n;
	prim=1;
	if (n==0||n==1)
	prim=0;
	for (i=2;i<=n/2;i++)
	if (n%i==0)
	prim=0;

scrie Numărul nu este prim;	if (prim==1)
	cout<<"numarul este prim";
	else
	cout<<"numarul nu este prim";
	return 0;
	}

4.6. Algoritmii pentru prelucrarea divizorilor ai unui număr

Algoritmii pentru prelucrarea divizorilor ai unui număr sunt următori:

- algoritmul pentru generarea divizorilor proprii ai unui număr
- algoritmul pentru generarea divizorilor primi ai unui număr

4.6.1. Algoritmul pentru generarea divizorilor proprii ai unui număr

Paşii algoritmului sunt:

1. Se citeşte numărul de la tastatură
2. Se verifică dacă restul împărţirii lui n la i este 0, unde i ia valori din intervalul [1,n]
   - în caz afirmativ se afişează i

Pseudocod	C++

întreg n,i;
citeşte n;
pentru i ←1,n execută
dacă (n%i==0) atunci
scrie i
	#include<iostream>
	using namespace std;
	int main()
	{int n, i;
	cout<<"n="; cin>>n;
	for (i=1;i<=n;i++)
	if (n%i==0)
	cout<<i<<" ";
	return 0;}

4.6.2. Algoritmul pentru generarea divizorilor primi ai unui număr

Paşii algoritmului sunt:

1. Se citeşte numărul n de la tastatură
2. i=2
3. cât timp n!=1 se execută
   - ✓ se verifică dacă restul împărţirii lui n la i este 0
     - în caz afirmativ
       - se afişează i
       - cat timp n%i=0 se executa instrucţiunea n=n/i
   - ✓ i=i+1

Pseudocod	C++

întreg n,i;
citeşte n;
i ←2;
cât timp (n!=1) execută
dacă (n%i==0) atunci
scrie i
cât timp (n%i==0)
n ←n/i;
i ←i+1;
	#include<iostream>
	using namespace std;
	int main()
	{int n, i;
	cout<<"n="; cin>>n;
	i=2;
	while (n!=1)
	{if (n%i==0)
	{cout<<i<<" ";
	while(n%i==0)
	n=n/i;
	}
	i=i+1;
	}
	return 0;
	}

BIBLIOGRAFIE

1. Emanuela Cerchez, Marinel Şerban – Programarea în limbajul C/C++ pentru liceu, Editura Polirom, 2005
2. Pavel Florin Moraru – Informatica pentru liceu şi bacalaureat, Editura Donaris, 2009.
3. Clara Ionescu, Adina Bălan – Informatica pentru grupe de performanţă, Editura Dacia, 2004.
4. Mariana Miloşescu – Manual clasa a IX-a Informatică, profil real, Editura Didactică şi Pedagogică, 2004