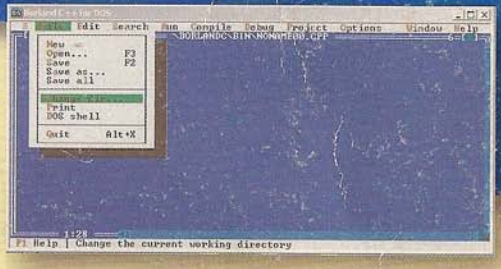
Ministerul Educaţiei şi Cercetării

Informatică

Profil real



Mioara Gheorghe
(coordonator)

Constanţa Năstase
Monica Tătărâm

Manual pentru clasa a X - a

CORINT

Ministerul Educaţiei şi Cercetării

Informatică

Profil real

Specializările:
matematică-informatică
ştiinţe ale naturii

Mioara Gheorghe
(coordonator)

Constanţa Năstase

Monica Tătărâm

CORINT

Manual pentru clasa a X - a

LIMBAJE DE PROGRAMARE – ELEMENTE DE BAZĂ

1. NOȚIUNI INTRODUCTIVE

1.1. Evoluția limbajelor de programare

În clasa a IX-a, s-au construit algoritmi pentru rezolvarea unor probleme din diverse arii de activitate (matematică, fizică, chimie etc.). Algoritmii au fost reprezentați prin secvențe pas cu pas sau în pseudocod. Pentru a prelucra un algoritm prin intermediul unui sistem de calcul, algoritmul trebuie descris într-un limbaj de programare.

Rețineți!

- *Limbajul de programare* reprezintă un mijloc de comunicare între utilizatorul uman, care este programatorul, și sistemul de calcul.
- Descrierea algoritmului în limbaj de programare se face cu ajutorul unui *program*.
- Un *program* este o succesiune de comenzi — *instrucțiuni* ce vor fi executate de sistemul de calcul.
- Un calculator poate să „înțeleagă" mai multe limbaje de programare întrucât fiecare limbaj are un „traducător" — *compilator* propriu.

Evoluția limbajelor de programare a avut loc în paralel cu evoluția sistemelor de calcul (figura 1).

Figura 1. Evoluția limbajelor de programare

1

♦ Generațiile cele mai importante ale limbajelor de programare

1. *Limbaje cod-mașină*

Denumite și *limbaje de bază* sau *de nivel zero*, *limbajele cod-mașină* descriu instrucțiunile în sistemul de numerație binar (secvențe de 1 și 0). Programele sunt executate numai de calculatorul pentru care au fost scrise.

Primul program a fost realizat pentru mașina mecanică a lui Charles Babbage (1834) de către contesa Ada Lovelace, fiica poetului Lord Byron.

2. *Limbaje de asamblare*

Limbajele de asamblare au la bază un set de coduri (*mnemonice*) care sunt reprezentări simbolice ale instrucțiunilor mașină. Un program specializat, asamblorul, translatează aceste coduri în sistemul binar, astfel încât să poată fi decodificate și prelucrate de procesorul calculatorului. Fiecare tip de procesor are un limbaj de asamblare propriu.

3. *Limbaje de nivel înalt*

Limbajele de programare de nivel înalt sunt mai apropiate de limbajul natural în care gândim și comunicăm noi. Aceste limbaje folosesc cuvinte din vocabularul limbii engleze, sunt accesibile și au o arie largă de aplicație: calcule științifice sau economice, reprezentări grafice, probleme de optimizare, jocuri.

Cele mai reprezentative limbaje de nivel înalt sunt:

- FORTRAN (FORmula TRANslation) — a apărut în anul 1955, fiind destinat calculelor tehnico-științifice.
- COBOL (COmmon Businesss Oriented Language) — a apărut în anul 1960; limbajul este orientat spre rezolvarea problemelor economice.
- BASIC (Begginer's Allpurpose Symbolic Instructions Code) — limbajul a fost conceput în anul 1964, impunându-se puternic în perioada 1975-1980. Variantele realizate mai recent (Quick Basic, Visual Basic) sunt utilizate cu succes pentru dezvoltarea unor aplicații complexe.
- PASCAL — definit în anul 1971 de către Niklaus Wirth, a fost îmbunătățit în noi variante: Turbo Pascal, Borland Pascal, Delphi varianta vizuală. Versiunea actuală permite și programarea orientată spre obiecte (OOP).
- C/C++ — este creat în anul 1972 de către Dennis Ritchie și Brian Kernigham de la firma Bell Laboratories pentru dezvoltarea sistemului de operare Unix. Acest limbaj dispune de facilități specifice limbajelor de asamblare (calculul pe biți, prelucrarea adreselor). Versiunea C++ a fost dezvoltată de dr. Bjarne Stroustrup în laboratoarele AT&T Bell pentru programarea orientată spre obiecte.
- JAVA — a fost proiectat în cadrul companiei Sun Microsystems pentru aparatură electronică inteligentă conectată în rețea, pornind de la limbajul C/C++; limbajul JAVA este dedicat programării în Internet.

