MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA

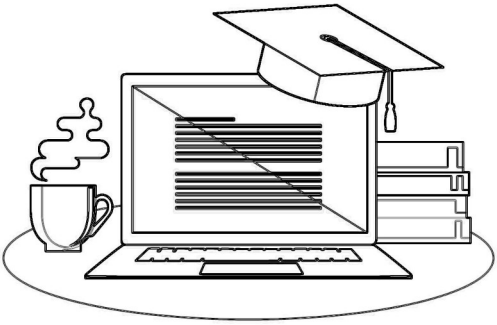
IP CENTRUL DE EXCELENȚĂ ÎN INFORMATICĂ ȘI TEHNOLOGII INFORMAȚIONALE

CATEDRA INFORMATICA II

ANDRIAN DASCAL

PROGRAMARE STRUCTURATĂ ȘI PROCEDURALĂ ÎN C++

Îndrumar pentru activitățile teoretice, destinat cadrelor didactice și elevilor din IP CEITI



CHIȘINĂU, 2022

Elaborat conform Curriculumului modular „Programarea structurată” și „Programarea procedurală” reactualizate în 2020. Discutat și examinat la ședința catedrei „Informatica II” din 18.01.2023, Proces Verbal Nr. 5, șef catedră, Luminița Gribineț

Autor:

Andrian Dascal, magistru în Informatică și Tehnologii Informaționale, grad didactic II, IP CEITI.

Redactare științifică:

Ioan Jeleascov, magistru în Informatică și Tehnologii Informaționale, consilier în proiectarea arhitecturii software în cadrul unității "StoneHard" din Londra, Marea Britanie.

Toate drepturile asupra acestei ediții aparțin autorului. Orice tipărire sau retipărire fără acordul în scris al autorului atrage răspunderea potrivit legii.

© Andrian Dascal, 2022

DRAGI PRIETENI,

„Cărțile care te ajută cel mai mult sunt cele care te provoacă să gândești cel mai mult, iar pe cele mai bune dintre ele ar trebuie lipit un avertisment: „Atenție! Îți poate schimba modul de a gândi !” ”. Theodore Parker

În prezent lucrurile s-au schimbat mul față de ultimii zece ani, iar tehnologia a avansat în punctul în care majoritatea oamenilor dețin un dispozitiv smartphone. Trăim într-o lume în care computerele se pot găsi peste tot și majoritatea dintre ele sunt conectate între ele prin rețeaua internet. Calculatoarele sunt utilizate peste tot în jurul nostru, dar mulți dintre utilizatori nu cunosc conceptul de programare.

Un limbaj de programare este un set bine definit de expresii și reguli valide de formulare a instrucțiunilor pentru un computer. Un limbaj de programare are definite un o mulțime de reguli sintactice. El oferă posibilitatea programatorului să specifice acțiunile pe care trebuie să le execute calculatorul, în ce ordine și cu ce date. Specificarea constă practic în întocmirea/scrierea secvențelor de cod necesare, aceasta înseamnă a programa.

Unele dintre motivele principale pentru care ar fi bine să înveți a programa sunt: te va învăța lecții din viață (de exemplu, să analizezi unele erori de cod și să înveți să le înlături corect); te va învăța să gândești logic (de exemplu, Steve Jobs spunea: “Toata lumea din aceasta tara ar trebui sa invete programare, pentru ca aceasta te invata defapt cum sa gândești”); te învață să scrii și să citești mai rapid; poate fi un hobby plăcut și interactiv; are un rol important în societatea de azi; te ajută să-ți îmbunătățești aptitudinile; te ajută la înțelegerea funcționării site-urilor; te ajută să rezolvi mai ușor probleme de un nivel sporit de dificultate; îți lărgește orizonturile gândirii și dacă să fim realiști, vei avea un loc de muncă bine plătit.

Prezenta lucrare reprezintă un suport teoretic având drept scop însușirea de către elevi a cunoștințelor necesare gândirii logice și formării culturii informaționale. Acest îndrumar este destinat cadrelor didactice debutante și elevilor care doresc să studieze un limbaj de programare de nivel înalt. Mult succes la fiecare!

Autorul

CUPRINSUL

INTRODUCERE 5

1. CONCEPTE FUNDAMENTALE
1.1 Vocabularul limbajului. Conceptul de dată. 7
1.2 Operatori. Operaţii de intrare/ieşire la consolă 19
1.3 Tipuri de instrucţiuni: vide, simple şi compuse 34

2. STRUCTURI DE CONTROL
2.1 Instrucţiuni de decizie simplă şi multiplă 43
2.2 Instrucţiuni ciclice. Particularităţi 49

3. TIPURI DE DATE STRUCTURATE
3.1 Tablouri unidimensionale şi bidimensionale 56
3.2 Şiruri de caractere. Funcţii standard. Conversii 68
3.3 Pointeri. Alocarea dinamică a memoriei 77
3.4 Fişiere. Funcţii de prelucrare a datelor 89
3.5 Metode directe de sortare a datelor 99
3.6 Structuri. Tablou de structuri 109

4. SUBPROGRAME ŞI MODULE
4.1 Noţiune de subprogram. Efecte colaterale 120
4.2 Recursivitate. Subprograme recursive 133
4.3 Module. Crearea şi implementarea modulelor 138

BIBLIOWEBOGRAFIE 146

ANEXE
A1 Testul de autoevaluare nr.1 – Expresii şi formule 147
A2 Testul de autoevaluare nr.2 – Structuri de control 150
A3 Testul de autoevaluare nr.3 – Tipuri e date structurate 153
A4 Testul de autoevaluare nr.4 – Subprograme şi module 155
A5 Modele de itemi pentru pregătirea de olimpiadă 157

INTRODUCERE

C++ rulează pe o varietate de platforme, cum ar fi Windows, Mac OS și diferite versiuni de UNIX. Limbajul C++ este o necesitate pentru studenți și profesioniști care lucrează pentru a deveni ingineri software.

Voi enumera câteva dintre avantajele cheie ale învățării limbajului C++:

(1) C++ este foarte aproape de partea hardware, deci aveți șansa de a lucra la un nivel profund (*low level programming*), ceea ce vă oferă mult control în ceea ce privește gestionarea memoriei, performanță mai bună și o dezvoltare software robustă.

(2) Programarea C++ vă oferă o înțelegere clară despre programarea orientată pe obiecte. Veți înțelege implementarea la nivel profund a polimorfismului atunci când veți implementa tabele virtuale și indicatori de tabele virtuale sau identificarea tipului dinamic.

(3) C++ este unul dintre limbajele de programare ecologice și iubit de milioane de dezvoltatori de software. Dacă ești un programator C++ grozav, atunci nu vei sta niciodată fără muncă și mai important, vei fi bine plătit pentru munca ta.

(4) C++ este cel mai utilizat limbaj de programare în programarea de aplicații și de sistem. Astfel, puteți alege domeniul dvs. de interes de dezvoltare software.

(5) C++ chiar vă învață diferența dintre compilator, linker și încărcător, diferite tipuri de date, clase de stocare, tipuri de variabile și domeniile lor, etc.

Există foarte multe motive pentru a învăța programarea în C++. Dar un lucru este sigur, pentru a învăța orice limbaj de programare, nu doar C++, trebuie doar să scrieți cod, să scrieți cod permanent.

Este foarte important să rezolvați multe probleme nu simple, ci probleme de diverse gradede complicitate și să continuați așa până deveniți un expert în scrierea de cod.

Capitolul 1

CONCEPTE FUNDAMENTALE

Introducere

Orice limbaj constituie un mijloc de comunicare între două entităţi: emiţătorul şi receptorul. În general limbajele sunt de două tipuri: naturale şi artificiale. Limbajele naturale s-au constituit de-a lungul timpului, în procesul conlucrării membrilor societăţii. Limbajele artificiale au fost şi sunt create pentru comunicarea într-un domeniu particular de activitate.

Limbajele de programare fac parte din categoria limbajelor artificiale, fiind utilizate în procesul de comunicare om-calculator. Un limbaj de programare reprezintă un mijloc de comunicare între programator şi calculator. Acest limbaj este un mijloc de comunicare particular, în care informaţia ce trebuie comunicată este codificată printr-un program pe baza a trei componente: un *set de acţiuni*, care acţionează asupra unui *set de date* într-o anumită *secvenţă de acţionare*.

La începutul anilor 70 a apărut limbajul C – creaţia lui Dennis Ritchie şi Brain Kernighan. Limbajul C++ este creaţia lui Bjarne Stroustrup şi reprezintă o extensie a limbajului C care permite programarea pe obiecte.

Realizarea unui program scris în C++ necesită parcurgerea a patru etape:

(1) *editare* – scrierea programului sursă, prin crearea unui fişier cu extensia cpp;
(2) *compilare* – se aduce în memoria internă programul sursă, se verifică erori şi se converteşte acest program în program obiect, având extensia obj;
(3) *link-editare* – se leagă programul obiect cu bibliotecile de sistem şi se transformă într-un program executabil având extensia exe;
(4) *execuţie* – se lansează în execuţie programul obiect: se efectuează citirea datelor, calculele şi scrierea rezultatelor, formându-se fişierul.

1.1 Vocabularul limbajului. Conceptul de dată.

În această temă vom analiza următoarele unități de conținut:
- *structura generală a unui program în limbajul C++;*
- *vocabularul limbajului C++ și unitățile sale lexicale;*
- *conceptul de dată și declararea tipurilor de date: întregi, reale și de tip caracter;*
- *conceptul de constante și declararea acestora.*

Limbajul C++ este caracterizat de trei elemente:
(1) *sintaxă* – este formată din totalitatea regulilor de scriere corectă a programelor;
(2) *semantică* – reprezintă semnificaţia construcţiilor corecte din punct de vedere sintactic;
(3) *vocabular* – este format din totalitatea cuvintelor care pot fi folosite într-un program.

Cel mai bun mod de a învăța un limbaj de programare este de a scrie cod. Deși foarte simplu, din analiza exemplului putem observa componentele fundamentale pe care un program C++ le are. Vom explica în continuare și structura codului, linie cu linie !

```
// primul meu program C++
#include using namespace std;
int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

- *Linia 1:* // primul meu program C++

Cele doua slash-uri indică un comentariu, pe o singură linie. Astfel, restul elementelor de pe linie nu vor avea vreo influență asupra execuției programului. Sunt folosite, fie pentru a include scurte explicații despre cod, fie pentru a suspenda, în procesul de depănare sau de execuție a unei linii.

- *Linia 2:* #include

Linile precedate de semnul # sunt directive, fiind interpretate de ceea ce se numeste preprocesor, înainte de compilarea programului. În acest caz, directiva include comunică preprocesorului să includă o secțiune de cod C++, anume headerul iostream, care permite realizarea operațiilor de intrare / ieșire, precum scrierea textului "Hello World !" pe ecran.

- *Linia 3:* using namespace std;

Numai o singură entitate poate exista cu un anumit nume într-un anumit domeniu de aplicare. Spațiul de nume ne permite să grupăm entități numite care ar avea un domeniu de aplicare global în domenii mai restrânse, oferindu-le un domeniu de nume. Aceasta permite organizarea elementelor de programe în diferite domenii logice la care se face referire prin nume.

- *Linia 4:* int main ()

Această linie inițiază declararea unei funcții (subprogram). În principiu, o funcție este un grup de instrucțiuni, asociate sub un nume. Acesta este, însă, subiectul unui capitol ulterior. Funcția main() este specială în C++, fiind prima executată la executarea programului, indiferent de locul în care aceasta este scrisă în codul sursă.

- *Liniile 5 și 8:* { și }

Acolada deschisă la linia 5 indică faptul că urmează definiția unei funcții, așa cum, cea închisă la linia 7 indică faptul că acolo se termină. Ce este cuprins între ele constituie corpul funcției, care determină efectul execuției ei. De asemenea așa acolade pot fi aplicate atunci când indică corpul unei instrucțiuni compuse de decizie sau ciclice.

- *Linia 6:* cout << "Hello World!" << endl;

Această linie conține o instrucțiune C++. Execuția instrucțiunilor se face în ordinea în care apar în corpul funcției. Instructiunea de față are trei părți. Prima parte, cout, indică dispozitivul de ieșire. A doua parte, operatorul „<<”, numit de inserție, arată că ceea ce urmează va fi inserat în cout. A doua parte, "Hello world !", este conținutul care urmează a fi afișat, iar endl de la sfârșit ne indică trecerea din rând nou.

- *Linia 7:* return 0;

Din punct de vedere tehnic, funcția principală în C sau C ++ trebuie să returneze o valoare, deoarece este declarată ca "int principal", ceea ce înseamnă că "funcția principală ar trebui să returneze tipul de date întreg", dacă principalul este declarat "void", atunci nu este nevoie de "return 0".

Vocabularul limbajului C++ este format din: setul de caractere, identificatori, cuvinte cheie, comentarii și separatori.

1.1.1 Setul de caractere

Setul de caractere utilizat pentru scrierea programelor C++ este setul de caractere al codului ASCII. Codul ASCII este format din:
(1) literele mari şi mici ale alfabetului latin (A-Z, a-z);
(2) cifrele sistemului de numeraţie zecimal (0-9);
(3) caracterele speciale (blank, +, *, %, =, {, !, #, etc.).

Există anumite caractere în C++ când sunt precedate de o bară oblică inversă, ele vor avea o semnificație specială. Unele sunt prezentate astfel:

Tabelul 1.1: Simboluri și semnificația lor

Nr.	Simbolul	Semnificația
1	\b	Backspace
2	\t	Tab orizontal
3	\v	Tab vertical
4	\n	Linie nouă
5	\f	Pagina nouă – formfeed
6	\r	Început de rând
7	\”	Ghilimele
8	\’	Apostrof
9	\\	Backslash
10	\?	Semnul întrebării
11	\a	Alarmă

Standarde de codificare a seturilor de caractere:
(1) EBCDIC (Extended Binary Coded Decimal Interchenge Code), un cod pe 8 biți, introdus de IBM;

(2) ASCII (American Standard Code for Information Interchange), introdus de ANSI (American National Standard Institute), este un cod pe 7 biți și permite codificarea a 128 de caractere (95 de caractere afișabile și 33 de caractere neafișabile, numite caractere de control).

1.1.2 Identificatori

Identificatorii (numele) au rolul de a denumi elemente ale programului precum constante, variabile, funcţii, etc. Identificatorii reprezintă o secvenţă de litere, cifre şi "_" (linia de subliniere), ei trebuie să înceapă cu "_" sau cu o literă. Ei nu pot fi cuvinte cheie (rezervate) ale limbajului. De asemenea ei reprezintă o succesiune de litere, cifre sau caracterul special "-" din care prima nu trebuie să fie cifră, iar restul caractere speciale în afară de "_".

Exemple	Explicații
corect: a1, var1, sum, etc.	contine cifra 1
incorect: 1var, mt&p, etc.	conține caracterul special &.

1.1.3 Cuvinte cheie / rezervate

Cuvintele cheie au un înţeles bine definit şi nu pot fi folosite în alt context. Pe baza caracterelor ce alcătuiește alfabetul limbajului, se alcătuiesc cuvintele care formează vocabularul sau lexicul limbajului și cu ajutorul cărora se construiesc expresiile și instrucțiunile limbajului.

Există doua categorii de cuvinte și anume:
(1) *cuvinte cheie* – acestea au un înțeles explicit într-un context concret;
(2) *cuvinte rezervate* – acestea nu pot fi folosite decât în scopul pentru care au fost definite.

Avantajele utilizării *cuvintelor rezervate* sunt următoarele:
- programul devine mai ușor de înțeles;
- se mărește viteza de compilare (analiza lexicală, sintactică și semantică este mai simplă la căutarea în tabele de simboluri);
- erorile sunt mai ușor de depistat.

Pe de altă parte, în cadrul unui limbaj de programare se vor utiliza cuvinte rezervate și cuvinte definite de utilizator, pentru a face referire la diverse variabile, fișiere, nume de subprograme, etc. Unele sunt prezentate astfel:

Tabelul 1.2: Cuvinte rezervate în C++

asm	do	if	return	typedef
auto	double	inline	short	typeid
bool	dynamic_cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit	namespace	static_cast	using
char	export	new	struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	while
default	friend	register	true	
delete	goto	reinterpret_cast	try	

1.1.4 Comentarii și separatori

Pentru ca un program să fie ușor de înțeles și pentru a evidenția unele notițe succinte într-o secvență de cod C++ se folosesc comentariile. Acestea sunt texte care vor fi ignorate de compilator. În limbajul C++ deosebim două tipuri de comentarii:

(1) /* comentariul va cuprinte mai multe linii de cod */
(2) // comentariul va cuprinte doar o linie de cod

Separatorii se folosesc pentru a delimita unitățile sintactice. Unii sunt prezentați astfel:

Tabelul 1.3: Separatori şi semnificaţia lor

Nr.	Simbolul	Semnificaţia
1	„ , ”	virgula este folosită pentru a separa identificatorii, expresiile, argumentele subprogramelor, etc.
2	„ ; ”	punct şi virgulă sunt folosite pentru a separa instrucţiunile C/C++ unele de altele
3	„ { ” „ } ”	acoladele sunt folosite pentru a separa instrucţiunile din interiorul blocului faţă de instrucţiunile din exteriorul blocului
4	„ [” „] ”	parantezele pătrate sunt folosite pentru a separa numele unui tablou de date de dimensiunea acestuia sau pentru a separa numele unui tablou de date de indexul unui element
5	„ ” „TAB”	Spaţiul şi TAB sunt folosite pentru a separa un element de program de celălalt şi pentru a creşte lizibilitatea codului

1.1.5 Conceptul de dată. Tipuri de date simple

Înainte de a scrie un program în orice limbaj de programare, trebuie să utilizăm diverse variabile pentru a stoca diverse informaţii. Variabilele nu sunt altceva decât locaţii de memorie rezervate pentru a stoca valori. Aceasta înseamnă că atunci când creaţi o variabilă rezervaţi un spaţiu în memorie.

Este posibil să dorim să stocăm informaţii de diferite tipuri de date, cum ar fi caracter, caracter lat, întreg, virgulă mobilă, virgulă mobilă dublă, boolean, etc. Pe baza tipului de date al unei variabile, sistemul de operare alocă memorie şi decide ce poate fi stocat în memorie rezervată.

Limbajul C++ oferă programatorului o gamă bogată de tipuri de date încorporate şi definite de utilizator. Următorul tabel listează şapte tipuri de date C++ de bază:

Nr.	Tipul de date şi descrierea lui	Cuvântul cheie
1	Boolean - stochează valoarea adevărată sau falsă.	bool

Nr.	Tipul de date și descrierea lui	Cuvântul cheie
2	Caracter - de obicei reprezintă un octet.	char
3	Întreg - cea mai naturală dimensiune a numărului întreg pentru sistemul de calcul.	int
4	Float - o valoare în virgulă mobilă cu precizie unică.	float
5	Double- o valoare în virgulă mobilă cu precizie dublă.	double
6	Void - fără valoare (tipul vid), reprezintă absența tipului.	void
7	WideChar - un tip de caracter extins	wchar_t

Mai multe dintre tipurile de bază pot fi modificate folosind unul sau mai mulți dintre acești modificatori de tip: signed, unsigned, short și long. În *Tabelul 1.4* sunt reprezentate tipurile de variabilă, câtă memorie este nevoie pentru a stoca valoarea în memorie și care este valoarea maximă și minimă care poate fi stocată într-un astfel de tip de variabile.

Tabelul 1.4: Tipurile de date și dimensiunile lor

Tipul	Dimensiunea în biți	Intervalul de date
char	1 octet = 8 biți	[-127, 127] sau [0, 255]
unsigned char	1 octet = 8 biți	[0, 255]
signed char	1 octet = 8 biți	[-127, 127]
int	4 octeți = 32 biți	[-2147483648, 2147483647]
unsigned int	4 octeți = 32 biți	[0, 4294967295]
signed int	4 octeți = 32 biți	[-2147483648, 2147483647]
short int	2 octeți = 16 biți	[-32768, 32767]
unsigned short int	2 octeți = 16 biți	[0, 65535]
signed short int	2 octeți = 16 biți	[-32768, 32767]

Tipul	Dimensiunea în biți	Intervalul de date
long int	8 octeți = 64 biți	[-2147483648, 2147483647]
signed long int	8 octeți = 64 biți	la fel ca long int
unsigned long int	8 octeți = 64 biți	[0, 4294967295]
long long int	8 octeți = 64 biți	[$-(2^{63})$, (2^{63}) -1]
unsigned long long int	8 octeți = 64 biți	de la 0 până la 18446744073709551615
float	4 octeți = 32 biți	
double	8 octeți = 64 biți	
long double	12 octeți = 96 biți	
wchar_t	8 biți sau 32 biți	un caracter extins

Limbajul C++ permite, de asemenea, definirea altor tipuri de variabile, pe care le vom acoperi în capitolele următoare, cum ar fi: enumerare, pointer, tablou de date, referință, structuri de date și clase. Următoarea secțiune va acoperi modul de definire, declarare și utilizare a diferitelor tipuri de variabile.

Putem crea un nume nou pentru un tip existent folosind *typedef*. Mai jos este sintaxa pentru a defini un nou tip folosind *typedef* :

```
typedef tip numeNou;
```

De exemplu, următoarea declarație îi spune compilatorului că *an-lumină* este un alt nume pentru *unsigned long long int*:

```
typedef unsigned long long int an_lumina;
```

Acum, următoarea declarație este perfect legală și creează o variabilă numită *an-lumină*:

```
an_lumina distanta;
```

1.1.6 Declararea variabilelor și a constantelor

O definiție de variabilă transmite compilatorului unde și ce spațiu de stocare să creeze pentru variabilă. O definiție de variabilă specifică un tip de date și conține o listă cu una sau mai multe variabile de acest tip, după cum urmează:

```
tip listă_de_variabile;
```

Aici, tipul trebuie să fie un tip de date C++ valid, inclusiv char, w_char, int, float, double, bool sau orice obiect definit de utilizator, etc., iar listă_de_variabile poate consta din unul sau mai multe nume de identificator separate prin virgule. Unele declarații valide sunt următoarele:

```
int i, j, k;
char c, ch;
float f, salariu;
dublu d;
```

Prima linie concomitent declară și definește variabilele i, j și k, deci instruiește compilatorului să creeze variabile numite i, j și k de tip int.

Important

- O variabilă nu poate fi utilizată în cadrul unui program fără a fi declarată. În cazul în care o variabilă nu este declarată, vom primi un mesaj de eroare în momentul compilării și lansării în execuție a programului.
- În denumirea variabilelor trebuie de ținut cont dacă literele folosite sunt majuscule sau minuscule. Astfel, o variabila „a” este diferită de o variabilă „A”.
- Variabilele care pot stoca valori non numerice, care sunt mai lungi de un singur caracter se numesc "șiruri de caractere".

Variabilele care sunt declarate în interiorul unei funcții sau bloc sunt variabile locale. Ele pot fi utilizate numai de instrucțiunile care se află în acea funcție sau bloc de cod. Variabilele locale nu sunt cunoscute de funcții în afara propriei lor.

Variabilele globale sunt definite în afara tuturor funcțiilor, de obicei deasupra programului. Ele își vor păstra valoarea pe toată durata de viață a programului și poate fi accesată de orice funcție, adică este disponibilă pentru utilizare în întregul program după declararea acesteia.

Calificatorii de tip oferă informații suplimentare despre variabilele pe care le preced.

Tabelul 1.5: Calificatorii de tip

Nr	Calificator	Descrierea
1	const	obiectele de tip *const* nu pot fi modificate de programul dumneavoastră în timpul execuției;
2	volatile	modificatorul *volatile* îi spune compilatorului că valoarea unei variabile poate fi modificată în moduri care nu sunt specificate în mod explicit de program;
3	restrict	un pointer calificat prin *restrict* este inițial singurul mijloc prin care obiectul către care indică poate fi accesat.

Urmează forma pentru a utiliza preprocesorul #define pentru a defini o constantă:

```
#define identificator valoarea
```

Putem utiliza prefixul *const* pentru a declara constante cu un anumit tip, după cum urmează:

```
const tip_variabilă = valoare;
```

Exemplul 1: Declararea / definirea variabilelor globale și locale. Moduri de definire a constantelor.

```
#include <iostream>
#define pi 3.141592653589793 // definirea constantei pi
using namespace std;
int suma; // declararea globala a variabilei
// programul principal
int main() {
    const int lungime = 10; // definirea constantei lungime
    int a, b; // declararea locala a variabilelor
    cout<<pi<<" "<<lungime;
}
```

Rezultat obținut la consolă după execuție:

```
3.14159 10
```

Exemplul 2: Suma a două variabile de tip int declarate implicit în cod. Modalități de afișare a rezultatului la consolă.

```
#include <iostream>
using namespace std;
int suma; // declararea globala a variabilei
// programul principal
int main() {
    int a=10, b=17; // declararea locala a variabilelor
    // Metoda 1 - afisarea la consola a sumei
    suma = a + b;
    cout<<a<<"+"<<b<<"="<<suma<<endl;
    // Metoda 2 - afisarea la consola a sumei
    cout<<a<<"+"<<b<<"="<<a+b<<endl;
    // Metoda 3 - afisarea la consola a sumei
    cout<<"Suma numerelor "<<a<<" si "<<b<<" este "<<a+b;
}
```

Rezultat obținut la consolă după execuție:

```
10+17=27
10+17=27
Suma numerelor 10 si 17 este 27
```

1.1 Modele de itemi pentru exersare

(1) Pentru următoarele probleme propuse, stabiliți care vor fi variabilele de intrare, variabilele de ieșire și explicați cum alegeți tipul lor de date:
   a) Ela are 15 portocale, iar Ionela are cu 15 portocale mai multe decât Ela. Câte mere au fetele împreună?
   b) Petru și Andrei împreună au parcurs cu bicicleta o distanță de 156 de kilometri. Andrei a parcurs o distanță de 2 ori mai mare decât Petru. Ce distanță a parcurs fiecare băiat ?
   c) Suma a cinci numere naturale consecutive este 510 ? Care sunt aceste cinci numere?
   d) Produsul a cinci numere naturale pare consecutive este 215.040? Care sunt aceste cinci numere?
   e) Alfabetul limbii engleze conține 26 de litere. Cum alculați cantitatea de informație în biți și apoi în octeți?

(2) Pentru următoarele probleme propuse, identificați care sunt constantele matematice și explicați cum le veți aplica în limbajul C++:
   a) Eva a desenat un cerc cu raza de 5 cm. Ce lungime, în decimetri, va avea cercul desenat de Ela? (*Răspuns:* $\pi \approx 3.141592653589793...$)
   b) Profesorul propune de desenat la tablă graficul unei funcții exponențiale, $f(x)=e^{2x+1}$, pentru aceasta este important de determinat câteva valori ale funcției în punctul x, valoare lui x poate fi cuprinsă, de exemplu, în intervalul [0, 100]. (*Răspuns:* $e \approx 2,7182818284...$)

(3) Funcția `Beep(x,y)` este folosită pentru injectoare și programe în cazul că acestea se realizează cu succes. Valoarea x este frecvența sunetului și valoarea y este durata în milisecunde până la activarea sunetului. Un exemplu de sunet ar fi: `Beep(900,1000);Sleep(300);`. Să nu uităm de includerea fișierului antet `<windows.h>` pentru funcționalitaea codului. Elaborați un program care va crea o melodie ce va include minim 15 aplicații ale funcției.

1.2 Operatori. Operaţii de intrare/ieşire la consolă

În această temă vom analiza următoarele unități de conținut:
- *operatori și precedența operatorilor;*
- *conversia explicită și expresii;*
- *operații de introducere a datelor de la tastatură conform specificațiilor propuse;*
- *operații de extragere a datelor la monitor conform specificațiilor propuse;*
- *operații de afișare a datelor cu format.*

Un operator este un simbol care îi spune compilatorului să efectueze anumite manipulări matematice sau logice. Limbajul C++ este bogat în operatori încorporați și oferă următoarele tipuri de operatori:

- Operatori aritmetici;
- Operatori relaționali;
- Operatori logici;
- Operatori pe biți;
- Operatori de atribuire;
- Operatori micști.

Analizați următorul exemplu pentru a înțelege toți *operatorii aritmetici* disponibili în C++. Copiați următorul program și testați cum funcționează.

```
#include <iostream>
using namespace std;
main() {
   int a = 21, b = 10, c;
   c= a+b; cout<<"Linia 1 - Valoarea lui c este: "<<c<<endl;
   c= a-b; cout<<"Linia 2 - Valoarea lui c este: "<<c<<endl;
   c= a*b; cout<<"Linia 3 - Valoarea lui c este: "<<c<<endl;
   c= a/b; cout<<"Linia 4 - Valoarea lui c este: "<<c<<endl;
   c= a%b; cout<<"Linia 5 - Valoarea lui c este: "<<c<<endl;
   c= a++; cout<<"Linia 6 - Valoarea lui c este: "<<c<<endl;
   c= a--; cout<<"Linia 7 - Valoarea lui c este: "<<c<<endl;
}
```

Tabelul 1.6: Operatori aritmetici

Nr.	Operator	Descrierea
1	“+”	Adunarea a două valori de tip int, float sau double;
2	“- “	Diferența a două valori de tip int, float sau double;
3	“*”	Produsul a două valori de tip int, float sau double;
4	“/”	Câtul a două valori de tip int, float sau double;
5	“%”	Restul de la împărțirea a două valori de tip int, float sau double;
6	“++”	Incrementarea unei valori întregi cu o unitate;
7	“--”	Decrementarea unei valori întregi cu o unitate;

Analizați următorul exemplu pentru a înțelege toți *operatorii relaționali* disponibili în C++. Copiați următorul program C++ și executați-l în mediul de programare. Vom prezenta un exemplu cu operatorul condițional[1].

```
#include <iostream>
using namespace std;
main() {
    int a = 21, b = 10;
    (a==b)?cout<<"Linia 1 - a este egal cu b"<<endl:
    cout<<"Linia 1 - a nu este egal cu b"<<endl;
    (a<b)?cout<<"Linia 2 - a este mai mic decat b"<<endl:
    cout<<"Linia 2 - a nu este mai mic decat b"<<endl;
    (a>b)?cout<<"Linia 3 - a este mai mare decat b"<<endl:
    cout<<"Linia 3 - a nu este mai mare decat b"<<endl;
    // sa schimbam valorile variabilelor initiale
    a = 5; b = 20;
    (a<=b)?
    cout<<"Linia 4 - "<<a<<" <= "<<b<<" adevarat"<<endl:
    cout<<"Linia 4 - "<<a<<" <= "<<b<<" fals"<<endl;
    (a>=b)?
    cout<<"Linia 5 - "<<a<<" >= "<<b<<" adevarat"<<endl:
    cout<<"Linia 5 - "<<a<<" >= "<<b<<" fals"<<endl;
}
```

1 Operatorul condițional (ternar) îndeplinește aceeși funcție precum instrucțiunea IF.

Tabelul 1.7: Operatori relaționali

Nr.	Operator	Descrierea
1	“==”	Verifică dacă valorile a doi operanzi sunt egale sau nu, dacă da atunci condiția devine adevărată.
2	“!=“	Verifică dacă valorile a doi operanzi sunt egale sau nu, dacă valorile nu sunt egale atunci condiția devine adevărată.
3	“>”	Verifică dacă valoarea operandului din stânga este mai mare decât valoarea operandului din dreapta, dacă da atunci condiția devine adevărată.
4	“<”	Verifică dacă valoarea operandului din stânga este mai mică decât valoarea operandului din dreapta, dacă da atunci condiția devine adevărată.
5	“>=”	Verifică dacă valoarea operandului din stânga este mai mare sau egală cu valoarea operandului din dreapta, dacă da atunci condiția devine adevărată.
6	“<=”	Verifică dacă valoarea operandului din stânga este mai mică sau egală cu valoarea operandului din dreapta, dacă da atunci condiția devine adevărată.

Analizați următorul exemplu pentru a înțelege toți *operatorii logici* disponibili în limbajul C++. Copiați următorul program C++ și executați-l în mediul de programare.