• LISP (LISt Processing Language), creat în 1965, și PROLOG (PROgramming LOGic), creat în 1973 — sunt limbaje dedicate rezolvării problemelor de inteligență artificială.

♦ Stiluri de programare

Evoluția limbajelor de programare a determinat formarea mai multor stiluri de programare. Stilul de programare reflectă atât modul de gândire al programatorului, cât și felul în care acesta descrie algoritmul la nivel de program.

1. Programarea nestructurată — stil „liber" de programare, fără reguli; din acest motiv, programele nestructurate au un aspect „dezordonat", fiind mai greu de urmărit și de depanat. Acest stil de programare este specific programatorilor care folosesc limbajele de programare FORTRAN, BASIC.

2. Programarea structurată — stil de programare care respectă principiul: *„orice program poate fi implementat doar prin structuri de control secvențiale, alternative sau repetitive"* (teorema de structură Böhm și Jacopini). Programele structurate pot fi realizate doar în limbajele de programare care au instrucțiuni echivalente structurilor de control. Pascal și C/C++ sunt astfel de limbaje.

3. Programarea orientată spre obiecte (OOP) — tendință nouă de programare care îmbină programarea structurată cu tehnica descrietii datelor și a prelucrărilor prin analogie cu obiectele din lumea reală. Un obiect este descris prin caracteristici și funcții, poate proveni din alt obiect sau poate genera, prin transformare, un obiect nou. Limbajele de programare Pascal și C/C++ au și versiuni OOP.

În acest manual, sunt prezentate elemente de bază ale programării structurate utile unui programator începător, cu exemplificări în limbajele Pascal și C/C++.

TEME

1. Realizați o scurtă prezentare a limbajelor de programare urmărind evoluția în timp a acestora.

2. Realizați un referat cu tema *Figuri celebre din lumea informaticii.*

3. Realizați un scurt eseu cu tema *De ce Pascal!*, în care să justificați numele acestui limbaj de programare.

1.2. Structura programelor

Indiferent de limbajul în care este scris, un program descrie datele și prelucrările unui algoritm. Opțional, pot exista și declarații tehnice prin care se solicită anumite resurse ale calculatorului (biblioteci, opțiuni de compilare, preprocesare). Structura generală a unui program realizat în limbajul Pascal, respectiv, în C/C++ este redată în tabelul 1.

Structura programelor Tabelul 1

LIMBAJUL PASCAL	LIMBAJUL C/C++
program identificator_program; *declarații opțiuni de compilare;* {$} *declarații de* **UNIT**-*uri;* (fișiere bibliotecă) **uses** crt, graph, dos; *definiții de constante;* **const** n=15; *definiții de tipuri de date* **type** sir=**array**[1..5]**of real**; *declarații de variabile;* **var** x, y : byte; *declarații de subprograme* (funcții și/sau proceduri) **begin** instrucțiuni; apeluri de subprograme; **end.**	*directive preprocesare* *includere fișiere bibliotecă header (antet)* **#include** <math.h> *definiții de constante;* **const** n=15; *definiții de tipuri de date;* **typdef float** sir[5]; *declarații de variabile;* **int** x,y; *declarații de subprograme* (funcții) ***void main()*** {instrucțiuni; apeluri de subprograme; }
Precizări: 1. Un program Pascal este un ansamblu de instrucțiuni, grupate în corpul programului principal și în subprograme, definite de programator — dacă acestea sunt necesare. 2. În orice program Pascal, pot fi folosite subprograme din unit-ul System, care nu trebuie declarat. 3. Corpul programului principal este delimitat prin **begin** și **end.** 4. Un bloc de instrucțiuni este delimitat prin **begin** și **end** 5. Fiecare instrucțiune se termină cu ; (punct virgulă).	*Precizări:* 1. Un program C/C++ este un ansamblu de instrucțiuni grupate în **funcții.** 2. Orice program C/C++ are cel puțin o **funcție** — funcția principală care se declară prin **void main ( ).** 3. Orice program C/C++ poate avea una sau mai multe funcții declarate de programator. 4. Un bloc de instrucțiuni este delimitat printr-o pereche de acolade { }. 5. Fiecare instrucțiune se termină cu ; (punct virgulă).

Exemplu:
Se citesc două numere întregi a și b; se afișează suma lor.

LIMBAJUL PASCAL	LIMBAJUL C/C++
program exemplu; **var** a, b: **integer;** **begin** {program principal} **write**(' a = '); **readln**(a); **write**(' b = '); **readln**(b); **writeln**(' Suma a + b = ', a + b); **end.**	**#include <iostream.h>** **int** a,b; **void main()** // funcția principală {**cout**<<" a = "; **cin**>>a; **cout**<<" b = "; **cin**>>b; **cout**<<"Suma a + b = "<<a + b <<endl; }

1.3. Vocabularul limbajului de programare

Vocabularul oricărui limbaj de programare este format din: setul de caractere, identificatori, separatori și comentarii.