```
#include <iostream>
using namespace std;
main() {
    int a = 5, b = 20;
    cout << "Linia 1 - a && b: "<<(a && b)<<endl;
    cout << "Linia 2 - a || b: "<<(a || b)<<endl;
    cout << "Linia 3 - !(a && b): "<<!(a && b)<<endl;
    cout << "Linia 4 - !(a || b): "<<!(a || b)<<endl;
}
```

După execuția secvenței de cod anterioare veți obține ca răspuns valori binare: 1 = adevărat și 0 = fals.

Tabelul 1.8: Operatori logici

Nr.	Operator	Descrierea
1	"&&"	Denumit operator logic ȘI. Dacă ambii operanzi sunt diferiti de zero, atunci condiția devine adevărată.
2	"||"	Denumit operator logic SAU. Dacă oricare dintre cei doi operanzi este diferit de zero, atunci condiția devine adevărată.
3	"!"	Denumit operator logic NU (negație). Utilizăm pentru a inversa starea logică a operandului său. Dacă o condiție este adevărată, atunci operatorul logic NU va face condiția falsă și invers la fel este valabil.

Analizați următorul exemplu pentru a înțelege toți *operatorii pe biți* disponibili în C++. Copiați următorul program C++ și executați-l în mediul de programare.

```
#include <iostream>
using namespace std;
main() {
   unsigned int a = 60;        // 60 = 0011 1100
   unsigned int b = 13;        // 13 = 0000 1101
   int c = 0; c = a & b;       // 12 = 0000 1100
   cout<<"Linia 1 - Valoarea lui c: "<<c<<endl;
   c = a | b;             // 61 = 0011 1101
   cout<<"Linia 2 - Valoarea lui c: "<<c<<endl;
   c = a ^ b;             // 49 = 0011 0001
   cout<<"Linia 3 - Valoarea lui c: "<<c<<endl;
   c = ~a;                // -61 = 1100 0011
   cout<<"Linia 4 - Valoarea lui c: "<<c<<endl;
   c = a << 2;            // 240 = 1111 0000
   cout<<"Linia 5 - Valoarea lui c: "<<c<<endl;
   c = a >> 2;            // 15 = 0000 1111
   cout<<"Linia 6 - Valoarea lui c: "<<c<<endl;
}
```

Tabelul 1.9: Operatori pe biți

Nr.	Operator	Descrierea
1	“&”	Operatorul binar ȘI copiază un bit în rezultat dacă acesta există în ambii operanzi.
2	“|“	Operatorul binar SAU copiază un bit dacă există în oricare operand.
3	“^”	Operatorul binar XOR copiază bitul dacă este setat într-un singur operand, dar nu în ambele.
4	“~”	Operatorul complementar este unar și are efectul de a „întoarce” biții.
5	“<<”	Operator de schimbare binar la stânga. Valoarea operanzilor din stânga este mutată la stânga cu numărul de biți specificat de operandul din dreapta.
6	“>>”	Operator binar de schimbare la dreapta. Valoarea operanzilor din stânga este mutată la dreapta cu numărul de biți specificat de operandul din dreapta.

Analizați următorul exemplu pentru a înțelege toți *operatorii de atribuire* disponibili în C++. Copiați următorul program C++ și executați-l în mediul de programare.

```
#include <iostream>
using namespace std;
main() {
    int a = 21, c;
    c =  a;  cout<<"Linia 1 - Valoarea lui c: "<<c<<endl;
    c +=  a; cout<<"Linia 2 - Valoarea lui c: "<<c<<endl;
    c -=  a; cout<<"Linia 3 - Valoarea lui c: "<<c<<endl;
    c *=  a; cout<<"Linia 4 - Valoarea lui c: "<<c<<endl;
    c /=  a; cout<<"Linia 5 - Valoarea lui c: "<<c<<endl;
    c  = 200;
    c %=  a;  cout<<"Linia 6 - Valoarea lui c: "<<c<<endl;
    c <<=  2; cout<<"Linia 7 - Valoarea lui c: "<<c<<endl;
    c >>=  2; cout<<"Linia 8 - Valoarea lui c: "<<c<<endl;
    c &=  2;  cout<<"Linia 9 - Valoarea lui c: "<<c<<endl;
```

```
    c ^=  2; cout<<"Linia 10 - Valoarea lui c: "<<c<<endl;
    c |=  2; cout<<"Linia 11 - Valoarea lui c: "<<c<<endl;
}
```

Tabelul 1.10: Operatori de atribuire

Nr.	Operator	Descrierea
1	“=”	Operator de atribuire simplă, atribuie valori de la operanzii din dreapta către operanzii din stânga.
2	“+=“	Adaugă operandul din dreapta operandului din stânga și atribuie rezultatul operandului din stânga.
3	“-=”	Scade operandul din dreapta din operandul din stânga și atribuie rezultatul operandul din stânga.
4	“*=”	Înmulțește operandul din dreapta cu operandul din stânga și atribuie rezultatul operandului din stânga.
5	“/=”	Împarte operandul din stânga cu operandul din dreapta și atribuie rezultatul operandului din stânga.
6	“%=”	Împarte operandul din stânga cu operandul din dreapta și restul obținut atribuie rezultatul operandului din stânga.
7	“<<=”	Schimb la stânga și alocarea operatorului.
8	“>>=”	Schimb la dreapta și alocarea operatorului.
9	“&=”	Operator de atribuire ȘI pe biți.
10	“^=”	Operator de atribuire XOR pe biți.
11	“|=”	Operator de atribuire SAU pe biți.

Analizați următorul exemplu pentru a înțelege toți *operatorii speciali* disponibili în C++. Copiați următorul program C++ și executați-l în mediul de programare.

```
#include <iostream>
using namespace std;
main() {
    double x=3.1415, n1 = 21.09399;
```

```
    float n2 = 10.20;
    int a, b = 10, i, j, var, *ptr, val;
    // aplicarea operatorului sizeof()
    cout<<"Dimensiunea lui "<<x<<" este: \t";
    cout<<sizeof(x)<<" biti\n";
    // aplicarea operatorului conditional
    a = (b<=10) ? 30 : 40;
    cout<<"Valoarea operatorului este: \t"<<a<<endl;
    // aplicarea operatorului secvential virgula
    j = 10;
    i = (j++, j+100, 999+j);
    cout<<"Valoarea secventei este: \t"<<i<<endl;
    // aplicarea operatorului de distributie
    cout<<"Valoarea lui (int)"<<n1<<" este: \t"<<(int)n1<<endl;
    cout<<"Valoarea lui (int)"<<n2<<" este: \t"<<(int)n2<<endl;
    // aplicarea operatorului de referentiere si dereferentiere
    var = 3000; ptr = &var; // ia adresa variabilei var
    val = *ptr; // ia valoarea disponibilă in variabila ptr
    cout<<"Valoarea variabilei var: \t"<<var<<endl;
    cout<<"Valoarea variabilei ptr: \t"<<ptr<<endl;
    cout<<"Valoarea variabilei val: \t"<<val<<endl;
}
```

Tabelul 1.11: Operatori speciali

Nr.	Operator	Descrierea
1	sizeof	Returnează dimensiunea unei variabile, de exemplu, sizeof(a), unde „a” este întreg, se va returna 4 octeți.
2	Condiție ? X : Y	Operatorul condiționat (?). Dacă „Condiția” este adevărată, atunci returnează valoarea lui instrucțiunii X, altfel returnează valoarea instrucțiunii Y.
3	, (virgula)	Operatorul virgulă determină executarea unei secvențe de operații. Valoarea expresiei este valoarea ultimei expresii a listei separate prin virgulă.
4	. (punct) și -> (săgeată)	Operatorii membri sunt utilizați pentru a face referire la membrii individuali ai claselor, structurilor și uniunilor.

Nr.	Operator	Descrierea
5	( )	Distribuție. Operatorii de casting convertesc un tip de date în altul. De exemplu, int (2.2000) ar returna 2.
6	&	Operatorul pointer & returnează adresa unei variabile. De exemplu, &a; va oferi adresa reală a variabilei „a”.
7	*	Operatorul pointer * este pointer către o variabilă. De exemplu, *var; va indica o variabilă var.

1.2.1 Precedența operatorilor

Prioritatea operatorului determină gruparea termenilor într-o expresie. Acest lucru afectează modul în care este evaluată o expresie. Unii operatori au prioritate mai mare decât alții.

De exemplu, operatorul de înmulțire are o prioritate mai mare decât operatorul de adunare. Aici, operatorii cu cea mai mare prioritate apar în partea de sus a tabelului, cei cu cea mai mică prioritate apar în partea de jos. În cadrul unei expresii, operatorii cu prioritate mai mare vor fi evaluați mai întâi.

Analizați următorul exemplu pentru a înțelege *prioritatea operatorilor* disponibili în C++. Copiați următorul program C++ și executați-l în mediul de programare.

```
#include <iostream>
using namespace std;
main() {
    int a = 20, b = 10, c = 15, d = 5, e;
    e = (a + b) * c / d;      // (30 * 15 ) / 5
    cout << "Valoarea exoresiei: (a+b)*c/d este \t"<<e<<endl;
    e = ((a + b) * c) / d;    // (30 * 15 ) / 5
    cout << "Valoarea exoresiei: ((a+b)*c)/d este \t"<<e<<endl;
    e = (a + b) * (c / d);    // (30) * (15/5)
    cout << "Valoarea exoresiei: (a+b)*(c/d) este \t"<<e<<endl;
    e = a + (b * c) / d;      //  20 + (150/5)
    cout << "Valoarea exoresiei: a+(b*c)/d este \t"<<e<<endl;
}
```

Tabelul 1.12: Prioritatea operatorilor

Categoria	Operator	Asociativitatea
Postfix	() [] -> . ++ - -	De la stânga la dreapta
Unar	+ - ! ~ ++ - - (type)* & sizeof	De la dreapta la stanga
Multiplicativ	* / %	De la stânga la dreapta
Aditiv	+ -	De la stânga la dreapta
Schimb	<< >>	De la stânga la dreapta
Relațional	< <= > >=	De la stânga la dreapta
Egalitate	== !=	De la stânga la dreapta
ȘI pe biți	&	De la stânga la dreapta
XOR pe biți	^	De la stânga la dreapta
SAU pe biți	|	De la stânga la dreapta
ȘI logic	&&	De la stânga la dreapta
SAU logic	||	De la stânga la dreapta
Condiţional	?:	De la dreapta la stanga
Atribuire	= += -= *= /= %=>>= <<= &= ^= |=	De la dreapta la stanga
Virgula	,	De la stânga la dreapta

1.2.2 Conversia explicită. Expresii

Operatorii de conversie de tip permit conversia unei valori dintr-un anumit tip de dată la un al tip. Sunt câteva modalități de a face aceasta în C++. Cea mai simplă, care a fost moștenită din limbajul C, este cea prin care expresia de convertit este precedată de noul tip cuprins între paranteze rotunde:

```
int i;
float f = 3.14;
i = (int) f;
```

Codul anterior convertește numărul real zecimal 3.14 la o valoare întreagă (rândul al 3-lea), restul se pierde. Aici, operatorul de conversie este (int). Altă modalitate de a face exact același lucru în C++ este folosirea notației funcționale: după cuvântul cheie corespunzător noului tip, urmează expresia de convertit cuprinsă între paranteze:

```
int i;
float f = 3.14;
i = int (f);
```

O expresie în C++ este o colecție de ordine de operatori și operanzi care specifică un calcul. O expresie poate conține zero sau mai mulți operatori și unul sau mai mulți operanzi, operanzii pot fi constante sau variabile. De asemenea o expresie poate conține și apeluri de funcție care retur-neză valori constante. Rezultatul obținut după exprimarea evaluării este atribuit variabilei prin utilizarea operatorului de atribuire.

Există diferite categorii de expresii în C++ în funcție de rezultatul obținut în urma evaluării unei expresii sau pe baza tipurilor de operand prezente într-o expresie.

Tabelul 1.13: Expresii

Nr.	Tipul și descrierea	Exemplu
1	*Expresii constante* – expresiile constante conțin numai valori constante într-o expresie.	`x=25+10;`
2	*Expresii integrale* – expresii integrale care au ca rezultat o valoare integrală după evaluarea unei expresii.	`x+int(12.0);`
3	*Expresii flotante* – expresii flotante care rezultă din valorile flotante după evaluarea unei expresii.	`x+float(9);`
4	*Expresii pointer* – expresie pointer care are ca rezultat o adresă a unei variabile după evaluarea unei expresii.	`y=&x;`
5	*Expresii relaționale* – Expresie relațională care rezultă dintr-o valoare bool, fie adevărată, fie falsă, după evaluarea unei expresii.	`a-b>=x-y;`

Nr.	Tipul și descrierea	Exemplu
6	*Expresii logice* – Expresie logică care are ca rezultat o valoare bool, fie adevărată, fie falsă, după evaluarea unei combinații de două sau mai multe expresii relaționale.	`a>20 || b==20`
7	*Expresii pe biți* – Expresiile pe biți efectuează operația la un nivel de biți într-o expresie.	`x&4.`

1.2.3 Operații de introducere și extragere a datelor

Fiecare începător în C++ a întâlnit obiecte de intrare / de ieșire (adică cin și cout) și ce funcționalități existau etc. Dar cel mai important lucru este că aceste două operațiuni nu fac parte din limbajul de bază C++. Aceste funcționalități cin și cout sunt definite în biblioteca standard C++ (se află în spațiul de nume std).

În timp ce învățăm limbajul, includem biblioteca iostream pentru a utiliza obiectele cin și cout pentru operațiunile de intrare / ieșire. C++ oferă multe astfel de biblioteci pentru diverse sarcini. În C++ ieșirea / intrare este realizată sub formă de secvență de octeți cunoscută sub numele de fluxuri.

Un flux este o secvență de octeți care poate fi accesat secvențial și poate produce sau consuma cantități nelimitate de date. În C++ întâlnim în mare parte două fluxuri: de intrare și de ieșire.

Fluxul de intrare deține datele de intrare de la utilizator, cum ar fi tastatura, fișierul, etc. și așteaptă în buffer până când programul este gata să-l execute. În mod similar, fluxul de ieșire reține datele de la dispozitivele de ieșire până când acestea sunt gata să accepte acele date, dispozitivele de ieșire includ monitorul, imprimanta, etc.

Important

- Buffer este un substituent temporar pentru multe limbaje de programare din memorie (ram/disc) pe care datele pot fi descărcate și apoi procesarea poate fi efectuată.
- Bufferul crește performanța computerului permițând operarea de citire/scriere în bucăți mai mari.

Ieșirea fără buffer este adresată imediat, în timp ce ieșirea cu buffer este stocată și transmisă consolei de ieșire ca o secvență de date întreagă. Bibliotecile standard C++ oferă un set extins de obiecte de intrare și ieșire.

C++ vine cu patru obiecte de flux standard predefinite în biblioteci, ceea ce este util pentru uzul nostru. Includerea <iostream> include automat și <ios>, <streambuf>, <istream>, <ostream> și <iosfwd>.

Fluxul de intrare standard (cin): Obiectul cin este o instanță a clasei istream. Obiectul cin este utilizat cu operatorul de extracție a fluxului (>>).

```
cin>>var1>>var2;
```

Fluxul de ieșire standard (cout): Obiectul cout este o instanță a clasei ostream. Obiectul cout este utilizat cu operatorul de inserare a fluxului (<<).

```
char str[] = "Hello World!";
cout <<str<<endl;
```

Fluxul de erori standard fără buffer (cerr): Obiectul cerr este o instanță a clasei ostream și este legat de dispozitivul de eroare standard, care afișează mesajul de eroare pe ecran. Obiectul cerr este un buffer, astfel încât fiecare flux inserat este afișat imediat pe dispozitivul de ieșire. Obiectul cerr este utilizat în combinație cu operatorul de inserare a fluxului (<<).

Fluxul de jurnal standard (clog): Obiectul clog este o instanță a clasei ostream și este legat de dispozitivul de eroare standard, care afișează un mesaj de eroare pe ecran, dar obiect clog este buffer-izat. Aceasta înseamnă că fiecare inserție clog va fi ținută într-un buffer până când buffer-ul este umplut sau până când bufffer-ul este eliberat. Obiectul clog este utilizat în combinație cu operatorul de inserare a fluxului (<<).

Analizați următorul exemplu pentru a înțelege funcționalitatea fluxurilor de date disponibili în C++. Copiați următorul program C++ și îl executați în mediul de programare.

```
#include <iostream>
using namespace std;
main() {
```

```
    float x; int b = 20;
    char persoana[20]="Andrian Dascal",a = 'A';
    // introducerea datelor de la tastatura
    cout<<"Introduceti un numar real: "; cin>>x;
    // afisarea tuturor variabilelor din program
    cout<<"Afisam datele: \t"<<a<<", "<<b<<", "<<x<<".";
    cout<<"\nBine ai revenit, " <<persoana<<".\n";
}
```

1.2.4 Precizia numerelor cu virgulă mobilă

Echivalentul zecimal al lui 1/3 este 0,33333... Un număr de lungime infinită ar necesita memorie infinită pentru stocare și avem de obicei 4 sau 8 octeți. Prin urmare, numerele în virgulă mobilă stochează doar un anumit număr de cifre semnificative, iar restul se pierd.

Precizia unui număr în virgulă mobilă definește câte cifre semnificative poate reprezenta fără pierdere de informații. La ieșirea numerelor în virgulă mobilă, *cout* are o precizie implicită de 6 cifre și trunchiază astfel orice număr.

Mai jos sunt câteva biblioteci și metode care sunt utilizate pentru a oferi precizie numerelor în virgulă mobilă, acestea sunt definite în fișierul antet <cmath>.

Tabelul 1.14: Funcții de precizie a numerelor reale

Nr.	Metoda	Descrierea
1	floor()	rotunjește valoarea dată la cel mai apropiat număr întreg care este mai mic decât valoarea dată;
2	ceil()	rotunjește valoarea dată la cel mai apropiat număr întreg, care este mai mare decât valoarea dată;
3	trunc()	rotunjește prin trunchiere eliminând cifrele după virgulă zecimală;
4	round()	rotunjirea numărului se face conform celui mai apropiat întreg, este definită în fișierele antet: <cmath> și <ctgmath>.

Nr.	Metoda	Descrierea
5	setprecision()	atunci când este utilizat împreună cu „fixed” oferă precizie numerelor în virgulă mobilă, evidențiind numărul de numere zecimale menționate în paranteze setprecision(), este definit în fișierul antet <iomanip>.

Există și alte metode pentru a oferi precizie numerelor în virgulă mobilă. Cele menționate mai sus sunt câteva dintre cele mai frecvent utilizate metode pentru a oferi precizie numerelor în virgulă mobilă în timpul codificării competitive.

Analizați următorul exemplu pentru a înțelege funcționalitatea fluxurilor de date disponibili în C++. Copiați următorul program C++ și îl executați în mediul de programare.

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
main() {
    double x = 5.56789;
    cout<<"Metoda floor(): \t"<<floor(x)<<endl;
    cout<<"Metoda ceil(): \t\t"<<ceil(x)<<endl;
    cout<<"Metoda trunc(): \t"<<trunc(x)<<endl;
    cout<<"Metoda round(): \t"<<round(x)<<endl;
    // setprecision(3)
    cout<<"Metoda setprecision(): \t";
    cout<<setprecision(3)<<x<<endl;
     // fixed<<setprecision(3)
    cout<<"Metoda setprecision(): \t"<<fixed;
    cout<<setprecision(3)<<x<<endl;
}
```

1.2 Modele de itemi pentru exersare

(1) Pentru fiecare problemă propusă la itemul (1) de la tema precedentă (pag. 17), elaborați un program în limbajul C++.

(2) Pentru fiecare problemă propusă la itemul (2) de la tema precedentă (pag. 17), elaborați un program în limbajul C++.

(3) De Morgan a enunțat două legi logice despre negarea unei propoziții compuse sau a unei expresii. Elaborați un program în C++ care va demonstra implementarea acestor legi.

(4) Elaborați un program în limbajul C++ care va verifica dacă un număr x este divizibil cu numărul k. Numerele x și k sunt întregi, introduse de la tastatură. Pentru rezolvarea problemei se va aplica operatorul condițional (ternar).

(5) Elaborați un program în limbajul C++ care va verifica dacă un număr x este par și dacă el aparține intervalului [a, b]. Numerele a și b sunt numere întregi introduse de la tastatură. Pentru rezolvarea problemei se va aplica operatorul condițional (ternar).

(6) Elaborați un program în limbajul C++ care va detemina soluțiile x_1 și x_2 ale polinomului de gradul II: $ax^2+bx+c=0$, unde elementele a, b și c sunt numere întregi introduse de la tastatură. Pentru rezolvarea problemei se va aplica operatorul condițional (ternar).

(7) Elaborați un program în limbajul C++ care va detemina dacă un număr natural poate fi un an bisect. Pentru rezolvarea problemei se va aplica operatorul condițional (ternar). Exemple de ani bisecți: 2016, 2020, etc.

1.3 Tipuri de instrucţiuni: vide, simple şi compuse

În această temă vom analiza următoarele unități de conținut:

- *evaluarea unei expresii și determinarea tipului rezultatului acesteia;*
- *testarea funcționalității unei aplicații de consolă;*
- *interpretarea rezultatelor unei aplicații de consolă;*
- *generarea numerelor conform specificațiilor propuse;*
- *colorarea textului și a fundalului textului rezultatelor de la consolă.*

Descrierea acţiunilor de prelucrare a datelor care au loc în corpul unei funcţii se realizează cu ajutorul instucţiunilor, care sunt executate în ordinea dată de fluxul de control al execuţiei, atât timp cât fluxul de control nu este ramificat de către o selecţie, de o ciclare sau de un salt.

O instrucţiune simplă este formată din cuvinte cheie, expresii și eventual una sau două instrucţiuni-corp. De exemplu instrucţiunea IF are o singură instrucţiune-corp, iar IF-ELSE are două.

```
// instructiunea IF
if (i%2!=0 && i>0 && i<10) cout<<i<<" "<<endl;
// instructiunea IF-ELSE
if (i%2==0) cout<<"Numarul "<<i<<" este par."<<endl;
else cout<<"Numarul "<<i<<" nu este par."<<endl;
```

Există un singur tip de instrucţiune compusă, ea începe cu simbolul ‘{’ urmat de un bloc de instrucţiuni (posibil vid) și se termină cu simbolul ‘}’. Cele două acolade sunt părţi componente ale instrucţiunii.

```
{
     instrucțiuni 1; // comentariu pentru instrucțiunea 1
     . . .
     instrucțiuni N; // comentariu pentru instrucțiunea N
}
```

O situaţie de excepţie o constitue instrucţiunea nulă, care este formată numai din simbolul punct şi virgulă, ‘;’.
Modele de probleme care se rezolvă prin aplicarea instrucțiunilor simple:

Nr.	Condiție
1	Elaborați un program în limbajul C++ care va rezolva ecuația algebrică de gradul I cu o necunoscută și având coeficienții reali.
2	Elaborați un program în limbajul C++ care va rezolva ecuația algebrică de gradul II cu două necunoscute și având coeficienții reali.

Modele de probleme care se rezolvă prin aplicarea instrucțiunilor compuse:

Nr.	Condiție
1	Elaborați un program în limbajul C++ care va implementa instrucțiunea ciclică FOR pentru a afișa toate numerele prime din intervalul [0, 1000] și va afișa numărul total de elemente ce satisfac condiția.
2	Elaborați un program în limbajul C++ care va implementa instrucțiunea ciclică WHILE pentru a afișa toate pătratele și cuburile perfecte din intervalul [0, 1000] și va afișa numărul total de elemente ce satisfac fiecare condiție separate prin spațiu.

1.3.1 Expresii. Evaluarea expresiilor. Tipul rezultatului unei expresii

În limbajul C/C++ expresiile sunt secvenţe de operatori şi operanzi utilizate în unul sau mai multe dintre următoarele scopuri:

(1) calculul unei r-valori (r-value), a unei valori rezultat, care poate fi un număr, o structură sau o clasă. Denumirea vine de la right-value, adică valoarea care s-ar atribui obiectului din stânga operatorului de atribuire dacă expresia supusă evaluării ar fi operandul din dreapta.

(2) determinarea unei l-valori (l-value), a unei valori de locaţie, care desemnează calea către un anumit obiect. Analog cu cele de mai sus, denumirea de left-value provine de la evaluarea expresiilor de atribuire: operandul din stânga este evaluat pentru a-i afla l-valoarea, adică lo-

caţia de memorie unde trebuie copiată r-valoarea operandului din dreapta.

(3) Generarea de „efecte secundare, colaterale” (side effects). Prin efect secundar înţelegem orice altă acţiune care are loc odată cu evaluarea expresiei, în afara celei de a-i calcula r sau/şi l-valoarea, de exemplu: modificarea valorilor unor variabile, schimbarea stării unor fluxuri de date, etc.

Expresiile pot fi definite recursiv în două moduri:

(a) o expresie este formată dintr-un operand sau dintr-un operator aplicat operanzilor săi;

(b) un operand este format dintr-un literal, un identificator de variabilă sau dintr-o expresie scrisă între paranteze rotunde.

De exemplu: expresia *a*(b+1)*, este formată din operatorul de multiplicare „*” aplicat operanzilor *a* şi *(b+1)*, primul operand este un identificator de variabilă, al doilea este expresia *b+1*. La rândul ei, această expresie este formată din operatorul de adunare „+” aplicat unui identificator şi unui literal.

Pentru fiecare operator limbajul are reguli stricte pentru determinarea tipului rezultatului în funcţie de tipul operanzilor, astfel că orice expresie are rezultatul de un tip bine precizat.

Tabelul 1.15: Expresii şi tipul rezultatului acestora

Nr	Expresie	Tipul rezultatului
1	`int a=2, b=5;` `cout<<sqrt(a+7)-pow(b/5,2);`	Răspuns: *2* Tipul rezultatului: *număr întreg*
2	`int a=2, b=9;` `cout<<pow(b*4,1.0/(a+1));`	Răspuns: *3.30193* Tipul rezultatului: *număr real*
3	`char a='a',b='z';` `cout<<int ((a+b)/3);`	Răspuns: *73* Tipul rezultatului: *număr întreg*
4	`int a=26, b=56;` `cout<<char (a+b);`	Răspuns: *R* Tipul rezultatului: *caracter*

1.3.2 Generare număr natural. Generarea elementelor dintr-un interval.

Funcția *rand()* este o funcție încorporată în C++ STL, care este definită în fișierul antet `<cstdlib>`. Această funcție este aplicată pentru a genera o serie de numere aleatoare. Numărul aleatoriu este generat utilizând un algoritm care oferă o serie de numere neînrudite ori de câte ori este apelată această funcție.

Funcția *rand()* este utilizată în C++ pentru a genera numere aleatoare din intervalul [0, RAND_MAX), unde RAND_MAX este o constantă a cărei valoare implicită poate varia între implementări, dar se acordă să fie cel puțin 32767.

Dacă sunt generate numere aleatorii cu *rand()* fără a apela mai întâi *srand()*, programul va crea aceeași secvență de numere de fiecare dată când rulează.

Sintaxă

```
int rand(void);
```

Să spunem că generăm 5 numere aleatoare cu ajutorul funcției *rand()* într-o buclă, atunci de fiecare dată când compilam și rulăm programul rezultatul nostru trebuie să fie aceeași secvență de numere. Analizați următoarea secvență de cod C++:

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(){
    for (int i=0; i<5; i++){
        cout <<rand()<<" ";
    }
}
```

Funcția *srand()* este o încorporată în C++ STL, care este un fișier antet definit în `<cstdlib>`. Funcția este folosită pentru a inițializa generatoare de numere aleatoare. Funcția *srand()* stabilește punctul de pornire pentru producerea unei serii de numere întregi pseudoaleatoare.

Dacă *srand()* nu este apelat, atunci funcția *rand()* este setată ca și cum *srand(1)* ar fi apelat la pornirea programului.

Sintaxă

```
int srand();
```

Practica standard este de a folosi rezultatul unui apel la *srand(time(0))* ca inițiere. Cu toate acestea, *time()* returnează o valoare *time_t* care variază de fiecare dată și, prin urmare, numărul pseudo-aleatoriu variază pentru fiecare apel de program.

Exemplul 1: Generarea a 10 numere oarecare din intervalul [10, 20].

```
#include<iostream>
#include<stdlib.h>
#include<time.h>
using namespace std;
int main(){
    int i,nr=10,a=10,b=20; srand(time(0));
    for(i=0; i<nr; i++){
        cout<<" "<<a+(rand()%(b-a)); // numere oarecare
    }
    cout<<endl;
}
```

Exemplul 2: Generarea a 10 numere pare din intervalul [10, 90].

```
#include<iostream>
#include<stdlib.h>
#include<time.h>
using namespace std;
int main(){
    int i,nr=10,a=10,b=90; srand(time(0));
    for(i=0; i<nr; i++){
        cout<<" "<<a+(rand()%(b-a)/2)*2; // numere pare
    }
    cout<<endl;
}
```

1.3.3 Culori. Culori pentru text şi fundal.

În limbajul C++, fundalul ecranului de ieșire este negru, iar culoarea textului este de culoare albă. Acum s-ar putea să ne întrebăm: cum am putea schimba culoarea textului consolei? Calamarii își pot schimba culoarea, așa că de ce nu poate și consola? Există un cod denumit ANSI, acesta ne permite să specificăm ce culoare ar trebui să fie textul nostru.

```
cout<<"\033[COLOUR_CODEm"<<"Acesta este un text colorat!";
```

Observați cum începe cu **\033[** și se termină cu **m**. Mai jos sunt specificate codurile de culoare, pentru colorarea textului și a fundalului textului.

Tabelul 1.16: Culori pentru text și fundal

Culoare	Text	Fundal
Negru	30	40
Roșu	31	41
Verde	32	42
Galben	33	43
Albastru	34	44
Magenta	35	45
Cyan	36	46
Alb	37	47

Culoare	Text	Fundal
Negru luminos	90	100
Roșu luminos	91	101
Verde luminos	92	102
Galben luminos	93	103
Albastru luminos	94	104
Magenta luminos	95	105
Cyan luminos	96	106
Alb luminos	97	107

1. Ce va afișa următoarea secvență de cod C++ ?

```
#include <iostream>
using namespace std;
main(){
    cout<<"\033[31m"<<"Textul este rosu!"<<endl;
    cout<<"\033[103m"<<"Textul este rosu!"<<endl;
    cout<<"\033[0m";
}
```

2. Ce va afișa următoarea secvență de cod C++ ?

```
#include <iostream>
using namespace std;
main(){
    cout<<"\033[103m"<<"\033[34m"<<" 200 "<<"\033[0m"<<" + ";
    cout<<"\033[102m"<<"\033[95m"<<" 500 "<<"\033[0m"<<" = ";
    cout<<"\033[101m"<<"\033[97m"<<" 700 "<<endl;
    cout<<"\033[0m";
}
```

O altă modalitate de a afișare color la consolă a unui mesaj text, a unei valori numerice sau a unui simbol, este prezentat mai jos:

```
#include <iostream>
#include <iomanip>   // include instrucțiunea setw()
#include <windows.h>
using namespace std;
main(){
    HANDLE c = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(c,0); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,1); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,2); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,3); cout<<setw(10)<<"Text\n";
    SetConsoleTextAttribute(c,4); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,5); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,6); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,7); cout<<setw(10)<<"Text\n";
    SetConsoleTextAttribute(c,8); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,9); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,10); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,11); cout<<setw(10)<<"Text\n";
    SetConsoleTextAttribute(c,12); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,13); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,14); cout<<setw(10)<<"Text"<<" ";
    SetConsoleTextAttribute(c,15); cout<<setw(10)<<"Text\n";
    // revenim la culoarea implicita
    SetConsoleTextAttribute(c,7);
}
```

1.3 Modele de itemi pentru exersare

(1) Care va fi rezultatul execuției următoarelor programe C++?

```
#include <iostream>
#include <cmath>
using namespace std;
int main(){
    int x;
    x=sqrt(25-pow(2,3));
    cout<<"S="<<x;
}
```

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
int main(){
   float x;
   x=sqrt(abs(25-pow(5,3)));
   cout<<"S="<<setw(5)<<x;
}
```

(2) Elaborați câte un program pentru fiecare caz, în limbajul C++, care va calcula rezultatul următoarelor expresii matematice:

a) $\frac{6+e^7}{15\cdot 5^e}-\frac{|e-5|}{\sqrt[3]{81}}$; c) $\frac{3}{\ln(e)}-\frac{\log(100)}{5!}$; e) $\frac{\sin^2(\pi)+\sqrt{7}}{\cos^2(\pi)-\pi^e}$;

b) $\frac{-1-e^7}{5^e\cdot\sin(\frac{3\cdot\pi}{2})}$; d) $\frac{\log(10)-\ln(e^3)}{2\cdot|1-7!|}$; f) $\frac{ctg^2(\pi)+e^{\frac{-1}{3}}}{tg^2(\pi)-1^e}$.

Include fișierul antet `<cmath>` pentru aplicarea funcțiilor matematice. Afișați rezultatul expresiilor pe 15 poziții și cu 7 cifre după virgulă.

(3) Elaborați un program în limbajul C++ care va calcula valoarea lui *P(c)*, dacă $P(x)=x^3-9x+14$ și $c=\sqrt[3]{8+\sqrt{37}}+\sqrt[3]{8-\sqrt{37}}$.

(4) Elaborați un program în limbajul C++ care va determina coeficienții polinomului *P(x)* de gradul II , dacă *P(1)=4*, *P(-1)=7* și *P(3)=24*.

(5) Elaborați un program în limbajul C++ care va determina rădăcinile polinomului $P(x)=x^3-15x^2+74x-120$ de gradul III, dacă se știe că una din rădăcini este media aritmetică a celorlalte două rădăcini.

Capitolul 2

STRUCTURI DE CONTROL

Introducere

În acest compartiment vom analiza importanța și implementarea instrucțiunilor de decizie și a instrucțiunilor ciclice. De multe ori în codurile noastre o să avem nevoie de o modalitate de a executa anumite linii de cod doar dacă se îndeplinește o anumită condiție. Instrucțiunile de decizie, numite și structuri alternative au fost create cu sopul de a rezolva unele solicitări condiționate. Iată câteva exemple reale:

(1) Dacă un elev are media cel puțin 5, atunci trece clasa (altfel, dacă are media mai mică decât 5, atunci nu trece clasa);
(2) Dacă o persoană cumpără un produs de 10 lei, atunci din contul lui bancar vor fi scoși 10 lei.
(3) Dacă rezolvi o problemă de 100 de puncte, primești steluțe drept recompensă.

Există unele probleme ce pot fi soluționate cu structuri de date mai complexe precum și folosirea unor noi instrucțiuni, care să permită repetarea de un număr oarecare de ori a unor părți din algoritm.

Să luăm ca exemplu algoritmul de calcul al sumei a două numere introduse de la tastatură. Generalizarea problemei pentru *n* numere va modifica substanțial algoritmul nostru față de cazul când erau doar două numere. Nu vom putea folosi câte o variabilă pentru fiecare număr introdus, deoarece nu cunoaștem exact câte numere avem. Chiar dacă s-ar cunoaște că avem 2500 de numere, ar fi dificil să utilizăm 2500 de variabile distincte.

Instrucţiunile ciclice se împart în două categori:

(a) care realizează o structură repetitivă condiţionată anterior: *while*, *for*.
(b) care realizează o structură repetitivă condiţionată posterior: *do-while*.

2.1 Instrucţiuni de decizie simplă şi multiplă

În această temă vom analiza următoarele unități de conținut:

- *identificarea părților componente ale unei instrucțiuni de decizie simplă și multiplă;*
- *translarea algorimilor de decizie unică, dublă și triplă;*
- *aplicarea operatorului ternar (condițional) conform specificațiilor propuse,*
- *implementarea algoritmilor în limbajul C++.*

Limbajul C++ acceptă condițiile logice obișnuite din matematică. Aceste condiții pot fi aplicate pentru a efectua diferite acțiuni pentru diferite decizii. În limbajul C++ deosebim următoarele instrucțiuni condiționale:

(1) Utilizați IF pentru a specifica un bloc de cod care urmează să fie executat, dacă o condiție specificată este adevărată;
(2) Utilizați ELSE pentru a specifica un bloc de cod care urmează să fie executat, dacă aceeași condiție este falsă;
(3) Utilizați ELSE-IF pentru a specifica o nouă condiție de testat, dacă prima condiție este falsă;
(4) Utilizați SWITCH pentru a specifica multe blocuri alternative de cod care urmează să fie executate.

2.1.1 Instrucţiunea de decizie IF

Utilizați instrucțiunea *if* pentru a specifica un bloc de cod C++ care urmează să fie executat dacă o condiție este adevărată. Dacă instrucțiunea nu este scrisă cu litere mici (if), dar cu litere mari (IF), atunci vagenera o eroare.

Sintaxă

```
if (condiție) {
  ...
}
```

În exemplul următor, testăm două valori pentru a afla dacă 20 este mai mare decât 18. Dacă condiția este adevărată, afișăm un text la consolă:

```
if (20>18) {
    cout<<"Numarul 20 este mai mare decat 18.";
}
```

2.1.2 Instrucţiunea de decizie IF-ELSE

Utilizați instrucțiunea *else* pentru a specifica un bloc de cod care urmează să fie executat dacă condiția este falsă.

Sintaxă

```
if (condiție) {
  . . .
} else {
  . . .
}
```

În exemplul următor, testăm două valori pentru a determina persoanele care au bursă în dependență de media generală. Dacă condiția este falsă, afișăm un text la consolă:

```
float media = 6.78;
if (media <=10 && media>8.50){
   cout<<"Felicitari, bursa ta este: "<<media*100;
} else {
   cout<<"Nu aveti media corespunzatoare pentru bursa!";
}
```

În exemplul anterior, numărul `6.78` este mai mic decât toate orice număr din intervalul [8.50, 10], deci condiția este falsă. Din această cauză, trecem la starea *else* și imprimăm pe ecran „`Nu aveti media corespunzatoare pentru bursa!`”. Dacă media era să fie conform condiției, atunci programul ar tipări „`Felicitari, bursa ta este: 678`”.

2.1.3 Instrucţiunea de decizie ELSE-IF

Utilizați instrucțiunea *else-if* pentru a specifica o nouă condiție dacă prima condiție este falsă.

Sintaxă

```
if (condiție 1) {
  . . .
} else if (condiție 2) {
  . . .
} else {
  . . .
}
```

În exemplul următor, testăm o valoare pentru a determina partea zilei în dependență de oră: „Buna dimineata!”, „Buna ziua!” și „Buna seara!”

```
int timp = 22;
if (timp < 10) {
    cout<<"Buna dimineata!";
} else if (timp < 20) {
    cout <<"Buna ziua!";
} else {
    cout <<"Buna seara!";
}
```

Ora 22 reprezintă timpul de seară. Deoarece nu se respectă ambele condiții ale instrucțiunii, din această cauză trecem la starea *else* și imprimăm pe ecran mesajul: „`Buna seara!`”.

2.1.4 Instrucţiunea de decizie prescurtată

Există, de asemenea, o prescurtare *if-else*, care este cunoscută sub numele de operator ternar, deoarece constă din trei operanzi. De asemenea i se mai spune și operator condițional, deoarece poate fi folosit pentru a înlocui mai multe linii de cod cu o singură linie. Este adesea folosit pentru a înlocui declarațiile simple *if-else*.

Sintaxă

```
Variabilă = (Condiție) ? ExpresieAdevarată : ExpresieFalsă;
```

În exemplul următor, vom verifica dacă un număr este pătrat perfect cu ajutorul ambelor instrucțiuni:

Instrucțiunea if-else	Operatorul ternar
`int a = 25;` `if ((int)sqrt(a)==sqrt(a)){` `  cout<<"Adevarat!";` `} else cout<<"Fals!";` `// raspunsul va fi Adevarat`	`int a = 25;` `((int)sqrt(a)==sqrt(a))?` `cout<<"Adevarat!":cout<<"Fals!";` `// raspunsul va fi Adevarat`

2.1.5 Instrucţiunea de decizie multiplă

Utilizați instrucțiunea *switch* pentru a selecta unul dintre multele blocuri de cod care urmează să fie executate.

Sintaxă

```
switch(expresie) {
      case x:  instrucțiune bloc 1;
               break;
      case y:  instrucțiune bloc 2;
               break;
      . . .
      default: instrucțiune implicită generală;
}
```

Instrucțiunea *switch* funcționează astfel:

1. Expresia de comutare este evaluată o dată;
2. Valoarea rezultat a expresiei este comparată cu valoarea fiecărui caz disponibil;
3. Dacă există o potrivire, blocul de cod asociat este executat;
4. Cuvintele cheie *break* și *default* sunt opționale, totuși ele sunt necesare de a fi prezente în secvența de cod.

În exemplul următor, vom folosi numărul unei zile lucrătoare pentru a afișa la ecran denumirea acelei zile din săptămână:

```
int ziua = 4;
switch (ziua) {
       case 1: cout << "Luni"; break;
       case 2: cout << "Marti"; break;
       case 3: cout << "Miercuri"; break;
       case 4: cout << "Joi"; break;
       case 5: cout << "Vineri"; break;
       case 6: cout << "Sambata"; break;
       case 7: cout << "Duminica"; break;
}
// raspunsul va fi Joi
```

Când se găsește o potrivire și se execută instrucțiunea sau blocul de instrucțiuni, atunci *break* permite ieșirea din blocul de comutare. Un *break* poate economisi mult timp de execuție, deoarece „ignoră" execuția restului codul din blocul de comutare.

Cuvântul cheie *default* specifică un cod de rulat dacă nu există potrivire cu nici un rezultat sau condiție propusă.

În exemplul următor, vom verifica dacă un număr este pătrat perfect cu ajutorul instrucțiunilor *if-else* și *switch* :

Instrucțiunea if-else	Instrucțiunea switch
`int a = 25;` `if ((int)sqrt(a)==sqrt(a)){` `  cout<<"Adevarat!";` `} else cout<<"Fals!";` `// raspunsul va fi Adevarat`	`switch((int)sqrt(a)==sqrt(a)) {` `  case 1: cout<<"Adevarat!";` `  break;` `  case 0: cout<<"Fals!";` `  break;` `} // raspunsul va fi de la case 1`

Așadar, am reușit să analizăm aceste 3 instrucțiuni de decizie (*if-else* , *operatorul ternar* și *switch*) implementându-le la soluționarea aceleiași probleme. Rămâne doar să analizați diferențele de cod ale fiecăreia și să nu le confundați.

2.1 Modele de itemi pentru exersare

(1) Pentru fiecare din itemii (4) - (7) de la tema 1.2 (pag. 32), elaborați un program în limbajul C++ care va rezolva această problemă cu implementarea instrucțiunii de decizie *if* .

(2) Pentru fiecare din itemii (4) și (5) de la tema 1.2 (pag. 32), elaborați un program în limbajul C++ care va rezolva această problemă cu implementarea instrucțiunii de decizie *switch*.

(3) Pentru fiecare din următoarele probleme, elaborați un program în limbajul C++ care va rezolva această problemă cu implementarea instrucțiunii de decizie *if* :
   a) Determinați dacă un punct *A(x,y)*, unde *x* și *y* sunt numere reale, aparține interiorului unui cerc. Cercul are raza *r=5 cm* și este situat cu centrul sistemului cartezian de coordonate.
   b) Determinați ultima cifră a numărului 7^n, dacă se știe că numărul *n* este natural nenul. Cum procedați?
   c) Determinați dacă două numere întregi *a* și *b* sunt: impare și consecutive; pare și consecutive; prime și consecutive. Cum procedați?

(4) Pentru fiecare din următoarele probleme, elaborați un program în limbajul C++ care va rezolva această problemă cu implementarea instrucțiunii de decizie *switch* :
   a) Determinați dacă trei numere naturale *a*, *b* și *c* sunt în ordine ascendentă sau descendentă.
   b) Determinați dacă patru numere naturale *a*, *b, c* și *d* pot forma un paralelogram sau un trapez isoscel. Cum procedați?
   c) Determinați dacă trei numere naturale *a*, *b* și *c* satisfac condițiilor următoare: pot forma un triunghi; satisfac condiției teoremei lui Pitagora. Cum procedați?

2.2 Instrucţiuni ciclice. Particularităţi.

În această temă vom analiza următoarele unități de conținut:

- *identificarea părților componente și a particularităților specifice unei instrucțiuni ciclice;*
- *aplicații ale instrucțiunilor: break, continue și goto conform specificațiilor propuse;*
- *implementarea algoritmilor ciclici conform specificațiilor propuse în limbajul C++.*

Buclele sau instrucțiunile ciclice pot executa un bloc de cod atâta timp cât este atinsă o condiție specificată. Buclele sunt utile, deoarece economisesc timp, reduc erorile și fac codul mai ușor de citit.

2.2.1 Instrucţiunea ciclică FOR

Bucla *for* este folosită de obicei pentru a itera printre numerele dintr-un interval în ordine crescătoare sau descrescătoare. Însă, limbajul C/C++ a inovat cu adevărat instrucțiunea respectivă, făcând-o mult mai flexibilă și mai practică.

Semantică for: se execută *condiția1*, aceasta este o instrucțiune de inițializare, ce se execută o singură dată. De obicei, inițializează o variabilă contor (iterator) cu o anumită valoare, de la care se începe iterația. Apoi se testează *condiția2*. Dacă aceasta este adevărată, se continuă cu pasul următor, în caz contrar, se iese imediat din *for* și programul continuă cu instrucțiunile următoare. După aceasta se execută blocul de instrucțiuni și după aceasta se execută *instructiune*, care de obicei incrementează sau decrementează iteratorul specificat la *condiția1* și *condiția2*.

Sintaxă

```
for (condiția1; condiția2; instrucțiune) {
      . . . // blocul de cod ce urmează a fi executat
}
```

Este bine de știut că oricare din cei trei „parametri” ai structurii *for* pot lipsi, însă cele două caractere „;” sunt obligatorii, pentru a-i separa. Dacă *condiția2* lipsește, se consideră că valoarea sa de adevăr este true întotdeauna.

În exemplul de mai jos, vom tipări numerele pare de la 0 la 10 și vom contoriza numărul acestora:

```
int pare=0;    // initializam variabila pare pentru contorizare
for (int i=0; i<=10; i++) {
    if (i%2==0){           // verificam daca numarul i este par
        cout<<i<<" "; pare++; // afisam numarul I si contorizam
    }
}
cout<<"\nNumarul total de numere pare este "<<pare<<endl;
```

2.2.2 Instrucțiunea ciclică WHILE

Bucla *while* trece printr-un bloc de cod atâta timp cât o condiție specificată este adevărată.

Semantică *while*: mai intâi, se testează dacă condiția este adevărată. Dacă da, se execută instrucțiunea, și procesul o ia de la capăt. Dacă nu, se iese din *while*.

Sintaxă

```
while (condiție) {
      . . . // blocul de cod ce urmează a fi executat
}
```

În exemplul de mai jos, codul din buclă va rula, iar și iar, atâta timp cât o variabilă (i) este mai mică de 5:

```
int i = 0;
while (i < 5) {
     cout << i << "\n"; // afisarea variabilei i din rand nou
     i++;               // incrementarea variabilei i
}
```

Nu uitați să incrementați variabila folosită în condiție, altfel bucla nu se va termina niciodată!

2.2.3 Instrucţiunea ciclică DO ... WHILE

Bucla *do ... while* va executa blocul de cod o dată, înainte de a verifica dacă condiția este adevărată, apoi va repeta bucla atâta timp cât condiția este adevărată.

Semantică *do ... while* : se începe prin execuția instrucțiunii sau blocului de instrucțiuni, apoi se testează condiția. În cazul în care condiția este îndeplinită, procesul se reia. Dacă nu, atunci se iese din *do ... while*. Se poate observa că setul de instrucțiuni este executat cel puțin o dată, datorită poziționării testului de condiție.

Sintaxă

```
 do {
      . . . // blocul de cod ce urmează a fi executat
}
while (condiție);
```

Exemplul de mai jos folosește o buclă *do ... while*. Bucla va fi întotdeauna executată cel puțin o dată, chiar dacă condiția este falsă, deoarece blocul de cod este executat înainte ca condiția să fie testată:

```
int i = 0;
do {
    cout << i << "\n"; // afisarea variabilei i din rand nou
    i++;               // incrementarea variabilei i
}
while (i < 5);
```

2.2.4 Instrucţiunile break şi continue implementare cu instrucţiuni ciclice

Ați întâlnit deja instrucțiunea *break* folosită anterior pentru a „sări” dintr-o declarație *switch*. Declarația *break* poate fi folosită și pentru a ieși dintr-o buclă.

Instrucțiunea *continue* întrerupe o iterație (în buclă), dacă apare o condiție specificată și continuă cu următoarea iterație din buclă.

Următorul exemplu ne permite ieșirea din buclă atunci când se realizează condiția din instrucțiunea de decizie *if* : i==4.

```
for (int i = 0; i < 10; i++) {
    if (i == 4) break; // conditia de ieșire din bucla for
    cout<<i<<" ";      // va afisa datele într-un rând
}
```

Următorul exemplu ne permite omiterea (saltul) în buclă atunci când se realizează condiția din instrucțiunea de decizie *if* : i==4.

```
for (int i = 0; i < 10; i++) {
    if (i == 4) continue; // conditia de salt in bucla for
    cout<<i<<" ";         // va afisa datele într-un rând
}
```

Scrieți secvența de cod în limbajul C++ și apoi verificați rezultatele în mediul de programare. Analizați rezultatele execuției secvențelor de cod prezentate anterior.

Execuția cu *break*	**Execuția cu *continue***
0 1 2 3	0 1 2 3 5 6 7 8 9 10

Pentru bucla *while* și bucla *do ... while* implementați instrucțiunile *break* și *continue* asemănător exemplelor anterior prezentate. Explicați cum funcționează aceste instrucțiuni.

2.2.5 Instrucţiunile goto

Instrucțiunea *goto* este o declarație de salt, denumită și declarație de salt necondiționat. Declarația *goto* poate fi folosită pentru a sări de oriunde în orice loc într-o funcție.

Instrucțiunea *goto* transferă controlul în locația specificată de etichetă, ea trebuie să aibă aceeași funcție cu eticheta la care se referă, poate apărea înainte sau după etichetă.

Sintaxă

```
goto identificator;
```

Instrucțiunea a cărei etichetă este dată în parametrul *identificator* trebuie să fie în funcția curentă. Toate numele date în parametrul *identificator* sunt membri ai spațiului de nume intern și, prin urmare, nu se suprapun cu alți identificatori. Eticheta declarației are sens numai pentru o declarație *goto,* în caz contrar, etichetele declarațiilor sunt ignorate. Etichetele nu pot fi redeclarate.

În exemplul de mai jos, codul implemenează instrucțiunea *goto* pentru determinarea mediei aritmetice a numerelor întregi din intervalul [0, 100]:

```
#include <iostream>
using namespace std;
int main() {
    float num, media, s = 0.0;
    int i, n=100;
    for(i=1; i<=n; ++i){
        if (i<0.0) {
            // Controlul programului se mută la salt
            goto salt;
        }
        s+=i;
    }
    salt:
        media=s/(i-1);
        cout<<"\nMedia = "<<media;
        return 0;
}
```

Important

- Puteți scrie orice program C++ fără a utiliza instrucțiunea *goto,* deoarece poate face programul complex și confuz. Declarația *goto* în programarea modernă este considerată o practică proastă de programare.
- Instrucțiunea *goto* poate fi înlocuită în majoritatea programelor C++ cu instrucțiuni *break* și *continue*.

2.2 Modele de itemi pentru exersare

(1) Pentru intervalul de numere întregi [*a*, *b*], aplicați instrucțiunea ciclică for și elaborați câte un program C++ care va afișa la consolă:
a) toate numerele pare și câte sunt acestea;
b) toate numerele impare și câte sunt acestea;
c) toate numerele prime și câte sunt acestea;
d) toate pătratele perfecte și câte sunt acestea;
e) toate cuburile perfecte și câte sunt acestea;
f) toate elementele din seria lui Fibonacci și câte sunt acestea.

(2) Aplicați instrucțiunea ciclică *while* și elaborați câte un program C++ care va afișa la consolă:
a) numărul de cifre ale unui întreg pozitiv;
b) suma cifrelor pare ale unui număr întreg pozitiv;
c) suma cifrelor impare ale unui număr întreg pozitiv;
d) suma cifrelor prime ale unui număr întreg pozitiv;
e) suma cifrelor din seria lui Fibonacci ale unui număr întreg pozitiv.

(3) Aplicați instrucțiunea ciclică *do ... while* și elaborați câte un program C++ care va afișa la consolă rezultatele de la itemul precedent. Ce modificări trebuie efectuate pentru a obține rezultatul propus?

(4) Elaborați un program C++ care va permite citirea de la tastatură a unui număr format din cel mult 9 cifre și va fișa la consolă de câte ori se repetă în scrierea lui cifra unităților și de câte ori se repetă cifra zecilor.

(5) Elaborați un program C++ care va permite calcularea următoarei valori matematice: $\sqrt{2+\sqrt{4+...+\sqrt{98+\sqrt{100}}}}$. Cum veți proceda? Care dintre cele trei instrucțiuni ciclice cel mai bine se potrivește și de ce?

Capitolul 3

TIPURI DE DATE STRUCTURATE

Introducere

Tipurile de date structurate, spre deosebire de cele simple, sunt combinații de alte tipuri definite prin descrierea tipurilor componentelor și prin indicarea unor metode de structurare.

Structura datelor este o stocare care este utilizată pentru păstra și organiza datele. Este o modalitate de a aranja datele într-un sistem de calcul, astfel încât să poată fi accesate și actualizate eficient.

În funcție de cerințe și proiect, este important să alegem structura de date potrivită pentru problema care ne propunem s-o rezolvăm. De exemplu, dacă dorim să stocăm date secvențial în memorie, atunci putem alege structura de date *array (tablou unidimensional sau bidimensional)*.

Practic, structurile de date sunt împărțite în două categorii:

(1) structuri de date liniare;
(2) structuri de date neliniară.

În structurile de date liniare (*array*, *stiva*, *coada*, *listele liniare simple și duble*, *listele circulare simple și duble*) elementele sunt aranjate în succesiune unul după altul. Deoarece elementele sunt aranjate într-o ordine specială, ele sunt ușor de implementat. Cu toate acestea, atunci când complexitatea programului crește, structurile de date liniare ar putea să nu fie cea mai bună alegere din cauza complexității operaționale.

Spre deosebire de structurile de date liniare, elementele din structurile de date neliniare (*arbori*, *grafuri*) nu sunt în nicio secvență. În schimb, ele sunt aranjate într-o manieră ierarhică în care un element va fi conectat la unul sau mai multe elemente. Structurile de date neliniare sunt împărțite în continuare în structuri de date bazate pe grafuri și arbori.

3.1 Tablouri unidimensionale şi bidimensionale

În această temă vom analiza următoarele unități de conținut:

- *declararea, inițializarea, citirea și afișarea tablourilor unidimensionale și bidimensionale;*
- *prelucrarea datelor tablourilor unidimensionale și bidimensionale conform specificațiilor propuse;*
- *posibilitățile clasei array și vector în C++ STL;*
- *implementarea tablourilor unidimensionale și bidimensionale în limbajul C++.*

Un *tablou unidimensional* poate fi numit și *vector* sau *array 1D,* iar un *tablou bidimensional* poate fi numit și *matrice* sau *array 2D.* Vectorii sunt folosiți pentru a stoca mai multe valori într-o singură variabilă, în loc să declare variabile separate pentru fiecare valoare.

Elementele vectorului sunt accesate folosind un index întreg. Indicele vectorului începe cu 0 și merge până la dimensiunea vectorului minus 1. Numele vectorului este, de asemenea, un pointer către primul element al vectorului. Nu există nicio verificare a indexului în afara limitelor în C++. În limbajul C nu este o eroare a compilatorului pentru inițializarea unui vector cu mai multe elemente decât dimensiunea specificată.

Avantaje	Dezavantaje
1. Acces aleatoriu al elementelor folosind indexul vectorului. 2. Utilizarea a mai puține linii de cod, deoarece creează un singur vector de elemente multiple. 3. Acces facil la toate elementele. 4. Traversarea prin vector devine ușoară folosind o singură buclă. 5. Sortarea devine ușoară, deoarece poate fi realizată prin scrierea a mai puține linii de cod.	1. Permite introducerea unui număr fix de elemente care este decis în momentul declarării. 2. Inserarea și ștergerea elementelor pot fi costisitoare, deoarece elementele trebuie să fie gestionate în conformitate cu noua alocare de memorie.

3.1.1 Tablourile unidimensionale (array 1D sau vector)

Pentru a declara un *vector,* definim tipul variabilei, specificăm numele *vector-ului* urmat de paranteze drepte și specificăm numărul de elemente pe care ar trebui să le stocheze:

```
int note[4];
```

Anterior am declarat o variabilă de tip *int* care deține un *vector* de patru elemente. Pentru a insera valori în el, putem folosi un *vector* implicit, plasăm valorile într-o listă separată prin virgulă, în interiorul acoladei.

Accesăm un element din *vector* referindu-ne la numărul de index din parantezele drepte, []. Pentru a modifica valoarea unui anumit element din *vector*, consultăm numărul de index (poziție).

```
int note[]= {10,8,5};     // vector de date implicit
cout<<note[0];            // va afisa nota 10 (primul element)
note[1]=7; cout<<note[1]; // va afisa nota 7 in loc de 8
```

Pentru a obține dimensiunea unui *vector*, putem utiliza operatorul *sizeof()*:

```
int note[]= {10,8,5,9};
cout<<sizeof(note); // va afisa 16
```

De ce rezultatul a arătat 16 în loc de 4, când *vector*-ul conține 4 elemente? Se datorează faptului că operatorul sizeof() returnează dimensiunea unui tip în octeți. Așadar, un tip de date *int* este de obicei de 4 octeți, deci din exemplul de mai sus, 4*4 (4 octeți * 4 elemente) = 16 octeți.

Pentru a afla câte elemente are un *vector*, trebuie să împărțim dimensiunea *vector-ului* la dimensiunea tipului de date pe care îl conține:

```
int note[]= {10,8,5,9};
cout<<sizeof(note)/sizeof(int); // va afisa 4
```

Să elaborăm un program care va permite introducerea datelor unui vector de numere întregi de la tastatură și afișarea acestor date la consolă.

```
#include <iostream>
using namespace std;
int main() {
    int n,i,a[10];
    cout<<"Introduceti dimeniunea vectorului: "; cin>>n;
    system("CLS"); // curatarea ecranului de consola
    cout<<"Introduceti elementele vectorului: "<<endl;
    for (i=0;i<n;i++){ // pacurgerea indicilor elementelor
        cout<<"a["<<i<<"]= "; cin>>a[i]; // citirea
    }
    system("CLS"); // curatarea ecranului de consola
    cout<<"Afisam elementele vectorului: "<<endl;
    for (i=0;i<n;i++){ // pacurgerea indicilor elementelor
        cout<<"a["<<i<<"]= "<<a[i]<<endl; // afisarea
    }
}
```

3.1.2 Tablourile bidimensionale (array 2D sau matrice)

Pentru a declara o *matrice,* definim tipul variabilei, specificăm numele *maticei* urmat de paranteze drepte și specificăm numărul de elemente pe care ar trebui să le stocheze:

```
int note[2][4]; // 2 linii si 4 coloane are matricea
```

Ca și în cazul *vector-ilor*, putem insera valori implicite, separate prin virgulă în interiorul acoladelor.

```
int note[2][5]= {
    {5,6,7,8,9}, {10,9,8,7,6} // cele doua linii ale matricei
};
```

Fiecare set de paranteze pătrate dintr-o declarație a *matricei* adaugă o altă dimensiune *matricei*. Se spune că o *matrice* ca cea prezentată mai sus are două dimensiuni. *Matricele* pot avea orice număr de dimensiuni, cu cât o *matrice* are mai multe dimensiuni, cu atât codul devine mai complex. Următoarea *matrice* are trei dimensiuni, se mai numește *array 3D* :

```
int note[2][2][2]= {
    {         {5,6}, {10,9}
    },
    {         {8,7}, {5,10}
    }
};
```

Pentru a accesa un element dintr-o *matrice*, specificăm un număr de index în fiecare dintre dimensiunile *matricei*. Instrucțiunea [0][2], accesează valoarea elementului din primul rând și a treia coloană a matricei.

```
int note[2][5]= {
    {5,6,7,8,9}, {10,9,8,7,6} // cele doua linii ale matricei
};
cout<<note[0][2]; // se va afisa nota 7 din primul rand
```

Pentru a modifica valoarea unui element, consulăm numărul de index al elementului în fiecare dintre dimensiuni:

```
int note[2][5]= {
    {5,6,7,8,9}, {10,9,8,7,6} // cele doua linii ale matricei
};
note[0][0]=10;
cout<<note[0][0]; // se va afisa nota 10 in loc de 5
```

Elaborăm un program în limbajul C++ care va permite introducerea datelor unei *matrici* de numere întregi de la tastatură și afișarea acestor date la consolă.

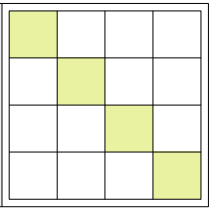
```
#include <iostream>
using namespace std;
int main() {
    int i,j,m[5][5],x,y;
    // citim numarul de linii si numarul de coloane
    cout<<"Introduceti numarul de linii: "; cin>>x;
    cout<<"Introduceti numarul de coloane: "; cin>>y;
    system("CLS");
    cout<<"Introduceti elementele matricei: "<<endl;
```

```
    for (i=0;i<x;i++){     // parcurgerea pe linie
        for (j=0;j<y;j++){ // parcurgerea pe coloana
            cout<<"m["<<i<<"]["<<j<<"]= "; cin>>m[i][j];
        }
        cout<<endl; // separam elementele de pe randuri
    }
    system("CLS");
    cout<<"Afisam elementele matricei: "<<endl;
    for (i=0;i<x;i++){     // parcurgerea pe linie
        for (j=0;j<y;j++){ // parcurgerea pe coloana
            cout<<"m["<<i<<"]["<<j<<"]= "<<m[i][j]<<"\t";
        }
        cout<<endl; // separam elementele de pe randuri
    }
}
```

3.1.3 Operaţiuni specifice cu elementele unei matrici pătratice, M[n][n]

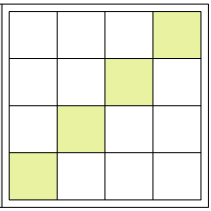
Secvența de cod pentru afișarea elementelor unei matrici de pe diagonala principală:

```
for (i=0;i<x;i++){
    for (j=0;j<y;j++){
        if (i==j) cout<<m[i][j];
    }
    cout<<endl;
}
```



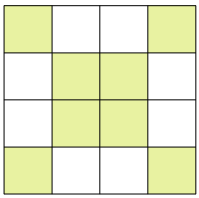
Secvența de cod pentru afișarea elementelor unei matrici de pe diagonala secundară:

```
for (i=0;i<x;i++){
    for (j=0;j<y;j++){
        if (x-i-1==j) cout<<m[i][j];
    }
    cout<<endl;
}
```

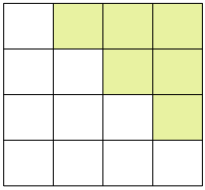
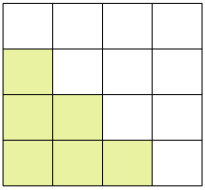
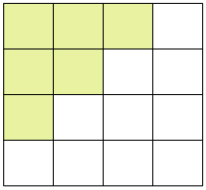
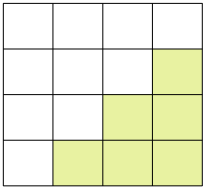


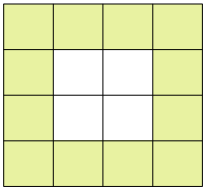
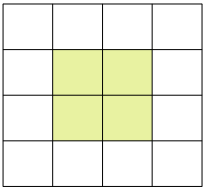
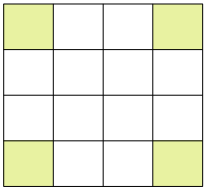
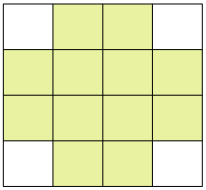
Secvența de cod pentru afișarea elementelor unei matrici de pe diagonala principală și cea secundară împreună:

```
for (i=0;i<x;i++){
    for (j=0;j<y;j++){
        if (x-i-1==j || i==j) cout<<m[i][j];
    }
    cout<<endl;
}
```



Scrieți secvența de cod C++ pentru afișarea elementelor conform condiției:

			
elementele situate deasupra diagonalei principale	*elementele situate sub diagonala principală*	*elementele situate deasupra diagonalei secundare*	*elementele situate sub diagonala secundară*

			
elementele situate la periferie	*elementele situate până la periferie*	*elementele situate în cele 4 vârfuri*	*elementele matricei fără cele 4 vârfuri*

3.1.4 Clasa array în C++ STL

Array este o colecție de obiecte omogene și acest container de vectori este definit pentru vectori de dimensiune constantă sau dimensiune statică. Acest container se înfășoară în jurul vectorilor de dimensiuni fixe și informațiile despre dimensiunea sa nu se pierd atunci când sunt declarate la un pointer. Pentru a utiliza clasa *array*, trebuie să includem antetul acestuia:

```
#include <array>
```

Array este un fel de container secvențial și nu este folosit des în programarea obișnuită sau în programarea competitivă, dar uneori funcțiile sale membru îi oferă un avantaj superior față de un vector obișnuit pe care-l folosim în mod obișnuit la rezolvarea problemelor.

Sintaxă:

```
array <tip, dimensiune> nume;
```

3.1.5 Clasa vector în C++ STL

Vector este la fel ca șivectorul dinamic, având capacitatea de a se redimensiona automat atunci când un element este inserat sau șters, stocarea lui fiind gestionată automat de container. Pentru a utiliza clasa *vector*, trebuie să includem antetul acestuia:

```
#include <vector>
```

Elementele *vector*-ilor sunt plasate în stocare contiguă, astfel încât să poată fi accesate și traversate folosind *iteratori*. În *vector*, datele sunt inserate la sfârșit. Inserarea la sfârșit necesită timp diferențial, deoarece uneori vectorul poate avea nevoie de a fi extins. Eliminarea ultimului element durează doar timp constant, deoarece nu are loc nici o redimensionare. Inserarea și ștergerea la început sau la mijloc este liniară în timp.

Tabelul 3.1: Funcții de capacitate

Nr	Funcția	Explicația
1	size()	returnează numărul de elemente din vector;
2	max_size()	returnează numărul maxim de elemente pe care le poate conține vectorul;
3	capacity()	returnează dimensiunea spațiului de stocare alocat în prezent vectorului, exprimată ca număr de elemente;
4	resize(n)	redimensionează containerul astfel încât să conțină „n” elemente;
5	empty()	returnează dacă containerul este gol;
6	shrink_to_fit()	reduce capacitatea containerului de a se potrivi cu dimensiunea sa și distruge toate elementele dincolo de capacitate;
7	reserve()	solicită ca capacitatea vectorului să fie cel puțin suficientă pentru a conține n elemente.

Exemplul 1: Elaborăm un program C++ pentru a prezenta aplicabilitatea funcțiilor de capacitate.

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> v;
    for (int i=1; i<=5; i++)
       v.push_back(i);
    cout<<"Dimensiunea: \t\t"<<v.size();
    cout<<"\nCapacitatea: \t\t"<<v.capacity();
    cout<<"\nDimensiune maxima: \t"<<v.max_size();
    // verificam daca vectorul este gol sau nu
    if (v.empty()==false) cout<<"\nVectorul nu este gol!";
    else cout<<"\nVectorul este gol!";
    v.shrink_to_fit(); // restrangem vectorul
    v.resize(4); // redimensionam vectorul la dimensiunea 4
    // afisam dimensiunea vectorului dupa redimensionare
```

```
    cout<<"\n\nRedimensionam: \t\t"<<v.size();
    cout<<"\nElementele vectorului sunt: ";
    for (auto it=v.begin(); it!=v.end(); it++){
        cout<<*it<<" ";
    }
}
```

Tabelul 3.2: Funcții de accesare a elementelor

Nr	Funcția	Explicația
1	operator de referință [n]	returnează o referință la elementul din poziția „n” din vector;
2	at(n)	returnează o referință la elementul din poziția „n” din vector;
3	front()	returnează o referință la primul element din vector;
4	back()	returnează o referință la ultimul element din vector;
5	data()	returnează un pointer direct la matricea de memorie folosită intern de vector pentru a stoca elementele deținute.

Exemplul 2: Elaborăm un program C++ pentru a prezenta modalitățile de accesare a elementelor vectorului.

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> v;
    for (int i=1; i<=10; i++)
        v.push_back(i*10); // elementele vor fi 10 20 … 100
    cout<<"\nv[2] \t\t"<<v[2];
    cout<<"\nv.at(4) \t"<<v.at(4);
    cout<<"\nv.front() \t"<<v.front();
    cout<<"\nv.back() \t"<<v.back();
    // pointer catre primul element
    int *pos=v.data();
```

```
    cout<<"\n\nValoarea primul element: "<<*pos;
    cout<<"\nAdresa primului element: "<<pos;
}
```

Tabelul 3.3: Modificatori

Nr	Funcția	Explicația
1	assign()	atribuie o nouă valoare elementelor vectorului prin înlocuirea celor vechi;
2	push_back()	împinge elementele într-un vector din spate;
3	pop_back()	scoate sau elimina elemente dintr-un vector din spate;
4	insert()	inserează elemente noi înaintea elementului în poziția specificată;
5	erase()	elimină elemente dintr-un container din poziția sau intervalul specificat;
6	swap()	schimbă conținutul unui vector cu un alt vector de același tip, dimensiunile pot fi diferite;
7	clear()	elimină toate elementele containerului vectorial
8	emplace()	extinde containerul prin introducerea unui nou element în poziție;
9	emplace_back()	este folosit pentru a insera un nou element în containerul vector, noul element este adăugat la sfârșitul vectorului.

Exemplul 3: Elaborăm un program C++ pentru a prezenta aplicabilitatea modificatorilor.

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> v;  // declaram vectorul v
    v.assign(5,10); // umplem vectorul cu "10" de 5 ori
```

```
    cout<<"Elementele vectorului sunt: ";
    for (int i=0; i<v.size(); i++)
        cout<<v[i]<<" ";
    v.push_back(15); // inseram "15" pe ultima pozitie
    int n=v.size();
    cout<<"\nUltimul element este: \t"<<v[n-1];
    v.pop_back();    // eliminam ultimul element
    v.insert(v.begin(),5);   // inseram "5" la inceput
    cout<<"\nPrimul element este: \t"<<v[0];
    v.erase(v.begin());      // eliminam primul element
    cout<<"\nPrimul element este: \t"<<v[0];
    v.emplace_back(20);      // inseram "20" la sfarsit
    n = v.size();
    cout<<"\nUltimul element este: \t"<<v[n-1];
    v.clear(); // stergem datele vectorului
    cout<<"\nDimensiunea vectorului dupa curatare: "<<v.size();
    vector<int> v1, v2; // declaram vectorii v1 si v2
    v1.push_back(1); v1.push_back(2);
    v2.push_back(3); v2.push_back(4);
    cout<<"\n\nVectorii initiali: \n\tv1: ";
    for (int i=0; i<v1.size(); i++)
        cout<<v1[i]<<" ";
    cout << "\n\tv2: ";
    for (int i=0; i<v2.size(); i++)
        cout<<v2[i]<<" ";
    v1.swap(v2); // interschimbam datele celor 2 vectori
    cout<<"\nVectorii interschimbati: \n\tv1: ";
    for (int i=0; i<v1.size(); i++)
        cout<<v1[i]<<" ";
    cout<<"\n\tv2: ";
    for (int i=0; i<v2.size(); i++)
        cout<<v2[i]<<" ";
}
```

Important

- Fișierele antet <array> și <vector> pot fi studiate mai detaliat accesând site-ul oficial: https://cplusplus.com/.
- Încercați să elaborați o diagramă Venn pentru a evidenția aspectele comune și deosebirile acestor două clase, pentru a le aplica mai ușor.

3.1 Modele de itemi pentru exersare

(1) Fie o matrice pătratică *M[n][n]*, unde *n* este un număr întreg pozitiv. Elaborați un program în C++ care va afișa rezultatul pentru următoarele operațiuni:
   a) afișarea la ecran a elementelor divizibile cu 3 situate pe diagonala principală a matricii;
   b) afișarea la ecran a elementelor nedivizibile cu 7 situate pe diagonala secundară a matricii;
   c) afișarea la ecran a sumei elementelor de pe diagonala principală;
   d) afișarea la ecran a mediei aritmetice a elementelor de pe diagonala secundară.

(2) Fie o matrice dreptunghiulară *M[n][m]*, unde *n* și *m* sunt numere întregi pozitive, iar elementele matricii sun numere naturale de cel puțin două cifre. Elaborați un program în C++ care va afișa rezultatul pentru următoarele operațiuni:
   a) afișarea la ecran a elementelor ce au suma cifrelor un număr par;
   b) afișarea la ecran a elementelor ce au produsul cifrelor un număr impar;
   c) afișarea la ecran a numărului de elemente ce sunt pătrate perfecte;
   d) afișarea la ecran a numărului de elemente ce au cifra zecilor o cifră primă (*cifrele prime sunt: 2, 3, 5 și 7*).

(3) Fie două matrici pătratice $F[2][2]=\begin{pmatrix}3&6\\5&2\end{pmatrix}$ și $G[2][2]=\begin{pmatrix}4&8\\1&9\end{pmatrix}$, unde *n* =2. Elaborați un program în C++ care va afișa rezultatul pentru următoarele operațiuni:
   a) F + G; b) F - G; c) F * G;

3.2 Şiruri de caractere. Funcţii standard. Conversii

În această temă vom analiza următoarele unități de conținut:
- *declararea, inițializarea, citirea și afișarea șirurilor de caractere în C/C++;*
- *prelucrarea datelor șirurilor de caractere conform specificațiilor propuse;*
- *posibilitățile clasei string în C++ STL;*
- *implementarea șirurilor de caractere în limbajul C++.*

Limbajul C++ are în definiția sa o modalitate de a reprezenta o secvență de caractere ca obiect al clasei. Această clasă se numește `std:: string`. Clasa `String` stochează caracterele ca o secvență de octeți cu funcționalitatea de a permite accesarea caracterului de un singur octet. Există două modalități de aplicare a șirurilor de caractere în mod obișnuit în limbajul de programare C++:
- șiruri de caractere care sunt în stilul limbajului C;
- șiruri de caractere care sunt obiecte ale clasei de șiruri (STL C++).

3.2.1 Şiruri de caractere din limbajul C adaptate pentru C++

În programarea C, colecția de caractere este stocată sub formă de vectori de tip char. Acest lucru este acceptat și în programarea C++.

```
char s[] = "C++";
```

În codul de mai sus, *s* este un șir și conține 4 caractere. Deși „C++” are 3 caractere, caracterul nul „\0” este adăugat automat la sfârșitul șirului. Modalități alternative de definire a unui șir de caractere:

```
char s[50] = "abcd";
char s[] = {'a', 'b', 'c', 'd', '\0'};
char s[5] = {'a', 'b', 'c', 'd', '\0'};
```

La fel ca și vectorii, nu este necesar să folosim tot spațiul alocat șirului. De exemplu:

```
char s[100] = "C++";
```

Exemplul 1: Elaborăm un program C++ pentru a afișa mai întâi un șir (cuvânt) introdus de utilizator de la tastatură, apoi vom afișa un șir (enunț) introdus de utilizator de la tastatură.

```
#include <iostream>
using namespace std;
int main(){
    char s1[100],s2[100];
    // citim și afisam un cuvant
    cout<<"Introduceti un cuvant: "; cin>>s1;
    cout<<"Ati introdus cuvantul: "<<s1<<endl;
    // citim și afisam un enunt
    cout<<"Introduceti un enunt: "; cin.get(s2,100);
    cout<<"Ati introdus enuntul: "<<s2<<endl;
}
```

Pentru a citi textul care conține spațiu liber, se poate folosi funcția *cin.get*. Această funcție are două argumente, primul argument este numele șirului (adresa primului element al șirului), iar al doilea argument este dimensiunea maximă a vectorului. În programul de mai sus, *s* este numele șirului și *100* este dimensiunea maximă a vectorului.

În limbajul C++ putem crea și un obiect de tip șir pentru a păstra șirurile de caractere. Spre deosebire de utilizarea vectorilor de caractere, obiectele șir nu au lungime fixă și pot fi extinse conform cerințelor dvs.

Funcțiile de gestionare a șirurilor de caractere sunt definite în fișierul antet `<<string.h>>` sau se poate întâlni și `<<cstring>>`. Funcții utilizate în mod obișnuit cu șirurile de caractere în C care pot fi aplicate și în C++:

- *strlen()* - calculează lungimea unui șir;
- *strcpy()* - copiază un șir în altul;
- *strcmp()* - compară două șiruri;
- *strcat()* - concatenează două șiruri;
- *strlwr()* - convertește șirul în minuscule;
- *strupr()* - convertește șirul în majuscule.

Exemplul 2: Elaborăm un program C++ pentru a prelucra funcțiile menționate anterior și a afișa rezultatele la consolă.

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
    char s1[20]=" informatica ", s2[30]=" matematica ";
    // afisarea celor doua cuvinte
    cout<<"\nCuvantul 1: "<<s1<<endl;
    cout<<"Cuvantul 2: "<<s2<<endl;
    // lungimea celor doua cuvinte
    cout<<"\nLungimea cuvantului 1: "<<strlen(s1)<<endl;
    cout<<"Lungimea cuvantului 2: "<<strlen(s2)<<endl;
    // functia de comparare (afiseaza 0 daca cuvintele coincid)
    cout<<"\nCompararea cuvintelor: "<<strcmp(s2,s1)<<endl;
    // functia de concatenare
    cout<<"\nConcatenarea cuvintelor: "<<strcat(s1,s2)<<endl;
    // functia de conversie in majuscule / minuscule
    cout<<"\nConversia in majuscule: "<<strupr(s2)<<endl;
    cout<<"Conversia in minuscule: "<<strlwr(s2)<<endl;
    // functia de copiere
    cout<<"\nCopierea cuvantului \"fizica\" in cuvintul 1: ";
    cout<<strcpy(s1,"fizica")<<endl;
}
```

3.2.2 Clasa string în C++ STL

Operațiile ce pot fi efectuate cu șirurile de caractere vor fi repartizate în mai multe categorii: funcții de intrare, funcții de capacitate, funcții de iterare și funcții de manipulare.

Tabelul 3.4: Funcții de intrare

Nr	Funcția	Explicația
1	getline();	este folosită pentru a stoca un flux de caractere introduse de utilizator în memoria obiectului;
2	push_back();	este folosită pentru a introduce un caracter la sfârșitul șirului;

Nr	Funcția	Explicația
3	pop_back();	introdusă începând cu standartul C++11, este folosită pentru a șterge ultimul caracter din șir.

Exemplul 3: Elaborăm un program C++ pentru a prezenta aplicabilitate funcțiilor de intrare.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    string s;
    // citirea și afisarea unui sir de caractere
    cout<<"Introduceti un enunt: "; getline(cin,s);
    cout<<"Ati introdus enuntul: "<<s<<endl;
    // inserarea unui caracter la sfarsit
    s.push_back('1');
    // afisarea sirului de caractere dupa inserare
    cout<<"Variabila dupa operatiunea push_back: "<<s<<endl;
    // stergerea unui caracter de la sfarsit
    s.pop_back();
    // afisarea sirului de caractere dupa stergere
    cout<<"Variabila dupa operatiunea pop_back: "<<s<<endl;
}
```

Tabelul 3.5: Funcții de capacitate

Nr	Funcția	Explicația
1	capacity();	returnează capacitatea alocată șirului, care poate fi egală sau mai mare decât dimensiunea șirului, spațiul suplimentar este alocat astfel încât atunci când noile caractere sunt adăugate în șir, operațiunile pot fi efectuate eficient;
2	resize();	modifică dimensiunea șirului, dimensiunea poate fi mărită sau micșorată;
3	length();	determină lungimea șirului de caractere;

Nr	Funcția	Explicația
4	shrink_to_fit();	scade capacitatea șirului și o face egală cu capacitatea minimă a șirului, această operațiune este utilă pentru a economisi memorie suplimentară dacă suntem siguri că nu mai trebuie adăugate caractere.

Exemplul 4: Elaborăm un program C++ pentru a prezenta aplicabilitate funcțiilor de capacitate.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    // initializarea sirului de caractere
    string s="Eu sunt elev si iubesc sa invat.";
    cout<<"Afisam enuntul initial: "<<s<<endl;
    // afisam lungimea sirului
    cout<<"Lungimea sirului este: "<<s.length()<<endl;
    // afisam capacitatea sirului
    cout<<"Capacitatea sirului este: "<<s.capacity()<<endl;
    s.resize(12); // redimensionarea sirului folosind resize()
    cout<<"\nDupa redimensionare: "<<s<<endl;
    // afisam lungimea sirului
    cout<<"Lungimea sirului este: "<<s.length()<<endl;
    // diminuarea capacității sirului folosind shrink_to_fit()
    s.shrink_to_fit();
    cout<<"Capacitate sirului este: "<<s.capacity()<<endl;
}
```

Tabelul 3.6: Funcții de iterare

Nr	Funcția	Explicația
1	begin();	returnează un iterator la începutul șirului;
2	end();	returnează un iterator la sfârșitul șirului;
3	rbegin();	returnează un iterator invers, indică sfârșitul șirului;
4	rend();	returnează un iterator invers, indică începutul șirului.

Exemplul 5: Elaborăm un program C++ pentru a prezenta aplicabilitate funcțiilor de iterare.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    string s="computer";
    string::iterator i1; // declar un iterator
    string::reverse_iterator i2; // declar un iterator invers
    // afisarea standard a sirului de caractere
    cout<<"Sirul care utilizeaza iteratorii inainte: \t";
    for(i1=s.begin(); i1!=s.end(); i1++){
       cout<<*i1;
    }
    // afisarea inversa a sirului de caractere
    cout<<"\nSirul care utilizeaza iteratorii inapoi: \t";
    for(i2=s.rbegin(); i2!=s.rend(); i2++){
       cout<<*i2;
    }
}
```

Important

- În limbajul C++, pointerii sunt variabile care stochează adresele de memorie ale altor variabile.
- Operatorul „*” este folosit inainte de variabilele „i1” și „i2” pentru a declara pointerii, iar operatorul „&” este folosit pentru a ne oferi adesa unei variabile în memorie, de exemplu: &it.

Tabelul 3.7: Funcții de manipulare

Nr	Funcția	Explicația
1	copy("char array", len, pos);	copiază subșirul din vectorul de caractere țintă menționat în argumentele sale, este nevoie de 3 argumente, vectorul de caractere țintă, lungime pentru a fi copiată și poziția de pornire în șir pentru a începe copierea;

Nr	Funcția	Explicația
2	swap();	interschimbă un șir cu altul.

Exemplul 6: Elaborăm un program C++ pentru a prezenta aplicabilitate funcțiilor de manipulare.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    string s1="informatica si matematica";
    string s2="biologia si chimia";
    // operatiunea de copiere
    char c[80];
    s1.copy(c,15,0);
    cout<<"Vectorul de caractere nou: "<<c<<endl;
    // afisam sirurile initiale
    cout<<"\nAfisam sirul 1: "<<s1<<endl;
    cout<<"Afisam sirul 2: "<<s2<<endl;
    // interschimbam sirurile
    s1.swap(s2);
    // afisam sirule dupa interschimbare
    cout<<"\nAfisam sirul 1 dupa interschimare: "<<s1<<endl;
    cout<<"Afisam sirul 2 dupa interschimare: "<<s2<<endl;
}
```

Important

- Fișierele antet <cstring> și <string> pot fi studiate mai detaliat, deoarece au fost analizate doar cele mai aplicabile funcții, celelalte funcții pot fi studiate pe site-ul oficial: https://cplusplus.com/.
- Clasa *string* include următoarele compartimente: tipuri de membri, funcții ale membrilor, constantele membre și supraîncărcări ale funcțiilor non-membre. Compartimentele fundamentale le-am analizat, mai multe funcții ce pot fi aplicate asupra clasei *string* pot fi studiate pe site-ul oficial anexat mai sus.

Tabelul 3.8: Șiruri de caractere C++ vs C

Clasa string `string s="sir de caractere";`	**Vector de caractere** `char s[100]="sir de caractere";`
Un string este o clasă care definește obiecte care sunt reprezentate ca un flux de caractere.	Un vector de caractere este pur și simplu un șir de caractere care poate fi terminat cu un caracter nul.
În cazul string-urilor, memoria este alocată dinamic. Mai multă memorie poate fi alocată în timpul rulării, la cerere, deoarece nicio memorie nu este prealocată, nicio memorie nu este irosită.	Dimensiunea vectorului de caractere trebuie alocată static, nu poate fi alocată mai multă memorie în timpul execuției, dacă este necesar. Memoria alocată și neutilizată este, de asemenea, irosită.
Deoarece string-urile sunt reprezentate ca obiecte, nu are loc nicio dezintegrare a vectorului.	Există o amenințare de dezintegrare a vectorului în cazul vectorului de caractere.
String-urile sunt mai lente în comparație cu implementarea decât vectorul de caractere.	Implementarea vectorului de caractere este mai rapidă decât string.
Clasa String definește o serie de funcționalități care permit operații multiple pe șiruri de caractere.	Vectorii de caractere nu oferă multe funcții încorporate pentru a manipula șirurile decaractere.

3.2 Modele de itemi pentru exersare

(1) Elaborați un program C++ care va citi de la tastatură două șiruri de caractere *s1* și *s2*, apoi va concatena primele trei caractere din *s2* la *s1*. Pentru a rezolva problema aplicați funcția *strncat()*.

(2) Elaborați un program C++ care va citi de la tastatură un șir de caractere *s* și apoi va mai citi separat un caracter *c*. Dacă caracterul c se va găsi în *s*, atunci să se afișeze toate pozițiile din *s* unde se găsește caracterul *c*. Pentru a rezolva problema aplicați funcția *strchr()*.

(3) Elaborați un program C++ care va citi de la tastatură două șiruri de caractere *s1* și *s2*. Dacă *s2* este subșir al lui *s1* se va afișa poziția în *s1* unde se găsește. Pentru a rezolva problema aplicați funcția *strstr()*.

(4) Elaborați un program C++ care va citi de la tastatură două cuvinte *c1* și *c2, apoi va afișa* aceste cuvinte în ordine lexicografică Pentru a rezolva problema aplicați funcția *strcmp()*.

(5) Elaborați un program C++ care va citi de la tastatură un text *t* ce conține o listă de fructe și legume separate prin spațiu, virgule sau punct. Să se afișeze cuvintele câte unul pe o linie şi să se numere cuvintele din text. Pentru a rezolva problema aplicați funcția *strtok()*.

(6) Elaborați un program C++ care va citi de la tastatură un număr întreg și o să afișeze la ecran acest număr ca un șir de caractere folosind baza 2 de numerație. Pentru a rezolva problema aplicați funcția *itoa()*.

(7) Elaborați un program C++ care va citi de la tastatură două șiruri de caractere *s1* și *s2*. Să se elimine toate apariţiile lui *s2* în *s1*, dacă se știe că șirul *s2* conține doar un cuvânt. Pentru a rezolva problema aplicați funcțiile: *strstr()* și *strcpy()*.

3.3 Pointeri. Alocarea dinamică a memoriei

În această temă vom analiza următoarele unități de conținut:

- *declararea, inițializarea și accesarea variabilelor de tip pointer în C/C++;*
- *identificarea instrucțiunilor de referențiere / dereferențiere;*
- *alocarea dinamică a memoriei și eliberarea memoriei pentru tipul de date tablou (vector / matrice) și string;*
- *prelucrarea pointerilor conform specificațiilor propuse;*
- *implementarea pointerilor în limbajul C++.*

Pointerii în C++ sunt ușor și distractiv de învățat. Unele sarcini C++ sunt efectuate mai ușor cu pointeri, iar alte, cum ar fi alocarea dinamică a memoriei, nu pot fi efectuate fără pointeri.

3.3.1 Valoarea şi adresa unei variabile

După cum cunoaștem, fiecare variabilă este o locație de memorie și fiecare locație de memorie are adresa sa definită, care poate fi accesată folosind operatorul „&”, care denotă o adresă în memorie. Vom lua în considerare următoarele declarații care vor tipări adresa variabilelor definite:

```
#include <iostream>
using namespace std;
int main () {
   int  var1; char var2[10];
   cout<<"Adresa variabilelor: "<<&var1<<" "<<&var2<<endl;
}
```

După compilare și execuție, se va afișa la consolă adresele variabilelor:

```
Adresa variabilelor: 0x61fe1c 0x61fe12
```

Un pointer este o variabilă a cărei valoare este adresa unei alte variabile. Forma generală a unei declarații de variabilă pointer:

```
tip *numeVariabila;
```

Aici, *tip* este tipul de bază al indicatorului; trebuie să fie un tip C++ valid și *numeVariabila* este numele variabilei pointer. Asteriscul (*), pe care l-am folosit pentru a declara un pointer este același asterisc pe care îl utilizăm pentru înmulțire. Cu toate acestea, în această declarație, asteriscul este folosit pentru a desemna o variabilă ca pointer. Următoarele exemple reprezintă unele declarații valide ale pointerilor:

```
int    *i;    // pointer de tip integer
double *d;    // pointer de tip double
float  *f;    // pointer de tip float
char   *c;    // pointer de tip character
```

Tipul de date actual al valorii tuturor pointerilor, indiferent dacă este întreg, flotant, caracter sau altul, este același, adică un număr hexazecimal lung care reprezintă o adresă de memorie. Singura diferență între pointerii de diferite tipuri de date este tipul de date al variabilei sau constantei către care indică pointerul.

Sunt puține operații importante, pe care le vom efectua cu pointerii foarte frecvent:

(a) definim o variabilă pointer;
(b) atribuim adresa unei variabile unui pointer;
(c) accesăm valoarea la adresa disponibilă în variabila pointer, acest lucru se face prin utilizarea operatorului unar „*” care returnează valoarea variabilei situată la adresa specificată de operandul acesteia.

```
#include <iostream>
using namespace std;
int main () {
   int  a=20;  // declararea implicita a variabilei
   int  *b;    // variabila de tip pointer
   b=&a;       // stocam adresa variabilei a
   cout<<"Valoarea variabilei var: "<<a<<endl;
   cout<<"Adresa de memorie stocata variabilei b: "<<b<<endl;
   cout<<"Valoarea variabilei *b: "<<*b<<endl;
}
```

3.3.2 Pointerul null. Operaţii cu pointeri.

Limbajul C++ acceptă pointerul nul. Este întotdeauna o practică bună să atribuim pointerul NULL unei variabile pointer în cazul în care nu avem o adresă exactă de atribuit. Acest lucru se face în momentul declarării variabilei. Un pointer căruia i se atribuie NULL se numește pointer nul. Pointerul NULL este o constantă cu o valoare zero definită în mai multe biblioteci standard, inclusiv iostream.

```
#include <iostream>
using namespace std;
int main () {
   int  *a = NULL;
   cout<<"Valoarea variabilei a este "<<a;
}
```

Pe majoritatea sistemelor de operare, programelor nu li se permite accesul la memoria la adresa 0, deoarece acea memorie este rezervată de sistemul de operare. Cu toate acestea, adresa de memorie 0 are o semnificație specială; semnalează că indicatorul nu este destinat să indice o locație de memorie accesibilă.

Prin convenție, dacă un pointer conține valoarea nulă (zero), se presupune că nu indică la nimic. Pentru a verifica un pointer nul, puteți utiliza o instrucțiune *if* după cum urmează:

```
if (a)  // reuseste daca a nu este nul
if (!a) // reuseste daca a este nul
```

Astfel, dacă toți pointerii neutilizați primesc valoarea nulă și evităm utilizarea unui pointer nul, putem evita utilizarea greșită accidentală a unui pointer neinițializat. De multe ori, variabilele neinițializate dețin niște valori nedorite și devine dificilă depanarea programului.

Deoarece pointerul este o adresă care reprezintă o valoare numerică; prin urmare, putem efectua operații aritmetice la fel ca și cu o valoare numerică. Există patru operatori aritmetici care pot fi aplicate pointerilor: „+”, „++”, „ -” și „--”.

Pentru a înțelege aritmetica pointerilor, să considerăm că *p* este un pointer întreg care indică adresa 1000. Presupunând numere întregi de 32 de biți, să efectuăm următoarea operație aritmetică pe pointer:

```
p++;
```

Astfel, variabila *p* va indica locația 1004, deoarece de fiecare dată când *p* este incrementat, va indica următorul întreg. Această operațiune va muta indicatorul la următoarea locație de memorie fără a afecta valoarea reală din locația de memorie. Dacă *p* indică un caracter a cărui adresă este 1000, atunci operația de mai sus va indica locația 1001, deoarece următorul caracter va fi disponibil la locația 1001.

Preferăm să folosim un pointer în programul nostru în loc de un vector, deoarece pointerul variabil poate fi incrementat, spre deosebire de numele vectorului care nu poate fi incrementat, deoarece este un pointer constant. Aceleași considerații se aplică și pentru decrementarea unui pointer, care își scade valoarea cu numărul de octeți ai tipului său de date.

Pointerii pot fi comparați utilizând operatori relaționali, cum ar fi: „==”, „<” și „>”. Dacă *p1* și *p2* indică variabile care sunt legate între ele, cum ar fi elementele aceluiaș vector, atunci *p1* și *p2* pot fi comparate în mod semnificativ.

```
#include <iostream>
using namespace std;
const int MAX=3;
int main () {
    int  v[MAX]={10,100,200},*a;
    // inrementarea si compararea
    a=v;
    for (int i=0; i<MAX && a<=&v[MAX-1]; i++){
        cout<<"Adresa variabilei a["<<i<<"]= "<<a<<endl;
        cout<<"Valoarea lui a["<<i<<"]= "<<*a<<endl<<endl;
        a++;
    }
    // decrementarea
    a=&v[MAX-1];
    for (int i=MAX; i>0; i--){
        cout<<"Adresa variabilei a["<<i<<"]= "<<a<<endl;
```

```
        cout<<"Valoarea lui a["<<i<<"]= "<<*a<<endl<<endl;
        a--;
    }
}
```

3.3.3 Pointeri şi vectori. Vector de pointeri.

Pointerii și vectorii sunt strâns legați, de fapt sunt interschimbabili în multe cazuri. De exemplu, un pointer care indică la începutul unui vector poate accesa acel vector utilizând fie aritmetica pointerului, fie indexarea în stilul vectorului.

Cu toate acestea, pointerii și vectorii nu sunt complet interschimbabili, de exemplu, fie următorul program:

```
#include <iostream>
using namespace std;
const int MAX=3;
int main () {
    int  v[MAX]={10,100,200},*a;
    for (int i=0; i<MAX; i++){
        *v=i; // sintaxa este corecta
        v++;  // sintaxa este incorecta
    }
}
```

Este perfect acceptabil să aplicăm operatorul pointer * la *v*, dar este ilegal să modificăm valoarea *v*. Motivul este că *v* este o constantă care indică începutul unui vector și nu poate fi folosită ca *l-valoare*.

Un nume de vector generează o constantă pointer, acesta poate fi folosită în continuare în expresii în stil pointer, atâta timp cât nu este modificat. De exemplu, următoarea declarație este o instrucțiune validă care atribuie lui v[2] valoarea 500:

```
p++;
```

Înainte de a înțelege conceptul de vector de pointeri, să luăm în considerare un exemplu, care utilizează un vector de 3 numere întregi.

Poate exista o situație în care dorim să menținem un vector, care poate stoca pointeri către un *int* sau *char* sau orice alt tip de date disponibil. Urmează declararea unui vector de pointeri către un număr întreg.

```
int *p[MAX];
```

Aceasta declară *p* ca un vector de dimeniunea *MAX* pointeri întregi. Astfel, fiecare element din *p* deține acum un pointer către o valoare *int*. De asemenea, putem utiliza un vector de pointeri către o valoare *char*, pentru a stoca o listă de șiruri de caractere.

```
#include <iostream>
using namespace std;
const int MAX=5;
int main () {
    // vector de pointeri de tip int
    int  v[MAX]={10,20,30,40,50},*a[MAX];
    // vector de pointeri de tip char
    char *zile[MAX]={"Luni","Marti","Miercuri","Joi","Vineri"};
    for (int i=0; i<MAX; i++){
        a[i]=&v[i]; // atribuie adresa unui intreg
        cout<<"Valoarea lui v["<<i<<"]= "<<*a[i]<<endl;
        cout<<"Valoarea lui zile["<<i<<"]= "<<*(zile+i);
        cout<<endl<<endl;
    }
}
```

3.3.4 Pointer către pointer.

Un pointer către un pointer este o formă de indirecție multiplă sau un lanț de pointeri. În mod normal, un pointer conține adresa unei variabile. Când definim un pointer către un pointer, primul pointer conține adresa celui de-al doilea pointer, care indică către locația care conține valoarea reală.

O variabilă care este un pointer către un pointer trebuie declarată ca atare. Acest lucru se face prin plasarea unui asterisc suplimentar în fața numelui său. De exemplu, următoarea secvență este sintaxa de a declara un pointer către un pointer:

```
int **p;  // un pointer catre pointer de tip int
char **s; // un pointer catre pointer de tip char
```

Când o valoare țintă este indicată indirect de către un pointer către un pointer, accesarea acelei valori necesită ca operatorul asterisc să fie aplicat de două ori, așa cum se arată în exemplul de mai jos:

```
#include <iostream>
using namespace std;
int main () {
   int  v=3000,*p1,**p2;
   // preia adresa lui v
   p1=&v;
   // preia adresa lui p1 folosind operatorul &
   p2=&p1;
   // preia valoarea folosind p2
   cout<<"Valoarea lui v: \t"<<v<<endl;
   cout<<"Valoarea lui *p1: \t"<<*p1<<endl;
   cout<<"Valoarea lui **p2: \t"<<**p2<<endl;
}
```

Rezultat obținut la consolă după execuție:

```
Valoarea lui v:         3000
Valoarea lui *p1:       3000
Valoarea lui **p2:      3000
```

3.3.5 Memoria dinamică. Operatorii new şi delete.

Toate nevoile de memorie sunt determinate înainte de executarea programului prin definirea variabilelor necesare. Dar pot exista cazuri în care nevoile de memorie ale unui program pot fi determinate doar în timpul rulării. De exemplu, când memoria necesară depinde de datele de intrare ale utilizatorului. În aceste cazuri, programele trebuie să aloce dinamic memorie, pentru care limbajul C++ integrează operatorii new și delete.

Memoria dinamică este alocată utilizând operatorul new. Acesta este urmat de un specificator de tip de date și, dacă este necesară o secvență de

mai mult de un element, numărul acestora va fi între paranteze pătrate, [ ]. Așadar, returnează un pointer la începutul noului bloc de memorie alocat.

Sintaxă:

```
pointer = new tipul1;
pointer = new tipul2 [numar_de_elemente];
```

Prima expresie este utilizată pentru a aloca memorie pentru a conține un singur element de tip *tipul1*. Al doilea este folosit pentru a aloca un bloc (un vector) de elemente de tip *tipul2*, unde *numar_de_elemente* este o valoare întreagă reprezentând cantitatea acestora. De exemplu:

```
int *f1;
f1 = new int[5];
```

În acest caz, sistemul alocă dinamic spațiu pentru 5 elemente de tip int și returnează un pointer la primul element al secvenței, care este atribuit lui *f1* (un pointer). Prin urmare, *f1* indică acum un bloc valid de memorie cu spațiu pentru 5 elemente de tip *int*.

Există o diferență substanțială între declararea unui vector normal și alocarea memoriei dinamice pentru un bloc de memorie folosind new. Cea mai importantă diferență este că dimensiunea unui vector obișnuit trebuie să fie o expresie constantă și astfel dimensiunea acesteia trebuie determinată în momentul proiectării programului, înainte de a fi rulat, în timp ce alocarea dinamică a memoriei efectuată de new permite atribuiți memorie în timpul rulării folosind orice valoare variabilă ca dimensiune.

Memoria dinamică solicitată de programul nostru este alocată de sistem din heap-ul de memorie. Cu toate acestea, memoria computerului este o resursă limitată și poate fi epuizată. Prin urmare, nu există garanții că toate cererile de alocare a memoriei folosind operator new vor fi acordate de către sistem.

C++ oferă două mecanisme standard pentru a verifica dacă alocarea a avut succes:

(1) Gestionarea excepțiilor. Folosind această metodă, o excepție de tip *bad_alloc* este aruncată atunci când alocarea eșuează. Această me-

todă de excepție este metoda folosită implicit prin new și este cea folosită într-o declarație ca:

```
f1 = new int[5]; // daca alocarea esueaza, atunci se arunca o exceptie
```

(2) Nothrow. Când o alocare de memorie eșuează, în loc să arunce o excepție *bad_alloc* sau să închei programul, pointerul returnat de new este un pointer nul, iar programul își continuă execuția în mod normal. Această metodă poate fi specificată folosind un obiect special numit *nothrow*, declarat în antetul <new>, ca argument pentru *new*:

```
f1 = new (nothrow) int[5];
```

În acest caz, dacă alocarea acestui bloc de memorie eșuează, eșecul poate fi detectat verificând dacă f1 este un pointer nul:

```
int *f1;
f1 = new (nothrow) int[5];
if (f1 == nullptr) {
  // eroare la atribuirea memoriei
}
```

Important

- Această metodă *nothrow* este probabil să producă cod mai puțin eficient decât excepțiile, deoarece implică verificarea explicită a valorii pointerului returnat după fiecare alocare.
- Mecanismul de excepție este în general preferat, cel puțin pentru alocările critice. Cu toate acestea, majoritatea exemplelor pot folosi mecanismul nothrow datorită simplității sale.

În cele mai multe cazuri, memoria alocată dinamic este necesară doar în anumite perioade de timp în cadrul unui program. Odată ce nu mai este necesară, poate fi eliberată, astfel încât memoria să devină din nou disponi-

bilă pentru alte solicitări de memorie dinamică. Acesta este scopul operatorului delete.

Sintaxă:

```
delete pointer;
delete[] pointer;
```

Prima declarație eliberează memoria unui singur element alocat folosind *new*, iar a doua eliberează memoria alocată pentru un vector de elemente care utilizează *new* și o dimensiune între paranteze pătrate, [ ].

Valoarea transmisă ca argument pentru *delete* va fi fie un pointer către un bloc de memorie alocat anterior cu *new*, sau fie un pointer nul. În cazul unui pointer nul, *delete* nu produce niciun efect.

```
#include <iostream>
#include <new>
using namespace std;
int main (){
    int i,n,*p;
    cout<<"Introduceti dimensiunea vectorului: ";
    cin>>n;
    p=new(nothrow) int[n];
    if (p==nullptr) cout<<"Memoria nu poate fi alocata!";
    else{
        for (i=0;i<n;i++){
            cout<<"a["<<i<<"]= "; cin>>p[i];
        }
        system("CLS");
        cout<<"Ati introdus: "<<endl;
        for (i=0;i<n;i++){
            cout<<p[i]<<" ";
        }
        delete[] p;
    }
}
```

Observăm cum valoarea dintre paranteze în noua instrucțiune este o valoare variabilă introdusă de utilizator, *n*, nu este o expresie constantă:

```
p=new (nothrow) int[n];
```

Există întotdeauna posibilitatea ca utilizatorul să introducă o valoare pentru *n* atât de mare încât sistemul nu poate aloca suficientă memorie pentru aceasta. De exemplu, când am încercat să dăm o valoare de un miliard, sistemul nostru nu a putut aloca atât de multă memorie pentru program și am primit mesajul text pe care l-am pregătit pentru acest caz, „*Memoria nu poate să fie alocata!*”.

Este considerată o bună practică ca programele să fie întotdeauna capabile să gestioneze eșecurile de alocare a memoriei, fie prin verificarea valorii pointerului (dacă nu este aruncată), fie prin capturarea excepției adecvate.

Rezultat obținut la consolă după execuție:

```
Introduceti dimensiunea vectorului: 5
a[0]= 1
a[1]= 2
a[2]= 3
a[3]= 4
a[4]= 5

Ati introdus:
1 2 3 4 5
```

3.3 Modele de itemi pentru exersare

(1) Elaborați un program C++ care va citi de la tastatură 3 numere naturale și va determina elementul situat între maxiul și minimul dintre aceste trei numere. Toate numerele vor fi alocate dinamic.

(2) Elaborați un program C++ care va citi de la tastatură 5 numere naturale și va determina câte dintre acestea au suma cifrelor egală cu un număr par. Toate numerele vor fi alocate dinamic.

(3) Elaborați un program C++ care va determina elementul maxim și cel minim dintr-un vector alocat dinamic cu numere întregi.

(4) Elaborați un program C++ care va determina suma elementelor pare și produsul elementelor impare dintr-un vector alocat dinamic cu numere întregi.

(5) Elaborați un program C++ care va determina elementul maxim și prim și cel minim și pătrat perfect dintr-o matrice alocată dinamic cu numere întregi.

(6) Elaborați un program C++ care va determina suma elementelor pare de pe diagonala prinipală și produsul elementelor impare de pe diagonala secundară dintr-o matrice pătratică alocată dinamic cu numere întregi.

(7) Elaborați un program C++ care va determina numărul de vocale și numărul de consoane dintr-un enunț. Pentru rezolvarea problemei se vor aplica pointerii.

(8) Elaborați un program C++ care va determina numărul de silabe „oa” și „ao” dintr-un enunț. Pentru rezolvarea problemei se vor aplica pointerii.

3.4 Fişiere. Funcţii de prelucrare a datelor

În această temă vom analiza următoarele unități de conținut:
- *declararea variabilelor de tip fișier (de intare / de ieșire);*
- *identificarea instrucțiunilor de deschidere / închidere ale fișierelor;*
- *prelucrarea fișierelor conform specificațiilor propuse;*
- *implementarea algoritmilor de prelucrare a fișierelor în limbajul C++.*

Înainte de a aborda fiecare subdiviziune, să aflăm mai întâi despre fișierul antet pe care îl vom folosi pentru a obține acces la metoda de gestionare a fișierelor. În limbajul C++, biblioteca *fstream* este folosită pentru a gestiona fișierele și este tratată cu ajutorul a 3 clase: *ofstream*, *ifstream* și *fstream*.

- *ofstream* - această clasă ajută la crearea și scrierea datelor în fișierul obținut din rezultatul programului, este cunoscut și sub numele de flux de ieșire;
- *ifstream* - folosim această clasă pentru a citi date din fișiere și este cunoscută și sub numele de flux de intrare;
- *fstream* - această clasă este combinația dintre ofstream și ifstream. Oferă capacitatea de a crea, scrie și citi un fișier.

Pentru a accesa următoarele clase, trebuie să includem *fstream* ca fișier antet, cum declarăm *iostream* în antet.

```
#include <iostream>
#include <fstream>
```

C++ ne oferă patru operațiuni diferite pentru manipularea fișierelor:
- open() – acesta este folosită pentru a crea un fișier;
- read() – acesta este folosită pentru a citi datele din fișier;
- write() – acesta este folosită pentru a scrie date noi în fișier;
- close() – acesta este folosită pentru a închide fișierul.

Înainte de a putea fi deschis un flux, trebuie creat un obiect de flux. Deschiderea unui flux înseamnă stabilirea unui canal între programul C++ și fișierul de pe disc. Aceasta se realizează prin secvența de caractere care se va muta în fișier; sau prin secvența de caractere care va părăsi fișierul și va ajunge la program; sau prin care secvența de caractere se vor deplasa încoace și încolo. Un flux este deschis numai pentru scriere (ieșire), citire (intrare) sau atât pentru citire, cât și pentru scriere. Poate fi deschis și din alte motive.

3.4.1 Deschiderea fișierelor

Pentru a citi sau a introduce date într-un fișier, trebuie să îl deschidem mai întâi. Acest lucru poate fi realizat cu ajutorul lui „*ifstream*" pentru citire și „*fstream*" sau „*ofstream*" pentru scriere sau adăugare la fișier. Toate aceste trei obiecte au funcția *open()* predefinită.

```
open("NumeFisier", Metoda);
```

NumeFisier ne indică numele fișierului sau calea către acest fișier care trebuie deschis, iar *Metoda* este un mod de a deschide un fișier.

Tabelul 3.9: Descrierea metodelor pentru fișiere

Metoda / Modul	Descrierea
iso::in	Fișier deschis în modul citire;
iso::out	Fișier deschis în modul de scriere;
iso::app	Fișierul a fost deschis în modul adăugare;
iso::ate	Fișierul a fost deschis în modul adăugare, dar citirea și scrierea au fost efectuate la sfârșitul fișierului;
iso::binary	Fișier deschis în modul binar;
iso::trunc	Fișier deschis în modul truncare;
iso::nocreate	Fișierul se deschide numai dacă există;
iso::noreplace	Fișierul se deschide numai dacă nu există;

3.4.2 Verificarea steagurilor de stare

Următoarele funcții membre există pentru a verifica stări specifice ale unui flux, toate returnează o valoare te tip bool.

Tabelul 3.10: Steaguri de stare

Steagul	Descrierea
bad();	Returnează true dacă o operație de citire sau scriere eșuează. De exemplu, în cazul în care încercăm să scriem într-un fișier care nu este deschis pentru scriere sau dacă dispozitivul pe care încercăm să scriem nu mai are spațiu.
fail();	Returnează true în aceleași cazuri ca bad(), dar și în cazul în care apare o eroare de format, cum ar fi atunci când un caracter alfabetic este extras atunci când încercăm să citim un număr întreg.
eof();	Returnează adevărat dacă un fișier deschis pentru citire a ajuns la sfârșit. Funcția membru *eof()* returnează 1, când se ajunge la sfârșitul fișierului și 0 în caz contrar. Programul citește caracterele fișierului, unul câte unul, până când se ajunge la sfârșitul fișierului.
good();	Este cel mai generic steag de stat: returnează false în aceleași cazuri în care apelarea oricăreia dintre funcțiile anterioare ar returna true. Rețineți că binele și răul nu sunt exact opuse (binele verifică mai multe steaguri de stat deodată).
clear();	Funcția membru clear() poate fi folosită pentru a reseta steaguri de stare.

3.4.3 Scrierea datelor în fişier şi citirea datelor din fişier

Până acum, am învățat cum să creăm fișierul folosind C++. Acum, vom învăța cum să scriem date în fișierul pe care l-am creat înainte. Vom folosi obiectele *fstream* sau *ofstream* pentru a scrie date în fișier. Pentru a realiza aceasta, vom aplica operatorul de inserare a fluxului (`<<`), împreună cu textul cuprins între ghilimele duble.

```
VarFisier<<"Textul inserat care va fi scris în fisier";
```

Obținerea datelor din fișier este un lucru esențial de efectuat, deoarece fără a obține datele, nu putem îndeplini nicio sarcină. Putem efectua citirea datelor dintr-un fișier cu *cin* pentru a obține date de la utilizator, dar apoi folosim *cin* pentru a prelua intrări de la consola standard a utilizatorului. Aici vom folosi *fstream* sau *ifstream*. Cu ajutorul funcției *open()*, vom crea un fișier nou numit „*VarFisier*” și apoi vom seta modul la „*ios::out*”, deoarece trebuie să scriem datele în fișier.

```
VarFisier.open("D:/Exemple/Fisier.txt", ios_base::out);
```

De asemenea, putem folosi *VarFisier* pentru a citi din fișier. Aici, `VarFisier>>mesaj` preia fluxul de fișiere, care sunt datele fișierului și folosește un delimitator de spațiu (îl descompune prin spații albe) și apoi pune conținutul în variabila *mesaj*.

```
ifstream VarFisier("FisierulMeu.txt");
string mesaj;
while(VarFisier>>mesaj) {
     cout<<mesaj<<" "; // afisarea are loc in rand
}
```

Înainte de a deschide un flux, obiectul flux trebuie să fie construit. Cel mai simplu mod de a-l exprima este:

```
fstream VarFisier;
```

Deoarece funcția membru *open()* returnează void, pentru a cunoaște dacă fișierul de pe disc a fost deschis cu succes, utilizăm funcția membru:

```
VarFisier.is_open();
```

Returnează 0 pentru fals, dacă fișierul nu s-a deschis și 1 pentru adevărat, dacă fișierul s-a deschis.

Închiderea unui flux înseamnă închiderea canalului prin care datele pot fi trimise încoace și încolo între program și fișier. Nu mai pot fi trimise date în nicio direcție folosind acel canal.

Închiderea fluxului nu înseamnă închiderea obiectului fluxului. Același flux poate fi folosit în continuare pentru a deschide un nou canal, care ar trebui să fie închis după utilizarea în transmiterea datelor. Facem un obicei din închiderea oricărui flux de fișiere, după ce acesta a fost deschis.

Atunci când un flux este închis, orice date din memorie care ar fi trebuit să fi fost în fișier sunt trimise la fișier înainte de a fi închis efectiv. Prototipul funcției membru pentru a închide *fstream* este:

```
VarFisier.close();
```

Deci, pentru a cunoaște dacă închiderea a avut succes, utilizăm funcția membru *is_open()*. Dacă închiderea fișierului a avut cu succes, aceasta ar returna 0, ceea ce înseamnă că fluxul nu mai este deschis. Dacă închiderea nu a reușit, ar returna 1 și înseamnă că fluxul nu ar putea fi închis.

Utilizăm funcția membru *get()*, punând caracterul citit în variabila *c*, care a fost deja declarată. Instrucțiunea *cout* trimite fiecare caracter la consolă.

```
char c;
while (!VarFisier.eof()) {
      VarFisier.get(c); cout<<c;
}
```

Exemplul 1: Elaborăm un program C++ pentru a crea un dosar în calculatorul personal, pe unitatea de stocare a datelor evidențiată prin partiția „*D:/*”. Denumirea dosarului va fi „*Exemple cu fisiere in C++*”. În acest dosar vom crea un fișier text cu denumirea „*FisierulMeu*”. În fișierul creat vom introduce următorul mesaj „*Programare C++*”.

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <direct.h>
```

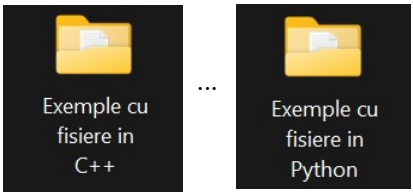
```
using namespace std;
int main(){
    //creare dosar
    if(mkdir("D:/Exemple cu fisiere in C++") == -1)
    cerr<<"Error: "<<strerror(errno)<<endl;
    else cout<<"Dosarul a fost creat cu succes!";
    //creare fisier text in dosar
    ofstream f("D:/Exemple cu fisiere in C++/FisierulMeu.txt");
    // scrierea informatiei in fisierul creat
    f<<"Programare C++"<<endl;
    f.close(); // inchidem fisierul creat
}
```

Următorul rând de cod: `cerr<<"Error: "<<strerror(errno)<<endl;` va afișa o eroare în cazul când dosaul este creatacest fișier deja este creat.

Pentru a redenumi un fișier text vom scrie următoarea instrucțiune după ce se va realiza închiderea fișierului care dorim să-l redenumim:

```
rename("D:/Exemple cu fisiere in C++/FisierulMeu.txt", "D:/Exemple cu fisiere in C++/FisierulMeuModificat.txt");
```

Pentru a redenumi un dosar sau dosarul în care se află fișierul nostru vom scrie următoarea instrucțiune după ce se va realiza închiderea fișierului din dosar, apoi vom redenumi:




```
rename("D:/Exemple cu fisiere in C++", "D:/Exemple cu fisiere in Python/");
```

Exemplul 2: Elaborăm un program C++ pentru a citi și a afișa la consolă informația care a fost introdusă într-un fișier de ieșire a datelor.

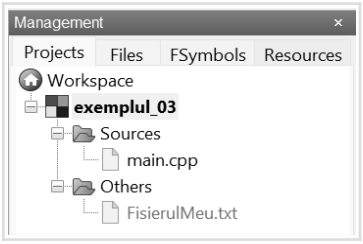
```
#include <iostream>
#include <fstream>
using namespace std;
```

```
int main(){
    ofstream f("D:/Exemple cu fisiere in C++/FisierulMeu.txt");
    f<<"Programare C++"<<endl;
    f.close();
    string s;
    // fisierul care a fost de iesire acum va fi de intrare
    ifstream g("D:/Exemple cu fisiere in C++/FisierulMeu.txt");
    // afisarea informatiei din fisier
    while (getline (g,s)){
          cout<<"Mesajul din fisier este: "<<s;
    }
    g.close(); // inchidem fisierul
}
```

Important

> ➢ Este important să reținem că vom folosi o buclă *while* împreună cu funcția *getline()*, care aparține clasei *ifstream*, pentru a citi datele din fișier linie cu linie și pentru a tipări conținutul fișierului.

Exemplul 3: Elaborăm un program C++ pentru a citi dintr-un fișier de pe prima linie dimensiunea n a unui vector de numere întregi, iar de pe a doua linie se vor citi cele n elemente. În același fișier să se afișeze separate prin spațiu suma tuturor elementelor din vector și media acestora.




```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    int a[10],n,i,suma=0;
    ifstream f("FisierulMeu.txt");
    // citim datele din fișier
    f>>n;
    for (i=0;i<n;i++){
```

```
        f>>a[i]; suma+=a[i];
    }
    // afisam datele din fișier la consola
    for (i=0;i<n;i++){
        cout<<a[i]<<" ";
    }
    f.close();
    // rescriem fișierul creat anterior cu rezultatele noastre
    ofstream g("FisierulMeu.txt");
    g<<suma<<" "<<suma*1.0/n;
    g.close();
}
```

Este important de precizat că datele anterioare din fișier se vor pierde după ce va fi rescris fișierul. Pentru ca datele să nu fie șterse după rescriere, vom realiza o modificare într-o linie de program, astfel încât datele inițiale să nu fie șterse, iar rezultatele să se fixeze în fișier pe linia a 3-a.

```
ofstream g("FisierulMeu.txt",ios::app);
```

Exemplul 4: Elaborăm un program C++ pentru a citi dintr-un fișier de pe prima linie valorile unui interval [a, b]. În același fișier să se afișeze separate prin spațiu toate numerele: pare, impare, pătrate perfecte, cuburi perfecte și prime.

```
#include <iostream>
#include <cmath>
#include <fstream>
using namespace std;
int main(){
    int a,b,i,j; bool p; // declaram variabilele necesare
    ifstream f("FisierulMeu.txt");
    // citim datele din fișier, apoi inchidem fisierul
    f>>a>>b; f.close();
    ofstream g("FisierulMeu.txt",ios::app);
    // verificam conditia pentru numerele pare
    g<<"\nNumerele pare sunt: "<<endl;
    for (i=a;i<=b;i++){
        if (i%2==0) g<<i<<" ";
```

```
    }
    // verificam conditia pentru numerele impare
    g<<"\n\nNumerele impare sunt: "<<endl;
    for (i=a;i<=b;i++){
        if (i%2!=0) g<<i<<" ";
    }
    // verificam conditia pentru numerele patrate perfecte
    g<<"\n\nNumerele patrate perfecte sunt: "<<endl;
    for (i=a;i<=b;i++){
        if (int(sqrt(i))==sqrt(i)) g<<i<<" ";
    }
    // verificam conditia pentru numerele cuburi perfecte
    g<<"\n\nNumerele cuburi perfecte sunt: "<<endl;
    for (i=a;i<=b;i++){
        if (int(cbrt(i))==cbrt(i)) g<<i<<" ";
    }
    // verificam conditia pentru numerele prime
    g<<"\n\nNumerele prime sunt: "<<endl;
    for (i=a;i<=b;i++){
        if (i==0 || i==1) continue;
        p=1;
        for (j=2; j<=i/2; ++j){
            if (i%j == 0){
                p=0; break;
            }
        }
        if (p==1) g<<i<<" ";
    }
    g.close(); // inchidem fisierul
}
```

3.4 Modele de itemi pentru exersare

(1) Elaborați un program C++ care va citi dintr-un fișier de pe prima linie dimensiunea *n* a unui vector de numere întregi, iar de pe linia imediat următoare va citi cele *n* elemente ale vectorului și le va afișa la consolă.

(2) Elaborați un program C++ care va citi dintr-un fișier de pe prima linie dimensiunea *n* a unui vector de numere întregi, iar de pe linia imediat următoare va citi cele *n* elemente ale vectorului și va afișa în același fișier pe liniile următoare suma și produsul elementelor din vector.

(3) Elaborați un program C++ care va citi dintr-un fișier de pe prima linie dimensiunile *m* și *n* ale unei matrici de numere întregi, iar de pe următoarele *m* linii și *n* coloane se vor citi datele matricii și le va afișa la consolă sub formă tabelară.

(4) Elaborați un program C++ care va citi dintr-un fișier de pe prima linie dimensiunea *n* a unei matrici pătratice de numere întregi, iar de pe următoarele *n* linii și *n* coloane se vor citi datele matricii și se va afișa la consolă sub formă de vector, în linii separate, toate elementele de pe diagonala principală și toate de pe diagonala secundară.

(5) Elaborați un program C++ care va citi dintr-un fișier de pe prima linie dimensiunea *n* a unei matrici pătratice de numere întregi, iar de pe următoarele *n* linii și *n* coloane se vor citi datele matricii și se va afișa la consolă sub formă de vector, toate elementele prime din matrice.

(6) Elaborați un program C++ care va citi dintr-un fișier de pe prima linie dimensiunea *n* a unei matrici pătratice de caractere, iar de pe următoarele *n* linii și *n* coloane se vor citi datele matricii și se va afișa la consolă de pe fiecare rând, numărul de vocale și numărul de consoane din matrice.

3.5 Metode directe de sortare a datelor

În această temă vom analiza următoarele unități de conținut:

- *identificarea principiului de sortare a datelor conform fiecărui algoritm direct;*
- *prelucrarea algoritmilor direcți de sortare a datelor conform specificațiilor propuse;*
- *implementarea algorimilor direcți de sortare a datelor în limbajul C++.*

Abordarea mea de a învăța mi-a făcut să am unele probleme cu algoritmii de sortare. Principalul meu argument a fost că nu păreau foarte practici. Cu unii din profesorii mei de la universitate am petrecut mult timp analizând diverși algoritmi de sortare și tot ce mă puteam gândi a fost că: aș putea obține același rezultat apelând pur și simplu metoda *sort()*. Ceea ce ar necesita doar o singură linie de cod.

Deci, de ce sunt importanți algoritmii de sortare? Și de ce sunt cele mai consecvente întrebări puse în timpul interviurilor pentru angajarea programatorilor? Motivul real este subtil și poate fi necesar să-l descoperiți singuri. Reacția voastră la acestă descoperire poate fi aceeași cu a mea când am participat la prima mea lecție din compartimentul algoritmilor de sortare.

Pe parcursul studierii algoritmilor de sortare, veți descoperi că pe lângă combinarea mai multor componente ale limbajului de programere, algoritmii de sortare vă ajută să înțelegeți exactitatea și viteza de execuție a programului. Algoritmii de sortare oferă o modalitate abstractă de a studia acuratețea programului.

În cele din urmă, nu puteți avea o discuție despre algoritmii de sortare fără a menționa performanța. Această componentă este vitală pentru a înțelege de ce este important să înțelegeți algoritmii de sortare și modul în care studiul vă va ajuta să vă îmbunătățiți abilitățile ca dezvoltator.

Să ne gândim la primele zile ale Twitter. Funcționalitatea de bază de trimitere a unui Tweet și de vizualizare a tweet-urilor altor utilizatori este aceeași astăzi ca și când au fost lansate. Cu toate acestea, dacă vă amintiți primele zile ale aplicației, veți ști că site-ul a scăzut în mod constant.

Au existat mai multe motive pentru performanța slabă a Twitter de la început. Cu toate acestea, principala problemă a fiind dezvoltarea aplicației fără a pune accentul pe scalabilitate. Acest lucru însemna că a funcționat perfect pentru un număr mic de utilizatori, cu toate acestea, cu zeci de milioane de utilizatori și Tweeturi, sistemul se bloca de zeci de ori pe zi. Povestea Twitter mi-a amintit de diferența dintre algoritmi.

Sunt toți algoritmii de sortare creați egali? Nici măcar pe aproape. De exemplu, dacă utilizați algoritmul *Bubble Sort* pe o colecție de 300.000 de numere întregi, va dura peste 9 minute. Cu toate acestea, dacă utilizați algoritmul *Quicksort*, acesta va sorta aceeași colecție de 300.000 de numere întregi în mai puțin de o secundă. Acest tip de variație de performanță poate fi înțeles doar atunci când ajungeți sub capotă și înțelegi cu adevărat cum funcționează algoritmii de sortare.

3.5.1 Algoritmul de sortare cu bule (BubbleSort)

Algotitmul de sortare cu bule este cel mai simplu algoritm care funcționează prin schimbarea în mod repetat a elementelor adiacente dacă acestea sunt în ordinea greșită. Acest algoritm nu este potrivit pentru seturi mari de date, deoarece complexitatea sa medie și în cel mai rău caz este destul de mare.

Date de intrare	Sortare ascendentă	Sortare descendentă
`5 1 4 2 8`	`1 2 4 5 8`	`8 5 4 2 1`

Implementarea algoritmului

(1) Sortarea cu bule începe cu primele două elemente, comparându-le pentru a verifica care dintre ele este mai mare.

- `(5 1 4 2 8)->(1 5 4 2 8)`, aici algoritmul compară primele două elemente și le interschimbă, deoarece `5 > 1`.
- `(1 5 4 2 8)->(1 4 5 2 8)`, interschimbăm `5 cu 4;`
- `(1 4 5 2 8)->(1 4 2 5 8)`, interschimbăm `5 cu 2;`
- `(1 4 2 5 8)->(1 4 2 5 8)`, deoarece aceste elemente sunt deja în ordine,adică `8 > 5`, algoritmul nu le schimbă.

(2) Acum, în timpul celei de-a doua iterații, ar trebui să arate astfel:

- **(1 4** 2 5 8)->**(1 4** 2 5 8)
- (1 **4 2** 5 8)->(1 **2 4** 5 8), interschimbăm 4 cu 2;
- (1 2 **4 5** 8)->(1 2 **4 5** 8)
- (1 2 4 **5 8**)->(1 2 4 **5 8**)

(3) Acum, vectorul este deja sortat, dar algoritmul nostru nu știe dacă este finalizat. Algoritmul are nevoie de o trecere întreagă fără nici o schimbare pentru a confirma că este sortat.

- **(1 2** 4 5 8)->**(1 2** 4 5 8)
- (1 **2 4** 5 8)->(1 **2 4** 5 8)
- (1 2 **4 5** 8)->(1 2 **4 5** 8)
- (1 2 4 **5 8**)->(1 2 4 **5 8**)

Secvența de cod rulează întotdeauna cu complexitatea de timp O (n^2), chiar dacă elementele vectorului este sortat. Mai jos este prezentată secvența de cod pentru algoritmul standard și pentru algoritmul optimizat în ordine ascendentă:

```
for (i=0;i<n-1;i++){
    for (j=0;j<n-i-1;j++){
        if (a[j]>a[j+1]){     // sortare ascendenta
           swap(a[j],a[j+1]); // interschimbarea elementelor
        }
    }
}
```

```
bool schimb;
for (i=0; i<n-1; i++){
    schimb = false;
    for (j=0; j<n-i-1; j++){
        if (a[j]>a[j+1]){     // sortare ascendenta
            swap(a[j],a[j+1]); schimb=true;
        }
    }
    if (schimb==false) break; // conditie de oprire
}
```

3.5.2 Algoritmul de sortare prin selecţie (SelectionSort)

Algoritmul de sortare prin selecție sortează un vectorastfel:
- pentru algoritmul de sortare ascendentă găsind în mod repetat elementul minim din partea nesortată și-l plasăm la început;
- pentru algoritmul de sortare descendentă găsind în mod repetat elementul maxim din partea nesortată și-l plasăm la început.

Algoritmul menține doi subvectori într-un vector de elemente inițializat:
- subvectorul care deja este sortat;
- subvectorul rămas, care nu a fost sortat.

În fiecare iterație a sortării prin selecție, elementul minim (luând în considerare ordinea crescătoare) din subvectorul nesortat este ales și mutat în subvectorul sortat.

Date de intrare	Sortare ascendentă	Sortare descendentă
64 25 12 22 11	11 12 22 25 64	64 25 22 12 11

Implementarea algoritmului

(1) Pentru prima poziție din vectorul sortat, întregul vector este parcurs de la indexul 0 la 4 în mod secvenţial. Prima poziție în care 64 este stocat în prezent, după parcurgerea întregului vector se observă că 11 este cea mai mică valoare. Astfel, înlocuiți 64 cu 11. După o iterație 11, care se este cea mai mică valoare din vector, tinde să apară în prima poziție a vectorului sortat.

```
•   (64 25 12 22 11)->(11 25 12 22 64)
```

(2) Pentru a doua poziție, unde este prezent 25, traversăm din nou restul vectorului într-un mod secvențial. După parcurgere, am constatat că 12 este a doua cea mai mică valoare din vector și ar trebui să apară pe poziția a doua în vector, așadar trebuie să interschimbăm aceste valori.

```
•   (11 25 12 22 64)->(11 12 25 22 64)
```

(3) Acum, pentru poziția a treia, unde este prezent 25, traversăm din nou restul vectorului și găsim a treia cea mai mică valoare prezentă. În timpul traversării, observăm că 22 este a treia cea mai mică valoare și ar trebui să apară pe poziția a treia în vector, astfel interschimbăm 22 cu elementul prezent la a treia poziție.

- (11 12 **25 22** 64)->(11 12 **22 25** 64)

(4) În mod similar, pentru a patra poziție, traversăm restul vectorului și găsim al patrulea cel mai mic element. Deoarece 25 este a patra cea mai mică valoare, se va plasa pe a patra poziție.

- (11 12 22 **25** 64)->(11 12 22 **25** 64)

(5) În cele din urmă, cea mai mare valoare prezentă în vector este plasată automat în ultima poziție. Vectorul rezultat este vectorul cu elementele sortate ascendent.

- (11 12 22 25 **64**)->(11 12 22 25 **64**)

Secvența de cod rulează întotdeauna cu complexitatea de timp O (n^2), chiar dacă elementele vectorului este sortat. Mai jos este prezentată secvența de cod pentru algoritmul standard în ordine ascendentă:

```
int minim;
for(i=0;i<n-1;i++){
    minim=i;
    for (j=i+1;j<n;j++){
        if (a[j]<a[minim]) minim=j;
    }
    if(minim!=i) swap(a[minim],a[i]);
}
```

3.5.3 Algoritmul de sortare prin inserţie (InsertionSort)

Sortare prin inserție este un algoritm simplu care funcționează similar modului în care sortăm cărțile de joc în mâinile noastre. Vectorul este practic împărțit într-o parte sortată și una nesortată. Valorile din partea nesortată sunt selectate și plasate în poziția corectă în partea sortată.

Practic, sortarea prin inserție este eficientă pentru valori mici de date. Acest algoritm este de natură adaptivă, adică este adecvată pentru seturile de date care sunt deja sortate parțial.

Date de intrare	Sortare ascendentă	Sortare descendentă
12 11 13 5 6	5 6 11 12 13	13 12 11 6 5

Implementarea algoritmului

(1) Inițial, primele două elemente ale vectorului sunt comparate cu ajutorul algoritmului prin inserție. Aici observăm că 12 este mai mare decât 11, prin urmare, nu sunt în ordine crescătoare și 12 nu este în poziția corectă. Astfel, schimbați 11 și 12. Deci, deocamdată 11 este stocat într-un subvector sortat.

- (**12 11** 13 5 6)->(**11 12** 13 5 6)

(2) Acum, trecem la următoarele două elemente și le comparăm. Aici observăm că 13 este mai mare decât 12, astfel încât ambele elemente par să fie în ordine crescătoare, prin urmare, nu va avea loc nicio schimbare. Elementul 12 este stocat într-un subvector sortată împreună cu 11.

- (11 **12 13** 5 6)->(11 **12 13** 5 6)

(3) Acum, în subvectorul sortat sunt prezente două elemente, care sunt 11 și 12. Trecând înainte la următoarele două elemente, care sunt 13 și 5. Atât elementul 5, cât și 13 nu sunt prezente la locul lor corect, așa că le schimbăm. După schimbare, elementele 12 și 5 nu sunt sortate, deci schimbați din nou. După aceasta, din nou elementele 11 și 5 nu sunt sortate, deci schimbăm din nou. Astfel primele 4 elemente sunt plasate în poziția corectă a lor din subvectorul sortat crescător.

- (11 12 **13 5** 6)->(11 12 **5 13** 6)
- (11 **12 5** 13 6)->(11 **5 12** 13 6)
- (**11 5** 12 13 6)->(**5 11** 12 13 6)

(4) Acum, elementele care sunt prezente în subvectorul sortat sunt: 5, 11 și 12. Trecând la următoarele două elemente, 13 și 6, în mod clar observăm că ele nu sunt sortate, deci efectuăm schimbul între ambele nume-

re. Acum observăm că 6 este mai mic decât 12, prin urmare, schimbă din nou. După aceasta, de asemenea, schimbăm 11 și 6, pentru că nu sunt elemente sortate corect, prin urmare, schimbăm din nou. În cele din urmă, vectorul este complet sortat.

- (5 11 12 **13 6**)->(5 11 12 **6 13**)
- (5 11 **12 6** 13)->(5 11 **6 12** 13)
- (5 **11 6** 12 13)->(5 **6 11** 12 13)

Secvența de cod rulează întotdeauna cu complexitatea de timp O (n^2), chiar dacă elementele vectorului este sortat. Mai jos este prezentată secvența de cod pentru algoritmul standard în ordine ascendentă:

```
int pozitie;
for(i=1;i<n;i++){
    pozitie=a[i];
    for (j=i-1;j>=0 && a[j]>pozitie;j=j-1){
        a[j+1]=a[j];
    }
    a[j+1]=pozitie;
}
```

3.5.4 Algoritmul de sortare prin numărare (CountingSort)

Sortarea prin numărare este o tehnică de sortare bazată pe chei între un anumit interval. Funcționează prin numărarea numărului de elemente care au valori cheie distincte. Apoi efectuăm unele operații aritmetice pentru a calcula poziția fiecărui element în secvența de ieșire.

Sortarea prin numărare face ipoteze despre date, de exemplu, presupune că valorile vor fi în intervalul de la 0 la 10 sau 10 – 99, etc. Alte ipoteze pe care le face algoritmul sunt datele de intrare vare vor fi toate numere reale. Acest algoritm de sortare nu este bazat pe comparație.

Date de intrare	Sortare ascendentă	Sortare descendentă
4 2 2 8 3 3 1	1 2 2 3 3 4 8	8 4 3 3 2 2 1

Implementarea algoritmului

(1) Aflăm elementul maxim din vectorul de date.

4	2	2	8	3	3	1
			maxim			

(2) Inițializăm un vector de lungime *maxim+1* cu toate elementele 0. Acest vector este folosit pentru stocarea numărului elementelor din vectorul inițial de date.

Indice:	0	1	2	3	4	5	6	7	8
Numărul de repetări:	0	0	0	0	0	0	0	0	0

(3) Stocăm numărul fiecărui element la indexul respectiv în tabloul de numărare (repetări). De exemplu: dacă numărul de repetări ale elementului „3” este 2, atunci numărul 2 este stocat în a treia poziție a vectorului de numărare (repetări). Dacă elementul „5” nu este prezent în matrice, atunci 0 este stocat în a 5-a poziție.

Indice:	0	1	2	3	4	5	6	7	8
Numărul de repetări:	0	1	2	2	1	0	0	0	1

(4) Stocăm suma cumulativă a elementelor vectorului de numărare, aceasta ajută la plasarea elementelor în indicele corect al vectorului sortat.

Indice:	0	1	2	3	4	5	6	7	8
Numărul de repetări:	0	1	3	5	6	6	6	6	7

(5) Găsim indicele fiecărui element al vectorului inițial din vectorul de numărare. Aceasta oferă numărul cumulat. După ce am plasat fiecare element în poziția corectă, îi micșorăm numărul de repetări cu o unitate. De exemplu: elementul „4” are numrul de repetări egal cu 6, prin urmare indicele elementului „4” în vectorul sortat ascendent va fi 6-1=5.

4	2	2	8	3	3	1

Secvența de cod rulează întotdeauna cu complexitatea de timp O (n+k), unde *n* este numărul de elemente din vectorul de intrare și *k* este domeniul de intrare. Urmează prezentarea secvenței de cod pentru algoritmul standard în ordinea ascendentă:

```
int nr[10],b[10], x=a[0];
for (i=1; i<n; i++) {
    if (a[i]>x) x=a[i];
}
for (i=0; i<=x; ++i) {
    nr[i]=0;
}
for (i=0; i<n; i++) {
    nr[a[i]]++;
}
for (i=1; i<=x; i++) {
    nr[i]+=nr[i-1];
}
for (i=n-1; i>=0; i--) {
    b[nr[a[i]]-1]=a[i]; nr[a[i]]--;
}
for (i=0; i<n; i++) {
    a[i]=b[i];
}
```

Algoritmul de sortarea prin numărare este utilizat atunci când:

- există numere întregi mai mici cu numărări multiple (repetări).
- este necesară o complexitate liniară a algoritmului.

3.5 Modele de itemi pentru exersare

(1) Elaborați un program C++ care va genera un vector de numere pare de trei cifre ce conține *n* elemente, numărul *n* se va introduce de la tastatură. Aplicați algoritmul BubbleSort pentru soratarea ascendentă și descendentă a elementelor acestui vector.

(2) Elaborați un program C++ care va genera o matrice pătratică de numere impare de două cifre ce conține *n*n* elemente, numărul *n* se va introduce de la tastatură. Aplicați algoritmul SelectionSort pentru soratarea ascendentă a elementelor de pe prima linie și sortarea descendentă a elementelor de pe ultima linie din această matrice.

(3) Elaborați un program C++ care va genera un vector de numere întregi pozitive ce conține *n* elemente, numărul *n* se va genera din intervalul [20, 50]. Aplicați algoritmul InsertionSort pentru soratarea elementele acestui vector și afișați toate perechile de numere care au în sumă un număr natural *s*, acesta se va introduce de la tastatură.

Date de intrare	Date de ieșire
n=6; A[]={8, 7, 1, 2, 3, 5}; s=10.	(8 2); (7 3);

(4) Elaborați un program C++ care va genera un vector de numere întregi din intervalul [0, 2], ce conține *n* elemente, numărul *n* se va genera din intervalul [20, 50]. Aplicați algoritmul CountingSort pentru a rezolva problema drapelului național olandez.

Date de intrare	Date de ieșire
n=12; A[]={0,1,2,2,1,0,0,2,0,1,1,0};	0 0 0 0 0 1 1 1 1 2 2 2

3.6 Structuri. Tablou de structuri

În această temă vom analiza următoarele unități de conținut:
- *declararea și inițializarea variabilelor de tip structură / tablou de structuri;*
- *identificarea instrucțiunilor de citire, afișare și accesare a datelor unei structuri / tablou de structuri;*
- *prelucrarea structurilor, uniunilor și enumerărilor conform specificațiilor propuse;*
- *implementarea algorimilor de prelucrare a structurilor / tablourilor de structuri în limbajul C++.*

Structura este o colecție de variabile de diferite tipuri de date sub un singur nume. Este similar cu o clasă prin faptul că ambele dețin o colecție de informații de diferite tipuri de date. Structurile din C++ pot conține două tipuri de membri:

- *membru de date*: acești membri sunt variabile normale C++. Putem crea o structură cu variabile de diferite tipuri de date în C++.
- *funcții membre*: acești membri sunt funcții normale C++. Alături de variabile, putem include și funcții în interiorul unei declarații de structură.

De exemplu: dorim să stocăm câteva informații despre o *persoană*: numele, prenumele, numărul de identitate și salariul. Putem crea cu ușurință diferite variabile: *nume*, *prenume*, *idnp* și *salariu* pentru a stoca aceste informații separat. Cu toate acestea, dacă în viitor am dori să stocăm informații despre *N* persoane, atunci ar trebui să creăm variabile diferite pentru fiecare informație caracteristică pentru fiecare persoană: *nume1*, *prenume1*, *idnp1*, *salariu1*, *nume2*, *prenume2*, *idnp2*, *salariu2*, . . . , *numeN*, *prenumeN*, *idnpN*, *salariuN*.

Putem observa cu ușurință cât de mare și dezordonat ar arăta codul. De asemenea, deoarece nu ar exista nicio relație între variabile (informații), va fi o sarcină descurajantă. O abordare mai bună va fi să avem o colecție din toate informațiile conexe sub un singur nume *Persoană* și să o folosim pentru fiecare persoană. Acum, codul pare mult mai simplu, lizibil și eficient.

Această colecție din toate informațiile conexe sub un singur nume *Persoană* este o structură.

3.6.1 Prezentarea generală a unei structuri.

Cuvântul cheie *struct* definește un tip de structură urmat de un identificator, numele structurii. Apoi, în interiorul acoladelor, putem declara unul sau mai mulți membri (declarăm variabile în interiorul acoladei) ai acelei structuri. Exemplu de structură conform celor menționate anterior:

```
struct persoana{
       char nume[20], prenume[30];
       long long int idnp;
       float salariu;
}
```

Aici structura *persoana* are definiți 4 membri: *nume*, *prenume*, *idnp* și *salariu*. Când se creează o structură, nu este alocată nicio memorie. Definiția structurii este doar modelul pentru crearea variabilelor. Odată ce declarăm o persoană de tip structură, putem defini o variabilă de structură astfel:

```
persoana profesor;
```

Aici este definită o variabilă *profesor* care este de tip structură, *persoana*. Când este definită variabila de tip structură, doar atunci memoria necesară este alocată de către compilator. Membrii variabilei structurii sunt accesați folosind un operator, punctul „ .”. Să presupunem că dorim să accesăm informația variabilei *profesor* cu cîmpul *salariu* și să îi atribuim valoarea 5000. Putem efectua această sarcină utilizând următorul cod:

```
profesor.salariu = 5000;
```

Indiferent de ce tip de sistem avem (pe 32 de biți sau pe 64 de biți), memoria tipului *float* este de 4 octeți, memoria tipului *long long int* este de 8 octeți și memoria de tip *char* este de 1 octet. Prin urmare, 1*20 + 1*30 + 8 + 4 = 62 de octeți de memorie sunt alocați pentru variabila *profesor* de tip structură.

Să creăm secvența de cod pentru a inițializa o structură corespunzătoare următoarelor condiții specifice:

a) automobil cu câmpurile: denumire, model, anul producerii, capacitate rezervor, cai putere;
b) student cu câmpurile: nume, prenume, bacMatematica, bacBiologia, bacRomana, bacEngleza.

Structura automobil	Structura elev
`struct automobil{` `  char denumire[20];` `  char model[20];` `  short int anul, cp;` `  float rezervor;` `}`	`struct elev{` `  char nume[20];` `  char prenume[20];` `  short int bacM, bacB;` `  short int bacR,bacE;` `}`
Memorie alocată: *48 octeți*	Memorie alocată: *48 octeți*

Să elaborăm un program în limbajul C++ care va citi de la tastatură date despre o persoană și apoi le va afișa la consolă. Structura persoana va avea câmpurile: numele, vârsta și bursa.

```
#include <iostream>
using namespace std;
// initializarea structurii
struct Persoana {
    char nume[50]; int varsta; float bursa;
};
int main() {
    Persoana p;
    cout<<"Introduceti numele: "; cin.get(p.nume, 50);
    cout<<"Introduceti varsta: ";cin>>p.varsta;
    cout<<"Introduceti bursa (cu zecimale): "; cin>>p.bursa;
    system("CLS");
    cout<<"Afisarea informatiei despre elev: "<<endl;
    cout<<"Nume: "<<p.nume<<endl;
    cout<<"Varsata: "<<p.varsta<<endl;
    cout<<"Bursa: "<<p.bursa<<endl;
}
```

Să modificăm programul anterior astfel încât să se citească de la tastatură datele a *n* persoane și să afișăm la consolă elevii cu bursa mai mare de 500. Pentru rezolvarea acestei probleme vom aplica un tablou de structuri.

```
#include <iostream>
using namespace std;
// initializarea structurii persoana
struct persoana {
    char nume[50]; int varsta; float bursa;
};
persoana p[100]; // declararea variabilei p de tip persoana
int main(){
    int n,i;
    cout<<"Introduceti numarul de persoane: "; cin>>n;
    // introducem datele despre persoanele din grup
    for (i=1; i<=n; i++){
        cout<<"Date despre persoana "<<i<<": "<<endl;
        cout<<"\tIntroduceti numele: "; cin>>p[i].nume;
        cout<<"\tIntroduceti varsta: "; cin>>p[i].varsta;
        cout<<"\tIntroduceti bursa (cu zecimale): ";
        cin>>p[i].bursa;
    }
    system("CLS");
    cout<<"Elevii ce au bursa mai mare de 500: "<<endl;
    for (i=1; i<=n && p[i].bursa>500; i++){
        cout<<"Date despre persoana "<<i<<": "<<endl;
        cout<<"\tNume: \t"<<p[i].nume<<endl;
        cout<<"\tVarsata: \t"<<p[i].varsta<<endl;
        cout<<"\tBursa: \t"<<p[i].bursa<<endl;
    }
}
```

3.6.2 Structuri imbricate.

Uneori într-o structură putem avea câmpuri de alt tip de structură sau mai simplu spus, putem avea cazuri când o structură apelează o altă structură deja inițializată. Exemplu:

```
struct Data{
    int zi, luna, an;
```

```
};
struct Persoana{
    char nume[21], prenume[21], sex;
    int varsta;
    Data data_angajarii;
    double salariu;
};
Persoana A;
```

Am declarat două tipuri de date: *Data* – pentru a memora o dată calendaristică și *Persoana* – pentru a memora informații despre o persoană și o variabilă *A* de tip *Persoana*. Variabila *A* are un câmp de tip *Data*, numit *data angajarii*. Prin intermediul acestuia avem acces la *anul*, *luna* și *ziua* angajării persoanei respective.

Să elaborăm un program în limbajul C++ care va citi datele despre angajați dintr-un fișier și va afișa la consolă lista angajaților interprinderii ce au salariul mai mare de 5000.

```
#include <iostream>
#include <fstream>
using namespace std;
struct Data{
    int zi, luna, an;
};
struct Persoana{
    char nume[21], prenume[21];
    Data data_angajarii;
    double salariu;
};
Persoana p[20];
ifstream fin("angajat.txt");
int main(){
    int n,i;
    cout<<"Introduceti numarul de persoane: "; fin>>n;
    // introducem datele din fisier
    for (i=1; i<=n; i++){
        fin>>p[i].nume>>p[i].prenume>>p[i].salariu;
        fin>>p[i].data_angajarii.zi;
        fin>>p[i].data_angajarii.luna>>p[i].data_angajarii.an;
    }
```

```
    // afisam datele conform conditiei
    system("CLS");
    cout<<"Persoanele cu salariul mai mare de 5000: "<<endl;
    for (i=1; i<=n; i++){
        if (p[i].salariu>5000){
            cout<<"\nDate despre persoana "<<i<<": "<<endl;
            cout<<"\tNume / Prenume: \t"<<p[i].nume;
            cout<<" "<<p[i].prenume<<endl;
            cout<<"\tData angajarii: \t";
            cout<<p[i].data_angajarii.zi<<".";
            cout<<p[i].data_angajarii.luna<<".";
            cout<<p[i].data_angajarii.an<<endl;
        }
    }
}
```

Datele propuse pentru testare din fișierul *angajat.text* sunt următoarele:

```
7
Albu Oleg 9000.00 01 09 2020
Bozu Alex 8000.00 01 09 2020
Cace Cati 7000.00 01 09 2021
Dabi Ella 6000.00 01 09 2021
Eana Emma 5000.00 01 09 2022
Feca Danu 4500.00 01 09 2022
Gaea Danu 4000.00 01 09 2022
```

3.6.3 Uniuni şi enumerări

Union este un tip de date definit de utilizator și toți membrii diferiți ai uniunii au aceeași locație de memorie. Membrul uniunii care ocupă cea mai mare memorie decide mărimea uniunii. Uniunea este folosită mai ales atunci când utilizatorul caută să folosească o singură locație de memorie pentru diferiți membri multipli.

Uniunile sunt foarte asemănătoare cu structurile. Se definesc în același mod, pe măsură ce se fac structurile. Pentru definirea uniunilor, trebuie să folosim cuvântul cheie „union” în limbajul C++.

```
union numeUniune{
      tip1 membru1;
      tip2 membru2;
      . . .
      tipN membruN;
}
```

```
struct numeStructura{
      tip1 membru1;
      tip2 membru2;
      . . .
      tipN membruN;
}
```

Uniunile sunt folosite în diferite situații în care codificatorul dorește să folosească aceeași memorie pentru diferiți membri multipli. De exemplu, dacă trebuie să creăm o structură de date arbore binar în care fiecare nod frunză va avea o valoare de date dublă și fiecare nod intern are doi pointeri, dar nu există date împreună cu ei.

Dacă creăm structuri pentru a implementa acest lucru, atunci fiecare nod va avea nevoie de 16 octeți, unde jumătate din octeți vor fi irosiți pentru fiecare nod. Dacă vom declara toate nodurile folosind *union*, atunci putem economisi o cantitate bună de spațiu. Acesta este modul în care putem folosi *union* pentru a economisi spațiu, în loc să folosim *struct*. Acesta este motivul principal pentru care uniunile sunt foarte asemănătoare cu structurile, dar uniunea are un avantaj foarte bun.

```
union {
     int no;
     char nume[20];
     float pret;
};
```

În această uniune, dimensiunea membrului *nume* este mai mare decât dimensiunea altor membri, astfel încât dimensiunea unei uniuni în memorie va fi de 20 de octeți.

Când definim o uniune fără nici un nume, ea se numește *uniune anonimă*. În asemenea caz, membrii uniunii sunt accesați direct după numele lor. Rețineți că acest tip de uniune este declarat local în interiorul funcției principale, *main()* .

Analizați următoarea secvență de cod C++ care implementează uniunea *punct.* Copiați secvența de cod în mediul de programare și stabiliți pentru cele 3 cazuri ce se va afișa la consolă. Cum explicați rezultatele execuției?

```
#include <iostream>
using namespace std;
union punct {
    int x,y;
```

```
};
int main(){
    union punct p;
    p.x=2;
    cout<<"Coordonatele: x= "<<p.x<<" si y= "<<p.y<<endl;
    // pentru cazul 1 se va afișa x= 2 și y= 2
    p.y=5;
    cout<<"Coordonatele: x= "<<p.x<<" si y= "<<p.y<<endl;
    // pentru cazul 2 se va afișa x= 5 și y= 5
    p.x=7; p.y=9;
    cout<<"Coordonatele: x= "<<p.x<<" si y= "<<p.y<<endl;
    // pentru cazul 3 se va afișa x= 9 și y= 9
}
```

În C++ enumerarea (*enum*), este un tip de date format din valori numite elemente, membri, etc., care reprezintă constante integrale. Enumerarea oferă o modalitate de a defini și grupa constantele integrale. De asemenea, face codul ușor de întreținut și mai puțin complex.

Enumerările sau enumerațiile sunt în general folosite atunci când ne așteptăm ca variabila să selecteze o valoare din setul posibil de valori. Acestea măresc abstracția și ne permit să ne concentrăm mai mult pe valori, decât să ne facem griji despre cum să le stocăm. De asemenea, este util pentru documentarea codului și în scopuri de lizibilitate.

Pentru a defini tipul enumerare în C++, trebuie să utilizați cuvântul cheie *enum* împreună cu elementele separate prin virgule. Sintaxa de bază a tipului respectiv de date este:

```
enum numeEnumerare{ element1, element2, element3, …, elementN }
```

Exemplu de aplicare a unei enumerări:

```
#include <iostream>
using namespace std;
enum Anotimpuri {Primavara, Vara, Toamna, Iarna} a;
int main(){
   a = Primavara; cout<<"Pozitia anotimpului este "<<a<<endl;
   a = Toamna; cout<<"Pozitia anotimpului este "<<a<<endl;
}
```

3.6.4 Modele de exerciţii teoretice rezolvate

1. Se consideră următoarele declaraţii, care este tipul expresiei x.x.y?

```
struct A{ int x; char y;
};
struct B{ float x; long y;
};
struct C{ struct A x; struct B y;
} x, y;
```

(a) long; (b) int; (c) char; (d) float.

2. Se consideră următoarele declaraţii, care dintre următoarele referiri este corectă din punct de vedere sintactic?

```
struct punct{ float x,y;
};
struct cerc{ float raza; punct centru;
} c;
```

(a) c.punct.y;
(b) c.raza.punct;
(c) c.centru.x;
(d) c.y.centru.

3. Se consideră următoarele declaraţii, în care variabila *e* memorează date despre un anumit elev. Care este instrucţiunea C++ prin care se iniţializează anul naşterii acestui elev cu valoarea 2010.

```
struct data{ int zi,luna,an;
};
struct elev { char nume[30]; data data_nasterii; float media;
} e;
```

(a) e.data_nasterii = 2010;
(b) e.data = 2010;
(c) e.data_nasterii.an = 2010;
(d) e.data.an = 2010.

La fiecare din itemii anteriori răspunsul corect este prezentat în varianta (c).

3.6 Modele de itemi pentru exersare

(1) Elaborați structura *examen* ce conține următoarele câmpuri: *notaEx1*, *notaEx2*, *notaEx3* și *mediaEx*. Declarați o variabilă care va prelua toate câmpurile structurii create.

(2) Elaborați structura *teza* ce conține următoarele câmpuri: *nume*, *prenume*, *clasa, notaTeza*. Declarați o variabilă care va prelua toate câmpurile structurii create pentru a afișa media aritmetică a notelor grupului de elevi.

(3) Elaborați structura *automobil* ce conține următoarele câmpuri: *denumire*, *model*, *motor, pretul* și *culoare*. Structura *motor* poate conține următoarele câmpuri: *serie*, *nrCilindri*, *volumul, caiPutere* și *combustibil*. Declarați o variabilă care va prelua toate câmpurile structurii create pentru a afișa informația grupului de automobile.

(4) Elaborați structura *elev* ce conține următoarele câmpuri: *nume*, *prenume*, *clasa, adresa* și *parinti*. Structura *adresa* poate conține următoarele câmpuri: *oras*, *strada*, *nrStrada, nrBloc* și *nrApartament*. Structura *parinti* poate conține următoarele câmpuri: *prenumeTata* și *prenumeMama*. Declarați o variabilă care va prelua toate câmpurile structurii create pentru a afișa informația grupului de elevi.

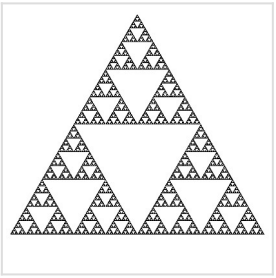
(5) Elaborați structura *chitanta* ce conține următoarele câmpuri: *enrgiaElectrică*, *gazNatural*, *apaPotabila* și *data*. Structura *data* reprezintă ultima zi și lună din anul curent când au fost transmise datele de la fiecare contor și poate conține următoarele câmpuri: *ziua*, *luna* și *anul*. Declarați o variabilă care va prelua toate câmpurile structurii create. Fiecare din serviciile comunale prezentate are o anumită dată când a transmis informația de la contor.

Capitolul 4

SUBPROGRAME ŞI MODULE

Introducere

O funcție în C++ este un set de instrucțiuni combinate care îndeplinesc o anumită sarcină. Corpul funcției este executat numai atunci când apelăm funcția. Crearea unei funcții crește reutilizarea și modularitatea în program. Întregul cod poate fi împărțit logic în funcții separate în care fiecare funcție îndeplinește o anumită sarcină.



Recursiunea are un instrument de rezolvare a problemelor, în care împarte problemele mai mari în sarcini simple și lucrează individual pentru a urma o secvență individuală. Avantajele recursiunii:

(1) codul furnizat este curat și compact prin simplificarea programului complex, utilizează mai puține variabile în codul programului;

(2) codul complex și buclele for imbricate sunt evitate aici.

Concluzionăm că recursiunea ajută în C++ la rezolvarea problemelor în conceptele structurii de date, cum ar fi: traversările, sortarea și căutarea, și poate fi utilizată eficient oriunde este necesar.

Programele contemporane sunt rareori scrise fără biblioteci. Structura programului C++ oferă conceptul de fișiere antet pentru a ușura utilizarea anumitor blocuri de cod reutilizabile. Astfel, utilizatorii își pot crea propriile fișiere antet și le pot include în fișierele sursă după cum este necesar.

4.1 Noţiune de subprogram. Efecte colaterale

În această temă vom analiza următoarele unități de conținut:
- *declararea și apelarea subprogramelor cu tip și fără tip;*
- *comunicarea între subprograme prin intermediul variabilelor globale / parametrilor;*
- *evitarea efectelor colaterale;*
- *tratarea excepțiilor (try, throw și catch);*
- *implementarea subprogramelor cu tip și fără tip în C++.*

O funcție este un set de instrucțiuni care preiau date de intrare, efectuiază anumite calcule și produc rezultate. Ideea este să plasăm împreună unele sarcini efectuate în mod obișnuit sau repetat și să facem o funcție, astfel încât, în loc să scriem același cod din nou și din nou pentru date de intrare diferite, să putem apela funcția.

4.1.1 Subprograme predefinite. Subprograme definite de utilizator.

În termeni simpli, o funcție este un bloc de cod care rulează doar atunci când este apelată.

Sintaxă

```
tip numeFunctie ( tip1 nume1,tip2 nume2, ... , tipN numeN);
```

Să elaborăm o funcție care va determina elementul maxim dintre două numere întregi:

```
int maxim2 (int a, int b){
    if (a>b) return a;
    else return b;
}
```

O declarație de funcție îi spune compilatorului despre numărul de parametri, funcția preia tipurile de date ale parametrilor și returnează tipul funcției. Introducerea numelor de parametri în declarația funcției este opțională

în declarația funcției, dar este necesar să le introducem în definiție. Mai jos sunt un exemplu de declarații de funcție.

```
int max(int, int); // funcția care ia două numere întregi ca
parametri și returnează un număr întreg;

int* swap(int*, int); // functia care ia un pointer int și o
variabilă int ca parametri și returnează un pointer de tip int;

char* call(char b); // funcția care ia un caracter ca parametru
și returnează o variabilă de referință;

int fun(char, int); // funcția care ia un caracter și un int ca
parametri și returnează un număr întreg.
```

Funcțiile definite de utilizator sunt blocuri de cod definite de utilizator, special personalizate pentru a reduce complexitatea programelor mari. Ele sunt cunoscute ca „funcții personalizate”, fiind construite pentru a satisface condiția în care utilizatorul se confruntă cu dificultăți, reducând în același timp complexitatea întregului program.

Funcțiile bibliotecii sunt numite și „funcții încorporate”. Aceste funcții fac parte dintr-un pachet compilator care este deja definit și constă dintr-o funcție specială cu semnificații speciale și diferite. Funcția încorporată ne oferă un avantaj, deoarece le putem folosi direct fără a le defini, în timp ce în funcția definită de utilizator trebuie să o declarăm și să definim o funcție înainte de a o folosi. Exemple de funcții incorporate: *sqrt( )*, *setw( )*, etc.

Parametrii trecuți la funcție se numesc *parametri reali*. De exemplu, în programul de mai jos, 5 și 10 sunt parametri reali. Parametrii primiți de funcție se numesc *parametri formali*. De exemplu, în programul de mai sus, x și y sunt parametri formali.

```
#include <iostream>
using namespace std;
int maxim2 (int a, int b){
    if (a>b) return a;
    else return b;
}
int main(){
    int x=5,y=10;
```

```
    cout<<"Maxim("<<x<<","<<y<<") = "<<maxim2(x,y);
}
```

Există două modalități cele mai populare de a trece parametri:

(1) *Transmitere după valoare:* În această metodă de trecere a parametrilor, valorile parametrilor actuali sunt copiate în parametrii formali ai funcției și cele două tipuri de parametri sunt stocate în locații de memorie diferite. Deci, orice modificări efectuate în interiorul funcțiilor nu sunt reflectate în parametrii actuali ai apelantului.
(2) *Transmitere după referință:* Atât parametrii actuali, cât și cei formali se referă la aceleași locații, astfel încât orice modificări făcute în interiorul funcției se reflectă de fapt în parametrii actuali ai apelantului.

Tabelul 4.1: Apelul după valoare și apelul prin referință

Apelul după valoare	**Apelul după referință**
a) O copie a valorii este transmisă funcției. b) Modificările făcute în interiorul funcției nu sunt reflectate asupra altor funcții c) Argumentele reale și formale vor fi create în locație de memorie diferită	a) O adresă a valorii este transmisă funcției. b) Modificările efectuate în interiorul funcției sunt reflectate și în afara funcției. c) Argumentele reale și formale vor fi create în aceeași locație de memorie.

Puncte de reținut despre funcțiile din C++:
(1) Majoritatea programelor C++ au o funcție numită main() care este apelată de sistemul de operare atunci când un utilizator rulează programul.
(2) Fiecare funcție are un tip de returnare. Dacă o funcție nu returnează nicio valoare, atunci void este folosit ca tip de returnare. Mai mult decât atât, dacă tipul de returnare al funcției este nul, putem folosi în continuare instrucțiunea return în corpul definiției funcției fără a specifica nicio constantă, variabilă etc. împreună cu aceasta, menționând doar instrucțiunea „return;" care ar simbolizează terminarea funcției.
(3) Pentru a declara o funcție care poate fi apelată doar fără niciun parametru, ar trebui să folosim „void fun(void)". Ca o notă secundară, în C++,

o listă goală înseamnă că o funcție poate fi apelată doar fără niciun parametru. În C++, atât void fun() cât și void fun(void) sunt aceleași.

Să elaborăm o funcție care va determina elementul maxim dintre elementele unui vector de numere întregi:

```
void maxVector(int v[10]){
    m=v[0];
    for (i=0; i<n; i++) {
        if (m<v[i]) m=v[i];
    }
    cout<<"Elementul maxim din vector este: "<<m<<endl;
}
```

Să elaborăm o funcție care va determina elementul maxim dintre elementele unei matrici de numere întregi:

```
void maxMatrice(int v[4][3]){
    m=v[0][0];
    for (i=0; i<nLinii; i++) {
        for (j=0; j<nColoane; j++){
            if (m<v[i][j]) m=v[i][j];
        }
    }
    cout<<"Elementul maxim din matrice este: "<<m<<endl;
}
```

Să elaborăm o funcție care va determina numărul de cuvinte dintr-un enunț:

```
int nrCuvinte(char *enunt){
    bool st=0; // variabila de stare
    int num=0;
    while (*enunt){
        if (*enunt==' ' || *enunt=='\n' || *enunt=='\t') st=0;
        else if (st==0){
            st=1; ++num;
        }
        ++enunt;
    }
```

```
    cout<<"Numarul de cuvinte din enunt este: "<<num<<endl;
}
```

4.1.2 Efecte colaterale.

Prin efect colateral se înțelege o atribuire în corpul funcției a unei valori la o variabilă globală sau la un parametru formal, variabilă. Efectele colaterale pot influiența în mod neașteptat execuția unui program și complică procesele de depanare.

```
#include <iostream>
using namespace std;
int a; // variabila globala
int f(int x){
    a++; // atribuirea altereaza valoarea variabilei globale a
    return a*x;
} // functia f returneaza valoarea expresiei a*x
int main(){
    a=0;
    cout<<f(1)<<" "<<f(1)<<" "<<f(1)<<endl;
}
```

```
#include <iostream>
using namespace std;
int a; // variabila globala
int f(int x){
    a++; // atribuirea altereaza valoarea variabilei globale a
    return 2*x;
} // functia f returneaza valoarea expresiei 2*x
int main(){
    a=2;
    cout<<f(a)<<" "<<f(a)<<" "<<f(a)<<endl;
}
```

Pentru valori identice ale lui x, funcția f returnează valori diferite.

Efectele colaterale induc abateri de la procesul standard de comunicare, prin care variabilele participante sunt desemnate explicit ca parametri formali în declarație și parametri actuali în apel.

Consecințele efectelor colaterale se pot propaga în domeniul de vizibilitate al declarațiilor globale și pot interfera cu cele similare, produse la execuția altor subprograme. La elaborarea programelor se vor aplica următoarele recomandări:

- comunicarea subprogramelor cu mediul de chemare se va face prin transmiterea de date spre funcție prin parametri formali valoare și întoarcerea unui singur rezultat prin numele ei;
- comunicarea subprogramelor cu mediul de chemare se va face prin transmiterea de date prin parametri formali valoare sau variabilă și întoarcerea rezultatelor prin parametri formali variabilă;
- variabilele globale pot fi folosite pentru a transmite datele în subprograme, însă valorile lor nu trebuie să fie schimbate de acestea.

4.1.3 Blocul try-catch

Este un fenomen "natural" ca în programe să se strecoare erori, de diverse categorii. Activitatea de programare implică și acțiuni mai puțin plăcute, adică testarea, depanarea și corectarea erorilor. Costurile de îndepărtare a erorilor creste de obicei direct proporțional cu întârzierea din cadrul procesului de dezvoltare când sunt descoperite.

Trebuie însă înțeleasă diferența dintre erori (bug-uri) și excepții. Excepțiile sunt situațiile neașteptate apărute în cadrul sistemului care rulează un program. Programele trebuie să fie pregătite pentru a trata aceste situații excepționale. Există două tipuri de excepții: sincrone și asincrone. În C++ s-a realizat un mecanism facil de tratare a excepțiilor. Astfel, o excepție este:

a) un obiect a cărui adresă este trimisă din zona de cod, unde a apărut problema către o zonă de cod care trebuie să o rezolve;
b) un răspuns la o circumstanță excepțională care apare atunci când se execută un program, cum ar fi o încercare de divizare la zero.

Există două tipuri de excepții pe care le puteți întâlni în acest proces:

- *Excepții sincrone.* Acest tip de eroare poate fi controlat de programul însuși. De exemplu, erori în cod făcute de programator sau parametri de intrare incorecți.
- *Excepții asincrone.* Acest tip de eroare nu are legătură directă cu codul și nu poate fi controlat de program. De exemplu, poate fi o eroare de

disc, o eroare la deschiderea unui fișier, un socket sau alocarea unui bloc de memorie.

Când are loc un eveniment care întrerupe funcționarea normală a programului, C++ se oprește de obicei și emite un mesaj de eroare. Când se întâmplă acest lucru, se spune că C++ aruncă o excepție. Am menționat deja că anumite cuvinte cheie sunt folosite în C++ pentru a face față erorilor sau excepțiilor.

Excepțiile oferă o modalitate de a transfera controlul dintr-o parte a unui program în alta. Tratarea excepțiilor C++ se bazează pe trei cuvinte cheie (încercați, prindeți și aruncați):

a) *throw* - Ne permite să definim un bloc de cod care va fi verificat pentru erori în timpul execuției sale.
b) *catch* - Cuvântul cheie indică captarea unei excepții, un bloc de cod care este executat atunci când apare o anumită excepție în blocul try.
c) *try* - Un bloc try identifică un bloc de cod pentru care vor fi activate anumite excepții. Este urmat de unul sau mai multe blocuri de *catch*. Ne permite să definim un bloc de cod care va fi verificat pentru erori în timpul execuției sale.

Presupunând că un bloc va genera o excepție, o metodă captează o excepție folosind o combinație de cuvinte cheie *try* și *catch*. Un bloc *try / catch* este plasat în jurul codului care ar putea genera o excepție. Codul dintr-un bloc *try / catch* este denumit cod protejat. Puteți enumera mai multe declarații de captură pentru a prinde diferite tipuri de excepții în cazul în care blocul de încercare (*try*) ridică mai multe excepții în situații diferite.

- *Aruncarea de excepții (throw):* Excepțiile pot fi aruncate oriunde în cadrul unui bloc de cod folosind instrucțiunea throw. Operandul instrucțiunii throw determină un tip pentru excepție și poate fi orice expresie, iar tipul rezultatului expresiei determină tipul de excepție aruncat.
- *Prinderea excepțiilor (catch):* Blocul de captură care urmează blocului de încercare prinde orice excepție. Puteți specifica ce tip de excepție doriți să prindeți și acest lucru este determinat de declarația de excepție care apare între paranteze după cuvântul cheie catch.

Exemplul 1: Împărțirea a două numere întregi.

```
#include <iostream>
using namespace std;
double impartire(int a, int b) {
   if(b==0) {
      throw "Eroare:\nConditia de impartire la zero!";
   }
   return (a/b);
}
// Programul principal
int main () {
    int x=50,y=0; double z = 0.0;
    // Blocul try / catch
    try {
        z=impartire(x,y); cout<<z<<endl;
    }
    catch (const char* mesaj){
        cerr<<mesaj<<endl;
   }
}
```

Deoarece creștem o excepție de tip const char *, în timp ce captăm această excepție, trebuie să folosim const char * în blocul de captură. Dacă compilăm și rulăm codul de mai sus, obținem următorul rezultat:

```
Eroare:
Conditia de impartire la zero!
```

Exemplul 2: Accesarea unui element dintr-un vector care nu există.

```
#include<iostream>
#include<vector>
using namespace std;
int main() {
    vector<int> v; // cream o variabila v de tip vector
    v.push_back(0); // adaugam componenta 0 la vectorul v
    v.push_back(1); // adaugam componenta 1 la vectorul v
    // Accesarea elementului al 3-lea care nu exista
    try{
```

```
        v.at(2);
    }
     // Mesajul de eroare returnat va fi pastrat in variabila
Eroare
    catch (exception &Eroare){
        cout<<"Eroare:\nA aparut o exceptie!"<<endl;
    }
}
```

Dacă compilăm și rulăm codul de mai sus, obținem următorul rezultat:

```
Eroare:
A aparut o exceptie!
```

Excepții standard: Limbajul C++ oferă o listă de excepții standard definite în *<exception>* pe care le putem folosi în programele noastre. Aceste excepții sunt incluse în spațiul de nume **std**.

Tabelul 4.2: Excepții standard în limbajul C++

Nr.	Excepția	Descrierea
1	std::exception	O excepție și clasa părinte a tuturor excepțiilor C ++ standard.
2	std::bad_alloc	Acest lucru poate fi aruncat de **new**.
3	std::bad_cast	Acest lucru poate fi aruncat prin **dynamic_cast**.
4	std::bad_exception	Acesta este un dispozitiv util pentru a gestiona excepțiile neașteptate într-un program C++.
5	std::bad_typeid	Acest lucru poate fi aruncat de **typeid**.
6	std::logic_error	O excepție care teoretic poate fi detectată prin citirea codului.
7	std::domain_error	Aceasta este o excepție aruncată atunci când este utilizat un domeniu matematic nevalid.

Nr.	Excepția	Descrierea
8	std::invalid_argument	Acest lucru este aruncat din cauza unor argumente nevalide.
9	std::length_error	Acest lucru este aruncat atunci când este creat un **std :: string** prea mare.
10	std::out_of_range	Acest lucru poate fi aruncat prin metoda **at**, de exemplu un **std :: vector** și **std :: bitset <> :: operator [] ()**.
11	std::runtime_error	O excepție care teoretic nu poate fi detectată prin citirea codului.
12	std::overflow_error	Aceasta este aruncată, dacă apare o supraîncărcare (overflow) matematică.
13	std::range_error	Acest lucru se întâmplă atunci când încercați să stocați o valoare care este în afara intervalului.
14	std::underflow_error	Acest lucru este aruncat dacă apare o scurgere (underflow) matematică.

Exemplul 3: Un program care folosește throw, try și câteva blocuri de catch.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
     // aici scriem afirmatiile care vor arunca urmatoarea ex-
ceptie
    try {
        throw -1; // declaratie tipica de aruncare
    }
    // orice exceptii de tip int aruncate in blocul try de mai
sus sunt tratate aici
    catch (int a){
          cerr<<"Am prins o exceptie cu valoare de tip int:
"<<a<<'\n';
    }
```

```
    // nu specificam numele variabilei, deoarece nu este necesar, nu-l folosim nicaieri in bloc
    catch (double){
          cerr<<"Am prins o exceptie cu valoare de tip double: "<<'\n';
    }
    // prindem exceptiile prin referinta constanta, orice exceptii duble aruncate in blocul try de mai sus sunt tratate aici
    catch (const string &str){
         cerr<<"Am prins o exceptie cu valoare de tip string: " << '\n';
    }
    cout<<"Continuam mai departe!\n";
}
```

Dacă compilăm și rulăm codul de mai sus, obținem următorul rezultat:

```
Am prins o exceptie cu valoare de tip int: -1
Continuam mai departe!
```

Instrucțiunea *throw* este folosită pentru a arunca o excepție -1 de tip int. Blocul *try* detectează apoi instrucțiunea *throw* și o mută în blocul *catch* corespunzător, care gestionează excepțiile *int*. Blocul *catch* este de tip *int* afișează mesajul de eroare corespunzător. După gestionarea excepției, programul își continuă execuția și afișează „Continuam mai departe!” .

Exemplul 4: Un exemplu mai popular, extragerea radicalului dintr-un număr.

```
#include <iostream>
#include <cmath> // pentru functia sqrt().
using namespace std;
int main(){
    double a;
    cout<<"Introduceti un numar real: "; cin>>a;
    // cauta exceptii in interiorul acestui bloc si le trimite la manipulatorul de exceptii catch
    try {
          // daca utilizatorul a introdus un numar negativ,
```

```
atunci este aruncata o exceptie
        if (a<0.0) throw "Nu se poate extrage radical dintr-un
numar negativ!"; // se arunca o exceptie de tipul const char*
            // daca utilizatorul introduce un număr pozitiv,
atunci operatia este efectuata si rezultatul este afisat
         cout<<"Radical din numarul "<<a<<" este "<<sqrt(a)<<'\
n';
    }
    // manipulatorul de excepții de tip const char*
    catch (const char* exception){
        cerr << "Error: " << exception << '\n';
    }
}
```

Dacă compilăm și rulăm codul de mai sus, obținem următorul rezultat:

```
Introduceti un numar real: -6.932
Error: Nu se poate extrage radical dintr-un numar negativ!
```

Un alt rezultat mai poate fi și acesta:

```
Introduceti un numar real: 6.932
Radical din numarul 6.932 este 2.63287
```

4.1 Modele de itemi pentru exersare

(1) Elaborați în limbajul C++ subprograme care ar permite citirea de la tastatură a unui cuvânt și vizualizarea la consolă a tuturor vocalelor de culoare verde și a consoanelor de culoare galbenă.

(2) Elaborați în limbajul C++ subprograme care ar permite citirea de la tastatură a datelor unei matrici de numere naturale. Afișați datele matricii la consolă astfel: toate elementele pare vor fi de culoare albastră, iar cele impare vor fi de culoare roșie.

(3) Elaborați în limbajul C++ subprograme care ar permite afișarea tuturor numerelor din intervalul [a, b] care au suma cifrelor sale un număr par mai mic decât 10. Tratați excepțiile posibile ale problemei. Un exemplu de număr ar fi 483, acest număr are suma finală a cifrelor 6.

(4) Elaborați în limbajul C++ subprograme care ar permite afișarea tuturor numerelor din intervalul [a, b] care au produsul cifrelor un număr prim mai mic decât 11. Tratați excepțiile posibile ale problemei. Un exemplu de număr ar fi 351, acest număr are produsul final al cifrelor 5.

(5) Elaborați în limbajul C++ subprograme care ar permite afișarea soluțiilor unui polinom de gradul 1, $ax+b=0$, unde numerele a și b sunt întregi care se generează dintr-un interval [m, n]. Tratați excepțiile posibile ale problemei. De exemplu: pentru a=5 și b=10, valoarea lui x este -2.

(6) Elaborați în limbajul C++ subprograme care ar permite afișarea soluțiilor unui polinom de gradul 2, $ax^2+bx+c=0$, unde numerele a, b și c sunt întregi care se generează dintr-un interval [m, n]. Tratați excepțiile posibile ale problemei. De exemplu: pentru a=2, b=6 și c=4, polinomul are două soluții: $x_1= -1$ și $x_2= -2$.

4.2 Recursivitate. Subprograme recursive

În această temă vom analiza următoarele unități de conținut:
- *descrierea și apelarea subprogramelor recursive;*
- *descrierea tipurilor de recursivitate: directă și indirectă;*
- *elaborarea subprogramelor cu mai mulți parametri, conform specificațiilor propuse;*
- *implementarea subprogramelor recursive în limbajul C++.*

Procesul în care un subprogram este apelat în cadrul aceluiaș subprogram este cunoscut ca *recursivitate*. Subprogramul care apelează acelaș subprogram este cunoscut sub numele de *subprogram recursiv.* Un subprogram care se autoapelează și nu efectuează nicio sarcină după apelul subprogramului este cunoscută sub numele de *recursivitate coadă*. Un algoritm se numește *recursiv* dacă definiția sa conține un apel direct sau indirect la același algoritm, o funcție recursivă este una dintre perfecționările matematice ale noțiunii intuitive de funcție computabilă.

Ramura recursivă este un algoritm care, datorită recursivității, ia în considerare anumite caracteristici individuale ale problemei care se rezolvă din domeniul definiției acesteia. *Ramura terminală* a recursiunii este o propoziție care definește o anumită situație inițială sau o situație în momentul încheierii. De regulă, în această propoziție este scris un caz elementar, în care răspunsul este obținut imediat, chiar și fără utilizarea recursiunii.

Un *pas recursiv* este o regulă al cărei corp conține în mod necesar, ca subscop, o chemare la predicatul în curs de definire. Regula principală a recursiunii: înainte de un apel recursiv, trebuie să existe o verificare a revenirii recursiunii.

În următoarele exemple, textul „`Eu sunt Andrian!`” va fi afișat pe ecranul monitorului pentru o lungă perioadă de timp și apoi programul se va bloca din cauza supraîncărcării stivei.

Recursivitate directă a funcției main() în C++:

```
#include <iostream>
using namespace std;
```

```
int main(){
    cout<<"Eu sunt Andrian!"<<endl;
    main();
}
```

Recursivitate indirectă a funcției main() în C++:

```
#include <iostream>
using namespace std;
void f();
int main(){
    cout<<"Eu sunt Andrian!"<<endl;
    f();
}
void f(){
    main();
}
```

Pot fi identificate următoarele avantaje interdependente ale recursiunii:

a) naturalețea prezentării unor algoritmi aparent complecși;
b) algoritmul recursiv este mai lizibil în comparație cu cel iterativ;
c) pentru multe sarcini comune, recursiunea este mai ușor de implementat decât iterația;
d) este potrivită pentru implementarea algoritmilor de parcurgere a listelor, arborilor, analizatoare de expresii, probleme combinatorii, etc.

Dezavantajele recursiunii sunt următoarele:

a) În comparație cu iterația, apelarea în mod repetat a unei funcții recursive durează mai mult timp. Acest lucru se datorează faptului că atunci când o metodă recursivă este apelată, parametrii acesteia sunt copiați în stivă. Când se finalizează un apel recursiv, valorile parametrilor anterioare sunt scoase din stivă, rezultând operații inutile. Un algoritm iterativ pentru aceeași problemă este mai rapid.
b) Dacă funcția recursivă conține cantități mari de variabile interne locale și un număr mare de parametri, atunci utilizarea recursiunii nu este eficientă. Acest lucru se datorează faptului că pentru fiecare apel recursiv, trebuie să facem copii ale acestor variabile și parametri. Cu un număr

mare de apeluri recursive, acest lucru va duce la o utilizare excesivă a memoriei.

Exemplul 1: Elementele șirului de numere 1,1,2,3,5,8,13,21,34, ...

Numerele Fibonacci au fost descrise pentru prima dată în matematica indiană, încă din anul 200 î.Hr., în lucrarea lui Pingala privind enumerarea posibilelor modele de poezie sanscrită formate din silabe de două lungimi.

Primele două elemente ale șirului sunt 1, iar celelalte se obțin prin suma a două elemente precedente, de exemplu $a_3= a_1+a_2= 1+1= 2$, $a_4= a_2+a_3= 1+2 =3$, $a_5= a_3+a_4= 2+3 = 5$, etc.

Concluzie: Formula termenilor în forma recursivă este $a_n=a_{n-1}+a_{n-2}$.

```
int fibonacci(int n){
    if (n==0) return 0;
    if (n==1) return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Exemplul 2: Suma și produsul șirului de numere 1,6,11, 16, 21, 26, 31, ...

Fie variabila *x* o constantă, iar $s_1 = 1$.

- Suma primelor două elemente este $s_2 = s_1+ x$. Noi cunoaștem că $s_2 = 1+6 = 7$, rezultă $x = 6$, deci suma primelor două elemente are formula: $s_2 = s_1+6 = 1+6 = 7$.
- Suma primelor trei elemente este $s_3 = s_2+6$. Noi cunoaștem că $s_3 = 1+6+11 = 18$, observăm că $18 < s_2+6 = 13$, deci 18-13 = 5, rezultă x = 5*n. Verificăm constanta $x = 5*n$ pentru suma $s_2 = s_1+5*n = 1+5*2 = 11$, deci 11-7 = 4, rezultă $x = 5*n-4$. Verificăm constanta $x = 5*n-4$ pentru suma $s_3 = s_2+5*n-4= 7+5*3-4 = 18$.
- Suma primelor patru elemente este $s_4 = s_3+5*n-4$. Noi cunoaștem că $s_4 = 1+6+11+16 = 34$, observăm că $34 = s3+5*4-4$, deci x = 5*n-4.
- Suma primelor cinci elemente este $s_5 = s_4+5*n-4$. Noi cunoaștem că $s_5 = 1+6+11+16+21 = 55$, observăm că $55 = s4+5*5-4$, deci x = 5*n-4.

Concluzie:

a) Formula sumei în forma recursivă este $s_n=s_{n-1}+(5*n-4)$.

b) Formula produsului în forma recursivă este $p_n=p_{n-1}*(5*n-4)$.

```
#include <iostream>
using namespace std;
int s(int n){
    if(n>1) return s(n-1)+(5*n-4); return 1;
}
int p(int n){
    if(n>1) return p(n-1)*(5*n-4); return 1;
}
// afisam valorile sumei si produsului pentru primele 5 numere
int main(){
    for (int i=1; i<=5; i++){
        cout<<"S["<<i<<"]="<<s(i)<<"\tP["<<i<<"]="<<p(i)<<endl;
    }
}
```

Exemplul 3: Suma și produsul elementelor din intervalul [a, b].

Pentru intervalul [1, 5] suma elementelor este: 1+2+3+4+5 = 15, iar produsul elementelor este: 1*2*3*4*5 = 120.

```
#include <iostream>
using namespace std;
int s(int suma,int a,int b){
    if (a>b) return suma;
    return a+s(suma,a+1,b);
}
int p(int produsul,int a,int b){
    if (a>b) return produsul;
    return a*p(produsul,a+1,b);
}
int main(){
    int a=1, b=5;
    cout<<"S["<<a<<","<<b<<"] este "<<s(0,a,b)<<endl;
    cout<<"P["<<a<<","<<b<<"] este "<<p(1,a,b)<<endl;
}
```

4.2 Modele de itemi pentru exersare

(1) Elaborați în limbajul C++ subprograme care ar permite afișarea la consolă a sumei și produsului următoarelor șiruri de numere întregi:

a)	1, 2, 3, 4, ...	e)	2, 5, 8, 11, 14, ...	i)	3, 9, 15, 21, ...
b)	1, 3, 5, 7, ...	f)	2, 6, 10, 14, 18, ...	j)	3, 10, 17, 24, ...
c)	1, 4, 7, 10, ...	g)	2, 7, 12, 17, 22, ...	k)	3, 11, 19, 27, ...
d)	1, 5, 9, 13, ...	h)	2, 8, 14, 20, 26, ...	l)	3, 12, 21, 30, ...

(2) Elaborați în limbajul C++ un subprogram care are ca parametru un număr natural *n* și returnează cel mai mare divizor al său. Numele funcției este *divizorMax*. De exemplu: divizorMax(24) = 3.

(3) Elaborați în limbajul C++ un subprogram care are ca parametru un număr natural *n* în baza 10 și returnează acest număr convertit în baza 16. Numele funcției este *baza16*. De exemplu: baza16(2023) = 7E7.

(4) Elaborați în limbajul C++ un subprogram care generează o matrice pătratică de dimensiunea 2^n-1, unde *n* este un număr natural, conform următoarelor reguli:

- elementul din mijlocul matricii este egal cu *n*;
- elementele de pe linia mediană și cele de pe coloana mediană (exceptând elementul din mijlocul matricii) sunt nule;
- folosind linia mediană și coloana mediană, se împarte matricea în alte 4 matrici care se generează similar, dar au dimensiunea 2^{n-1}-1.

Exemplu:

```
Pentru n=3:
1 0 1 0 1 0 1
0 2 0 0 0 2 0
1 0 1 0 1 0 1
0 0 0 3 0 0 0
1 0 1 0 1 0 1
0 2 0 0 0 2 0
1 0 1 0 1 0 1
```

4.3 Module. Crearea şi implementarea modulelor

În această temă vom analiza următoarele unități de conținut:

- *declararea și inițializarea variabilelor de tip structură / tablou de structuri;*
- *identificarea instrucțiunilor de citire, afișare și accesare a datelor unei structuri / tablou de structuri;*
- *prelucrarea structurilor de date conform specificațiilor propuse;*
- *implementarea algorimilor de prelucrare a structurilor / tablourilor de structuri în limbajul C++.*

Limbajul C are numeroase biblioteci care includ funcții predefinite pentru a simplifica programarea. În limbajul C, fișierele antet conțin setul de funcții standard predefinite ale bibliotecii. Solicităm să utilizați un fișier antet în programul personal incluzându-l cu directiva de preprocesare C „#include”. Toate fișierele antet au extensia „.h”. Prin includerea unui fișier antet, putem folosi conținutul acestuia în programul nostru.

De asemenea, C++ oferă utilizatorilor săi o varietate de funcții, dintre care una este inclusă în fișierele antet. În general, fișierele antet au o anumită extensie, cum ar fi „NumeAntet.h” sau „NumeAntet.hpp”.

Important

- Fișierele antet pot fi divizate după extensii astfel:
  a) *.h - dacă pot fi folosite într-un proiect C;
  b) *.hpp - dacă pot fi utilizate numai într-un proiect C++.
- Rolul fișierelor cu extensii specifice în limbajul C++:
  a) *.h pune / fixează declarații;
  b) *.cpp pune / fixează implementări;
  c) *.hpp înseamnă declarații și implementări împreună, cum ar fi clasele de module.

4.3.1 Fişiere antet standard şi aplicaţiile acestora

(1) `#include<stdio.h>`: este aplicat pentru a efectua operațiuni de intrare și ieșire folosind funcțiile scanf() și printf().
(2) `#include<iostream>`: este aplicat ca flux de intrare și ieșire folosind cin și cout.
(3) `#include<string.h>`: este aplicat pentru a efectua diverse funcționalități cu privire la manipularea șirurilor de caractere, cum ar fi strlen(), strcmp(), strcpy(), size(), etc.
(4) `#include<math.h>`: este aplicat pentru a efectua operații matematice precum: sqrt(), log2(), pow(), etc.
(5) `#include<iomanip.h>`: este aplicat pentru a accesa funcția set() și set-precision() pentru a limita zecimale în variabile.
(6) `#include<signal.h>`: este aplicat pentru a efectua funcții de gestionare a semnalului precum signal() și raise().
(7) `#include<stdarg.h>`: este aplicat pentru a executa funcții argument standard precum va_start() și va_arg(). De asemenea, este folosit pentru a indica începutul listei de argumente cu lungime variabilă și, respectiv, pentru a prelua argumentele din lista de argumente cu lungime variabilă din program.
(8) `#include<errno.h>`: este aplicat pentru a efectua operațiuni de tratare a erorilor precum: errno(), strerror(), perror(), etc.
(9) `#include<fstream.h>`: este aplicat pentru a controla datele de citire dintr-un fișier ca intrare și datele de scris în fișier ca ieșire.
(10) `#include<time.h>`: este folosit pentru a efectua funcții cu privire la date() și time() precum setdate() și getdate(). De asemenea, este folosit pentru a modifica data sistemului și, respectiv, pentru a obține timpul la CPU.
(11) `#include<float.h>`: conține un set de diverse constante dependente de platformă cu privire la valorile în virgulă mobilă. Aceste constante sunt propuse de ANSI C. Ele permit ca programele să fie mai portabile. Câteva exemple de constante incluse în acest fișier antet sunt: e (exponent), b (bază/radix), etc.
(12) `#include<limits.h>`: determină diferite proprietăți ale diferitelor tipuri de variabile. Macrocomenzile definite în acest antet limitează valorile diferitelor tipuri de variabile precum char, int și long. Aceste limite specifică faptul că o variabilă nu poate stoca nicio valoare dincolo de aces-

te limite, de exemplu un caracter fără semn poate stoca până la o valoare maximă de 255.

(13) `#include<assert.h>`: conține informații pentru adăugarea de diagnostice care ajută la depanarea programului.

(14) `#include<ctype.h>`: conține prototipuri de funcții pentru funcții care testează caractere pentru anumite proprietăți și, de asemenea, prototipuri de funcție pentru funcții care pot fi utilizate pentru a converti literele mari în litere mici și invers.

(15) `#include<locale.h>`: conține prototipuri de funcții și alte informații care permit modificarea unui program pentru localitatea curentă în care rulează. Acesta permite sistemului informatic să gestioneze diferite convenții pentru exprimarea datelor, cum ar fi ore, date sau numere mari din întreaga lume.

(16) `#include<setjmp.h>`: conține prototipuri de funcții pentru funcții care permit ocolirea secvenței obișnuite de apelare a funcției și de returnare.

(17) `#include<stddef.h>`: conține definiții comune ale tipurilor utilizate de C pentru efectuarea calculelor.

4.3.2 Crearea şi utilizarea fişierelor antet

Un fișier antet conține: definiții ale funcțiilor (1), definiții ale tipurilor de date (2) și macro-uri (3). Există 2 tipuri de fișiere antet:

1. Fișiere antet preexistente: fișierele care sunt deja disponibile în compilatorul C/C++ trebuie doar să le importam.
2. Fișiere antet definite de utilizator: Aceste fișiere sunt definite de utilizator și pot fi importate folosind „#include”.

Sintaxă

```
#include <TitluFisier.h> // pentru fișiere antet implicite
#include "TitluFisier.h" // prentru fisiere antet noi create
```

Putem include fișiere antet în programul nostru utilizând una dintre cele două sintaxe de mai sus, indiferent dacă este un fișier antet predefinit sau definit de utilizator. Preprocesorul „#include” este responsabil pentru direcționarea compilatorului către fișierul antet ce trebuie procesat înainte de compilare și include toate definițiile necesare pentru tipul de date și funcțiile create.

Putem utiliza diferite fișiere de antet într-un program. Când un fișier antet este inclus de două ori într-un program, compilatorul procesează conținutul acelui fișier antet de două ori. Acest lucru duce la o eroare în program. Pentru a elimina această eroare, sunt utilizate directive de preprocesor condiționat. Nu este bine să includem același fișier antet de două ori în niciun program.

Sintaxă

```
#ifndef HEADER_TitluFisier
#define HEADER_TitluFisier
   // continutul fisierului antet
#endif
```

Acest construct se numește wrapper „#ifndef”. Când antetul este inclus din nou, condiționalul va deveni fals, deoarece HEADER TitluFisier este definit. Preprocesorul va sări peste întregul conținut al fișierului și compilatorul nu îl va vedea de două ori. Uneori este esențial să includem mai multe fișiere de antet diverse, în funcție de cerințele programului. Pentru aceasta, sunt folosite mai multe condiționale.

Sintaxă

```
#if SYSTEM_ONE
        #include "TitluFisier1.h"
#elif SYSTEM_TWO
        #include "TitluFisier2.h"
#elif SYSTEM_THREE
   //...
#endif
```

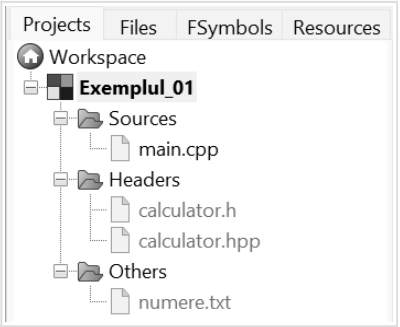
Beneficiile utilizării fișierelor antet *.hpp:

(1) Fișierul *.hpp va defini și implementa același fișier, va reduce numărul de fișiere.

(2) Nu este nevoie să adăugați *.cpp la proiect pentru compilare, compilați codul direct în fișierul *.obj al apelantului, nu mai este născut într-un *.obj separat, reducem numărul de compilărie, foarte potrivit pentru scrierea bibliotecii open source.

(3) Biblioteca Boost folosește un șablon, iar forma Sampling HPP poate menține o compatibilitate mai bună cu fiecare compilator (șablonul nu separă fișierele sursă și fișierele de declarație în două fișiere).

Exemplul 1: Crearea unui fișier antet care va permite efectuarea următoarelor operații matematice asupra a două numere întregi: suma, diferența, produsul, câtul, restul și ridicarea la putere. Cele două numere se vor citi dintr-un fișier text. Structura proiectului în mediul de programare Code::Blocks trebuie să fie asemenea cu imaginea alăturată.




```
#ifndef CALCULATOR_H_INCLUDED
#define CALCULATOR_H_INCLUDED
#include <cmath>
int s(int a, int b){
    return (a+b);
}
int dif(int a, int b){
    return (a-b);
}
int prod(int a, int b){
    return (a*b);
}
int cat(int a, int b){
    return (a/b);
}
int rest(int a, int b){
    return (a%b);
}
int put(int a, int b){
    return (pow(a,b));
}
#endif // CALCULATOR_H_INCLUDED
```

Exemplul 2: Aplicarea fișierului antet „calculator.h” pentru a afișa următoarele operații matematice atunci când vom avea trei numere întregi: suma, produsul și ridicarea la putere. Fișierul antet nu trebuie să fie modificat.

```
#include <iostream>
#include <fstream>
#include "calculator.h" // sau putem scrie calculator.hpp
using namespace std;
ifstream fin("numere.txt");
int a,b,c;
void citire(){
    fin>>a>>b>>c; fin.close();
}
int main(){
   citire();
   cout<<a<<" + "<<b<<" + "<<c<<" = "<<s(s(a,b),c)<<endl;
   cout<<a<<" * "<<b<<" * "<<c<<" = "<<prod(prod(a,b),c)<<endl;
   cout<<a<<" ^ "<<b<<" ^ "<<c<<" = ";
   cout<<(unsigned long long int)put(put(a,b),c)<<endl;
}
```

Fișierele antet `calculator.h` și `calculator.hpp` sunt cu conținut identic și pot fi implementate într-un program principal. Dacă compilăm și rulăm codul de mai sus, obținem următorul rezultat:

```
12 + 34 + 56 = 102
12 * 34 * 56 = 22848
12 ^ 34 ^ 56 = 18446744071562067968
```

Exemplul 3: Crearea unui fișier antet care va permite efectuarea următoarelor operațiuni asupra uni vector de numere întregi: afișarea elementului maximal și a celui minimal, afișarea numărului de elemente pare și impare.

```
#ifndef OPERATII_HPP_INCLUDED
#define OPERATII_HPP_INCLUDED
#include <cmath>
int i;
int maxim(int a[], int n){
    int m=a[0];
```

```
    for (i=0; i<n; i++){
        if(a[i]>m) m=a[i];
    }
    return m;
}
int minim(int a[], int n){
    int k=a[0];
    for (i=0; i<n; i++){
        if(a[i]<k) k=a[i];
    }
    return k;
}
int NrPareVector(int a[], int n){
    int pare=0;
    for (i=0; i<n; i++){
        if(a[i]%2==0) pare++;
    }
    return pare;
}
int NrImpareVector(int a[], int n){
    int impare=0;
    for (i=0; i<n; i++){
        if(a[i]%2!=0) impare++;
    }
    return impare;
}
#endif // OPERATII_HPP_INCLUDED
```

```
#include <iostream>
#include "calculator.hpp"
using namespace std;
int main(){
    int a[]={34,53,71,95,26,47,68}, n=sizeof(a)/sizeof(a[0]);
    cout<<"Elementul maxim din vector: "<<maxim(a,n)<<endl;
    cout<<"Elementul minim din vector: "<<minim(a,n)<<endl;
    cout<<"Numarul de elemente pare din vector: ";
    cout<<NrPareVector(a,n)<<endl;
    cout<<"Numarul de elemente impare din vector: ";
    cout<<NrImpareVector(a,n)<<endl;
}
```

4.3 Modele de itemi pentru exersare

(1) Elaborați în limbajul C++ un modul care va permite calcularea volumului corpurilor geometrice regulate dacă se cunoaște lungimea laturii corpului geometric. Corpurile geometrice regulate sunt: tetraedrul, hexaedrul și dodecaedrul.

Formula pentru volumul tetraedrului: $V=\frac{\sqrt{2}}{12}\cdot a^3$.	*Formula pentru volumul hexaedrului:* $V=a^3$.	*Formula pentru volumul dodecaedrului:* $V=\frac{1}{4}\cdot(15+7\sqrt{5})\cdot a^3$.

(2) Elaborați în limbajul C++ un modul care va permite generarea unui număr dintr-un anumit interval [a, b], conform condiției:
a) numărul să fie par;
b) numărul să fie impar;
c) numărul să fie prim;
d) numărul să fie palindrom.

(3) Elaborați în limbajul C++ un modul care va permite efectuarea unor operațiuni asupra elementelor unui vector de numere citit din fișier:
a) suma elementelor;
b) media aritmetică;
c) media geometrică;
d) media armonică.

(4) Elaborați în limbajul C++ un modul care va permite efectuarea unor operațiuni de afișare pentru elementele unei matrici pătratice:
a) elementul maximal;
b) elementul minimal;
c) poziția elementului maximal;
d) poziției elementului minimal.

(5) Elaborați în limbajul C++ un modul care va permite efectuarea unor operațiuni asupra elementelor unui șir de caractere citit din fișier:
a) numărul de vocale;
b) numărul de consoane;
c) numărul de cifre;
d) numărul de spații.

BIBLIOWEBOGRAFIE

1. Антон Спрол, „*Думай как программист. Креативный подход к созданию кода. C++ версия*”, 2017, 274 стр.
2. A. Gremalschi, I. Mocanu, I. Spinei, L. Gremalschi, „*Informatică - Manual pentru clasa a 10-a* ”, Editura Știința, 2020, 212 pag.
3. Мариус Бансила, „*Решение задач на современном C++*”, 2019, 304 стр.
4. Артур О'Двайр, „*Осваиваем C++17 STL*”, 2019, 354 стр.
5. Алексей Васильев, “*Программирование на C++ в примерах и задачах*”, 2017, 369 стр.
6. Михаил Абрамян, „*Введение в стандартную библиотеку шаблонов C++. Описание, примеры использования, учебные задачи*”, 2017, 400 стр.
7. Антон Спрол, „*Думай как программист. Креативный подход к созданию кода. C++ версия*”, 2017, 274 стр.
8. Daniel Vișan-Dimitriu, „*Limbajul C++ fără profesor, ediția a II-a revizuită pentru Code::Blocks*”, Editura Else, 2016, 118 pag.
9. Tutorial Point, „*Learn C++ programming language*”, 2014, 54 pag.
10. Runceanu A., Runceanu M., „*Noțiuni de programare în limbajul C++*”, Editura Academica Brâncuși, 2012, 483 pag.
11. Никита Культин, „*C/C++ в задачах и примерах (2-е издание)*”, 2009, 351 стр.
12. J. Saulié, „*C++ Language Tutorial* ”, 2007, 144 pag.
13. Bogdan Pătruț, Carmen Muraru, „*Aplicații în C și C++*”, Editura Edu-Soft, 2006, 218 pag.
14. Cerchez E., Șerban M., „*Programarea in limbajul C/C++ pentru liceu* ”, Editura Polirom, 2005, 296 pag.
15. Istrati S. G., „*Iniţializare în limbajele C şi C++* ”, Editura UTM, 2003, 106 pag.

16. Ulla Kirch-Prinz, P. Prinz, „*A Complete Guide to Programming in C++* ”, 2002, 846 pag.
17. Negrescu L., „*Limbajele C/C++ pentru începători*”, Editura Albastră, 1995, 783 pag.
18. Steve Oualline, „*Practical C++ Programming* ”, 1995, 549 pag.
19. Site-ul oficial al dezvoltatorilor C++: https://cplusplus.com

ANEXA 1

Test de autoevaluare nr. 1
EXPRESII ȘI FORMULE ÎN LIMBAJUL C++

Barem de corectare

Item	Descrierea	Punctaj
1	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: (a) includerea fișierului antet <cmath>; (b) definirea / declararea corectă a constantelor; (c) definirea / declararea corectă a datelor de intrare; (d) introducerea de la tastatură a datelor de intrare; (e) aplicarea corectă a funcțiilor matematice; (f) afișarea răspunsului conform condiției problemei.	6*2=12p
2	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: a) definirea / declararea corectă a datelor de intrare; b) introducerea de la tastatură a datelor de intrare; c) utilizarea unui algoritm eficient de rezolvare; d) afișarea răspunsului conform condiției problemei.	4*2=8p
3	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: a) includerea fișierului antet <cstring>; b) definirea / declararea corectă a datelor de intrare; c) introducerea de la tastatură a datelor de intrare; d) stabilitea culorii pentru cuvântul „Prenume”; e) stabilitea tipului italic pentru cuvântul „Prenume”; f) afișarea răspunsului conform condiției problemei.	6*2=12p
4	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: a) includerea fișierelor antet <stdlib.h> și <time.h>;	4*2=8p

Item	Descrierea	Punctaj
	b) definirea / declararea corectă a datelor de intrare; c) aplicarea corectă a funcțiilor de generare a numărului conform specificațiilor propuse; d) afișarea răspunsului conform condiției problemei.	
5	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: a) includerea fișierului antet <iomanip>; b) definirea / declararea corectă a datelor de intrare; c) introducerea de la tastatură a datelor de intrare; d) aplicarea corectă a operației matematice; e) afișarea răspunsului conform condiției problemei.	5*2=10p

Barem de notare în procente

Nota	10	9	8	7	6	5	4	3	2
Puncte	100-95	94-87	86-76	75-61	60-45	44-31	30-20	19-11	10-5

Barem de notare în puncte

Nota	10	9	8	7	6	5	4	3	2
Puncte	50-48	47-44	43-38	37-31	30-23	22-16	15-10	9-6	5-3

Timp recomandat: 45 - 60 minute

Varianta 1	Varianta 2
1. Elaborați un program C++ care va calcula valoarea expresiei matematice: $a^{\lvert a-b\rvert}+\log_2(2^e)-\sin(\pi)$. Numerele întregi a și b se vor introduce de la tastatură, variabilele e și π sunt constante.	1. Elaborați un program C++ care va calcula valoarea expresiei matematice: $a^{\lvert -b\rvert}+\sqrt[a]{\lvert(b-a)^e\rvert}-\cos(\pi)$. Numerele întregi a și b se vor introduce de la tastatură, variabilele e și π sunt constante.
2. Suma a trei numere pare consecutive este 72. Suma primelor două numere pare consecutive este 46. Elaborați un program C++ care va permite determinarea acestor trei numere și le va afișa la ecran.	2. Suma a trei numere prime consecutive este 71. Suma primelor două numere prime consecutive este 42. Elaborați un program C++ care va permite determinarea acestor trei numere și le va afișa la ecran.
3. Elaborați un program C++ care va permite citirea de la tastatură a prenumelui personal și va afișa la ecran un mesaj conform exemplului: `Buna seara Andrian`	3. Elaborați un program C++ care va permite citirea de la tastatură a prenumelui personal și va afișa la ecran un mesaj conform exemplului: `Buna dimineata Andrian`
4. Elaborați un program C++ care va permite generarea unui număr par de exact 3 cifre.	4. Elaborați un program C++ care va permite generarea unui număr impar de exact 5 cifre.
5. Elaborați un program C++ care va permite împărțirea a două numere întregi și va afișa rezultatul cu 9 cifre după virgulă.	5. Elaborați un program C++ care va permite împărțirea a două numere reale și va afișa rezultatul cu 6 cifre după virgulă.

ANEXA 2

Test de autoevaluare nr. 2
STRUCTURI DE CONTROL ÎN LIMBAJUL C++

Item	Descrierea	Punctaj
1	Se acordă câte 1 punct pentru fiecare răspuns corect.	2*1=2p
2	Se acordă câte 1 punct pentru răspunsul corect.	1*1=1p
3	Se acordă câte 1 punct pentru realizarea corectă a următoarelor operațiuni: (a) definirea / declararea corectă a datelor de intrare; (b) introducerea corectă a datelor de intrare; (c) aplicarea corectă a instrucțiunii de decizie; (d) afișarea răspunsului conform condiției problemei.	4*1=4p
4	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: (a) definirea / declararea corectă a datelor de intrare; (b) introducerea corectă a datelor de intrare; (c) aplicarea corectă a instrucțiunii ciclice; (d) afișarea răspunsului conform condiției problemei.	4*2=8p
5	Se acordă câte 2 puncte pentru realizarea corectă a următoarelor operațiuni: (a) definirea / declararea corectă a datelor de intrare; (b) introducerea corectă a datelor de intrare; (c) aplicarea corectă a instrucțiunii de decizie; (d) aplicarea corectă a instrucțiunii ciclice; (e) afișarea răspunsului conform condiției problemei.	5*2=10p

Barem de notare în puncte

Nota	10	9	8	7	6	5	4	3	2
Puncte	25-24	23-22	21-19	18-16	15-12	11-8	7-5	4-3	2

Timp recomandat: 45 - 60 minute

<table>
<tr><th>Varianta 1</th><th>Varianta 2</th></tr>
<tr><td>1. Precizați valoarea de adevăr a următoarelor enunțuri:
a) Structura alternativă dirijează execuția unei secvențe de instrucțiuni în funcție de valoarea unei condiții plasate în blocul de decizie. (A / F)
b) Folosind combinația "if ... else", putem stabili comenzi care să fice executate și când condiția instrucțiunii "if" este FALSE. (A / F)</td><td>1. Precizați valoarea de adevăr a următoarelor enunțuri:
a) Valorile din clauzele case a instrucțiunii de decizie switch, nu e neapărat să fice doar constante întregi sau caractere. (A / F)
b) Instrucțiunea switch analizează o problemă în funcție de cazuri. Dacă nici un caz nu se potrivește cu variabila, atunci se va executa o instrucțiune implicită (default). (A / F)</td></tr>
<tr><td>2. Ce va afișa la consolă următoarea secvență de cod C++:
A. 44 B. 27 C. 35 D. 19
<pre>#include <iostream>
#include <cmath>
using namespace std;
int main(){
flaoat x=9,f;
if (x<=7) f=-6*x+2;
else cout<<sqrt(pow(x,3));
}</pre></td><td>2. Ce va afișa la consolă următoarea secvență de cod C++:
A. 2 B. 1 C. 4 D. 0
<pre>#include <iostream>
#include <cmath>
using namespace std;
int main(){
flaoat x=9.6,f=2.7;
if (x>ceil(x))
cout<<abs(x);
}</pre></td></tr>
<tr><td>3. Elaborați un cod C++ care va determina dacă o valoare X aparține intervalului [a, b] și conține exact două cifre. Toate variabilele se vor citi de la tastatură, iar rezultatul se va afișa sub forma „Da” sau „Nu”. Aplicați operatorul ternar pentru a rezolva problema.</td><td>3. Elaborați un cod C/C++ care va determina dacă două numere A și B înmulțite formează un număr care se împarte la 3. Toate variabilele se vor citi de la tastatură, iar rezultatul se va afișa sub forma „Da” sau „Nu”. Aplicați instrucțiunea switch pentru a rezolva problema.</td></tr>
</table>

Varianta 1	Varianta 2
4. Elaborați un cod C/C++ care va afișa suma și media armonică a tuturor numerelor din internalul [0, N]. Numărul N este natural, introdus de la tastatură. Se va utiliza instrucțiunea ciclică DO ... WHILE.	4. Elaborați un cod C/C++ care va afișa suma și media geometică a tuturor numerelor din internalul [0, N]. Numărul N este natural, introdus de la tastatură. Se va utiliza instrucțiunea ciclică FOR.
5. Elaborați un program C/C++ care va permite găsirea tuturor numerelor perfecte din intervalul [1, 500]. De exemplu: numărul N=6 se numește perfect, deoarece este egal cu suma divizorilor săi mai mici ca N: 1+2+3=6.	5. Elaborați un program C/C++ care va permite găsirea tuturor numerelor palindrom din intervalul [100, 1000]. De exemplu: numărul N=121 se numește palindrom, deoarece este același număr când îl citim de la dreapta și de la stânga.

NOTIȚE

ANEXA 3

Test de autoevaluare nr. 3
TIPURI DE DATE STRUCTURATE ÎN LIMBAJUL C++

Descrierea	Punctaj
Se acordă câte 2 punct pentru următoarele operațiuni: (1) includerea fișierelor antet corespunzătoare; (2) citirea corectă a celor 20 de înregistrări conform structurii de date din fișierul extern creat; (3) închiderea fișierului de intrare; (4) pregătirea aceluiași fișier pentru rescriere; (5) păstrarea în același fișier a datelor inițiale fără a fi șterse; (6) închiderea fișierului de ieșire; (7) definirea / declararea corectă a datelor de intrare; (8) aplicarea corectă a instrucțiunii de decizie; (9) aplicarea corectă a instrucțiunii ciclice; (10) afișarea la consolă a datelor conform condiției intervalului; (11) afișarea la consolă a datelor conform condiției de sortare a datelor; (12) adăugarea câmpului nou în structură; (13) calcularea corectă a valorilor câmpului nou pentru fiecare înregistrare; (14) afișarea datelor în același fișier, sub formă tabelară, cu un rând mai jos de cele inițiale; (15) aplicarea eficientă a memoriei și dimensunii secvenței de cod C++.	15*2=30p

Barem de notare în puncte

Nota	10	9	8	7	6	5	4	3	2
Puncte	30-29	28-27	26-23	22-19	18-14	13-10	9-6	5-4	3-2

Timp recomandat: 45 - 60 minute

Varianta 1	Varianta 2
Elaborați un program în C++ care va implementa o structură *elev* și următoarele operațiuni asupra ei: • citirea datelor structurii dintr-un fișier text extern cu minim 20 de înregistrări; • afișarea la consolă a tuturor caracteristicilor elevilor de sex feminin care au media generală (*mg)* cuprinsă în intervalul [7.00, 9.99]; • afișarea tuturor elevilor și caracteristicile acestora în ordine ascendentă conform algoritmului *SelectionSort* după câmpul *nume*; • afișați lista elevilor și caracteristicile acestora în același fișier, fără a pierde informația precedentă, după ce adăugați la structura inițială câmpul *bursa*, care se va obține astfel: dacă *mg*>=7.00, atunci *bursa* = *mg* * 70, în caz contrar se va afișa 0.	Elaborați un program în C++ care va implementa o structură *profesor* și următoarele operațiuni asupra ei: • citirea datelor structurii dintr-un fișier text extern cu minim 20 de înregistrări; • afișarea la consolă a tuturor caracteristicilor profesorilor de sex masculin care au experiena de muncă (*vechime)* cuprinsă în intervalul [5, 20]; • afișarea tuturor profesorilor și caracteristicile acestora în ordine descendentă conform algoritmului *InsertionSort* după câmpul *da* (data angajării); • afișați lista profesorilor și caracteristicile acestora în același fișier, fără a pierde informația precedentă, după ce adăugați la structura inițială câmpul *premiu*, care se va obține astfel: dacă *vechime*<=5, atunci *premiu* = *vechime* * 0.7, în caz contrar se va afișa 2000.
`` struct data{ int ziua, luna, anul; } struct elev{ char nume[9], prenume[9]; data dn; // data nasterii float mg; // media } ```	``` struct data{ int ziua, luna, anul; } struct profesor{ char nume[9], prenume[9]; data da; // data angajarii int vechime; // experienta } ```

ANEXA 4

Test de autoevaluare nr. 4
SUBPROGRAME ȘI MODULE ÎN LIMBAJUL C++

Item	Descrierea	Punctaj
1	(1) crearea modulului în C++ conform condiției; (2) aplicarea tipului de date pointer în cele 4 subprograme create; (3) crearea celor 4 subprograme de sortare ascendentă conform celor 4 metode de sortare studiate; (4) crearea unui subprogram de citire a datelor vectorului din fișier text; (5) crearea unui proiect C++ care va implementa modulul creat și va afișa la consolă rezultatele sortărilor.	2 4*1 4*3 2 3
2	(1) crearea modulului în C++ conform condiției; (2) aplicarea tipului de date struct în cele 5 subprograme create; (3) crearea celor 5 subprograme conform celor 5 formule matematice aplicate; (4) crearea unui subprogram de citire a datelor tabloului de structuri din fișier text; (5) crearea unui proiect C++ care va implementa modulul creat și va afișa la consolă rezultatele calculelor.	2 5*1 5*2 2 3
	Total puncte:	***55 puncte***

Barem de notare în puncte

Nota	10	9	8	7	6	5	4	3	2
Puncte	55-52	51-48	47-42	41-34	33-25	24-17	15-11	10-6	5-3

Timp recomandat: 45 - 60 minute

<table>
<tr><th>Varianta 1</th><th>Varianta 2</th></tr>
<tr><td>1. Elaborați un modul, ascendent.h, care va aplica toți algoritmii de sortare studiați.
• Creați 4 subprograme cu pointeri pentru sortarea elementelor unui vector de numere întregi pozitive. Datele inițiale se vor citi din fișier.
• Includeți biblioteca creată la un program C++ și testați funcționalitatea acesteia.</td><td>1. Elaborați un modul, descendent.h, care va aplica toți algoritmii de sortare studiați.
• Creați 4 subprograme cu pointeri pentru sortarea elementelor unui vector de numere întregi negative. Datele inițiale se vor citi din fișier.
• Includeți biblioteca creată la un program C++ și testați funcționalitatea acesteia.</td></tr>
<tr><td>2. Elaborați un modul, cubul.h, care va include subprograme pentru prelucrarea informațiilor unei structuri de date. Structura de date cub va conține minim 5 înregistrări. Datele se vor citi din fișierul text extern. Fie următoarea structură:
<pre>struct cub{
      int latura;
}</pre>Creați subprograme pentru a calcula pentru cub:
• aria laterală, aria totală;
• volumul;
• raza sferei înscrise;
• raza sferei curcumscrise.
Includeți biblioteca creată la un program C++ și testați funcționalitatea acesteia.</td><td>2. Elaborați un modul, tetraedru.h, care va include subprograme pentru prelucrarea informațiilor unei structuri de date. Structura de date tetraedru va conține minim 5 înregistrări. Datele se vor citi din fișierul text extern. Fie următoarea structură:
<pre>struct tetraedru{
      int latura;
}</pre>Creați subprograme pentru a calcula pentru tetraedrul regulat:
• aria laterală, aria totală;
• volumul;
• raza sferei înscrise;
• raza sferei curcumscrise.
Includeți biblioteca creată la un program C++ și testați funcționalitatea acesteia.</td></tr>
</table>

ANEXA 5

Modele de itemi pentru pregătirea de olimpiadă[2]

Partea 1 *Probleme elementare*

Condiția	*Intrare*	*Ieșire*
(1) Fie o sferă de rază R, în care este înscris un cilindru cu aria suprafeţei laterale maximă. Elaborați un program C++ care va determina înălţimea H a cilindrului și aria maximală S a suprafeței cilindrului. Calculați suprafața laterală.	5	7.071 314.159
(2) Fie într-un triunghi dreptunghic cu lungimea ipotenuzei L şi măsura unui unghi ascuţit de 60^0 este înscris un dreptunghi, baza mare a căruia se află pe ipotenuză. Elaborați un program C++ care va determina lungimile laturilor dreptunghiului, pentru ca aria dreptunghiului să fie maximă. Calculați aria dreptunghiului.	24	5,196 12 62,353
(3) Fie un fir de ață este înfășurat simetric în jurul unei bobine în formă de cilindru. Firul de ață este înfășurat exact de 4 ori pe suprafața bobinei, absolut simetric, acoperind întreaga suprafață exterioară. Cunoscând dimensiunea circumferinței bobinei D și de asemenea lungimea acesteia L, elaborați un program C++ care va afișa care este lungimea firului de ață înfășurat.	4 12	20

2 În conformitate cu setul de probleme propuse la olimpiadele locale în cadrul instituției din perioada 1992 -2020. Condițiile problemelor fiind reformulate, adaptate.

Condiția	*Intrare*	*Ieșire*
(1) Fie un număr natural N. Elaborați un program C++ care va determina toți divizorii numărului.	`4`	`1 2 4`
(2) Fie un număr natural N. Elaborați un program C++ care va afișa suma cifrelor numărului N^2.	`12`	`9`
(3) Fie un număr natural N scris în sistemul binar. Elaborați un program C++ care va converti numărul binar N în sistemul zecimal.	`101101`	`45`
(4) Fie un număr natural N. Elaborați un program C++ care va rearanja cifrele acestui număr în așa fel încât să se obțină cel mai mare număr reprezentat prin cifrele sale.	`5631`	`6531`
(5) Fie coordonatele ale vârfurilor unui triunghi sunt $A(x_1,y_1)$, $B(x_2,y_2)$ și $C(x_3,y_3)$. Elaborați un program C++ care va determina poziția unui punct P(x,y) față de triunghi: interior, exterior sau pe o latură.	`1 1` `2 5` `5 2` `4 4`	`exterior`
(6) Fie 4 puncte în plan cu coordonatele sale. Elaborați un program C++ care va determina dacă punctele $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$ și $D(x_4,y_4)$ pot forma un pătrat sau un dreptunghi.	`1 1` `1 6` `6 6` `6 1`	`da` `nu`
(7) Fie trei numere reale A, B și C. Elaborați un program C++ care va determina dacă numerele pot reprezenta lungimile laturilor unui triunghi. Dacă există un asemenea triunghi, atunci afișați un mesaj care va specifica ce tip de triunghi este: ascuțit, isoscel, dreptunghic sau optuz.	`10` `10` `15`	`da` `isoscel`

Condiția	*Intrare*	*Ieșire*
(1) Fie A este un vector de N cifre. Elaborați un program C++ care va insera între oricare două numere impare suma acestora.	5 1 3 6 3 5	7 1 4 3 6 3 8 5
(2) Fie A este un vector de N numere reale. Elaborați un program C++ care va elimina elementele nule și apoi va afișa vectorul de elemente redimensionat după excluderea elementelor nule.	5 3 0 2 0 7	3 3 2 7
(3) Fie vectorii X și Y, elementele cărora sunt coordonatele a N puncte: punctul A[i] are coordonatele X[i] și Y[i]. Elaborați un program C++ care va determina coordonatele celui mai îndepărtat punct de la originea de coordonate.	5 2 3 4 5 3 5 5 2 4 4	4 5 6.40312
(4) Fie vectorii X și Y de dimeniunea N, elementele cărora sunt ordonate ascendent. Elaborați un program C++ care va determina reuniunea dintre acești vectori cu condiția ca vectorul obținut R să conțină elementele sortate descendent.	4 1 2 3 4 3 4 6 9	6 9 6 4 3 2 1
(5) Fie A este un vector de N numere naturale. Elaborați un program în C++ care va afișa o submulţime a cărei sumă se divide cu N.	5 4 2 6 1 5	15 4 6 5

Condiția	*Intrare*	*Ieșire*
(1) Fie o matrice pătrată K de dimensiunea NxN, care are toate elementele distincte. Elaborați un program C++ care va calcula media aritmetică a elementelor strict pozitive aflate deasupra diagonalei secundare.	3 1 2 3 4 5 6 7 8 9	3.666666
(2) Fie o matrice pătrată P de dimensiunea NxN, care are toate elementele distincte. Elaborați un program C++ care va rearanja liniile matricei P în așa fel încât elementul maximal de pe fiecare linie să se afle pe diagonala secundară. Dacă acest lucru nu este posibil, să se afișeze mesajul „Problema nu are soluție!”.	3 1 11 12 15 4 13 16 19 7	1 11 **12** 16 **19** 7 **15** 4 13
(3) Fie o matrice A de dimensiunea NxM. Un element al matricei se numește punte, dacă el este concomitent elementul maximal în linia sa și minimal în coloana sa. Elaborați un program C++ care va determina pozițiile [i][j] ale tuturor elementelor punte. Dacă nu există așa tipuri de elemente, atunci să se afișeze „Eroare!”.	3 1 2 3 4 5 6 7 8 9	[0][2]
(4) Fie o matrice A de dimensiunea NxM. O rază porneşte din colţul stânga jos al matricei A, formând un unghi de 45^0 cu prima coloană şi cu ultima linie. Raza se reflectă când ajunge pe prima sau ultima linie, respectiv prima sau ultima coloană. Elaborați un program C++ care va afişa la ecran toate poziţiile [i][j] atinse de rază, până când ea ajunge într-un colţ al matricei. Numerele naturale M şi N se vor introduce de la tastatură.	3 4	[3][1] [2][2] [1][3] [2][4] [3][3] [2][2] [1][1]

(1) Fie un mesaj sau cuvânt S. Elaborați un program C++ care va mișca continuu pe ecran mesajul S, caracterele care "ies" prin stânga să "intre" prin dreapta până la apăsarea unei anumite taste.

(2) Fie un mesaj sau cuvânt C. Elaborați un program C++ care va colora în acest mesaj toate vocalele în galben, iar toate consoanele în verde.

(3) Fie un mesaj sau cuvânt M ce conține prenumele personal și numărul de telefon mobil. Elaborați un program C++ care va afișa toate cifrele din numărul de telefon în *italic*, iar prenumele va fi colorat în roșu.

(4) Fie două şiruri de caractere A şi B de aceeaşi lungime. Elaborați un program C++ care va verifica dacă şirul A reprezintă o permutare a caracterelor şirului B. De exemplu, pentru A=„Concurs” și B=„onCucsr” se va afișa mesajul „Da”, iar pentru A=„Concurs” și B=„omcucsr” se va afișa mesajul „Nu”.

(5) Fie un şir de caractere **S** ce conţine mai multe cuvinte. Cuvintele sunt separate cel puţin printr-un spaţiu. Elaborați un program C++ care va suprima (elimina) cuvintele ce se află pe poziţii impare, iar pe cele de pe poziţii pare le va invers ordinea caracterelor. De exemplu, pentru mesajul S= „Ana anul acesta a acumulat cele mai bune note” se va obține mesajul „luna a elec enub”.

(6) Fie un şir de caractere S, obţinut în rezultatul codificării unei propoziţii X după următoarea regulă: caracterele propoziţiei X se numără din p în p, prin parcurgerea circulară în direcţia mişcării acelor ceasornicului, pornind de la primul din stânga caracter; pe parcursul acestei numărări, caracterele, cărora le revine numărul p, succesiv se şterg din X şi se alipesc la dreapta şirului S, care iniţial este vid. Elaborați un program C++ care va decodifica șirul dat S, adică va restabili propoziția X. De exemplu, pentru S=„iiamdplaO” și p = 3 se va afișa S=„Olimpiada”.

NOTIȚE