♦ Setul de caractere

Orice program este scris cu ajutorul următoarelor caractere:

— litere mari și mici ale alfabetului englez (A-Z, a-z), numite *caractere alfabetice*;

— cifrele sistemului de numerație zecimal, numite și *caractere numerice* (0-9);

— *caractere speciale*: +, -, *, /, =, &, [,], {, }, #, |, blank (spațiu) , _, ~, @.

Codificarea și reprezentarea informației alfanumerice folosește standardul ASCII (American Standard Code for Information Interchange).

♦ Identificatori

Un identificator reprezintă o succesiune de litere, cifre sau caracterul special „_"; primul caracter nu trebuie să fie cifră. Identificatorii pot avea orice lungime.

Exemple:

1. Identificatori: a, b1, cod_0, produs.
2. Succesiuni de caractere ce nu pot fi identificatori: 3*y* (primul caracter este o cifră), *ur+m* (conține un caracter special).

Orice identificator trebuie definit sau declarat într-o linie anterioară referirii sale.

Identificatorii desemnează constante, tipuri de date, variabile. Există un set de identificatori predefiniți, numiți *cuvinte-cheie* sau *cuvinte rezervate* (tabelul 2).

Tabelul 2

Cuvinte-cheie în limbajele de programare Pascal și C/C++

LIMBAJUL PASCAL	LIMBAJUL C/C++
and, or, while, for, do, repeat, array, mod, div, trunc, begin, end, type, procedure, function, nil.	while, void, for, do, struct, char, float, switch, NULL, include, const, floor, if, define.

♦ Separatori

Unitățile sintactice (ansambluri de caractere) sunt separate între ele fie prin unul sau mai multe spații libere (blank), fie prin sfârșitul de linie (caracterul CR), fie prin caracterul ; (punct virgulă) care se utilizează pentru separarea instrucțiunilor și a declarațiilor.

♦ Comentarii

În textul unui program, sunt necesare note explicative (comentarii) atașate unor secvențe de operații, declarări de tipuri de date/variabile, care nu au un rol activ în derularea programului. Acestea sunt delimitate în limbajul Pascal prin {....}, iar în limbajul C/C++ sunt precedate de //.

TEMĂ

Se consideră următorii identificatori. Încercuiți litera corespunzătoare identificatorului corect definit. Justificați răspunsul!

a) n3; b) 4_nr; c) stea; d) p.adresa; e) nr pare; f) x+y; g) mi#sol; h) var_4.

1.4. Mediul limbajului de programare studiat

Prin codificarea algoritmului în limbajul de programare ales, obținem un program. Programul este transmis calculatorului prin introducerea textului de la tastatură (*editare*) și memorat într-un *fișier sursă* cu extensia pas pentru limbajul Pascal (exemplu: program1.pas) sau extensia cpp pentru limbajul C/C++ (exemplu: program1.cpp). Prelucrarea fișierelor sursă se face prin operațiile cunoscute: New, Open, Save, Save As.

„Traducerea" textului de program din limbajul de programare ales în limbajul intern al calculatorului se face de către *compilator* prin *compilare*. Compilatorul realizează și verificarea sintactică a programului, semnalând erorile detectate. Corectarea erorilor se face prin *editare*, de la tastatură, fiind urmată de o nouă compilare. Când textul programului nu mai conține nici o eroare, compilatorul generează un *fișier obiect* cu extensia obj (exemplu: program1.obj). Fișierul obiect este executat prin comanda Run.

Pentru a oferi programatorilor accesul la fișiere și la resursele de editare, compilare și execuție, au fost dezvoltate *mediile de programare*.

Mediul de programare este o aplicație cu meniu interactiv care oferă o interfață accesibilă și prietenoasă (user friendly) pentru operațiile necesare dezvoltării, compilării, execuției și depanării programelor (FILE, EDIT, COMPILE, RUN, DEBUG).

♦ Descrierea unei sesiuni de lucru cu mediul de programare

Pas 1. Lansarea în execuție a mediului de programare

Mediul de programare se lansează fie de pe Desktop, prin activarea icon-ului corespunzător (shortcut), fie prin lansarea în execuție a fișierului executabil din folder-ul BIN numit bp.exe pentru limbajul Pascal și bc.exe pentru limbajul C/C++. *Exemplu:* C:\bp\bin\bp.exe sau C:\borlandc\bin\bc.exe.

Pas 2. Schimbarea folderului de lucru

În folderul de lucru se vor salva programele elaborate pe durata unei sesiuni de lucru. Pentru schimbare se alege File → Change directory... (figura 2).

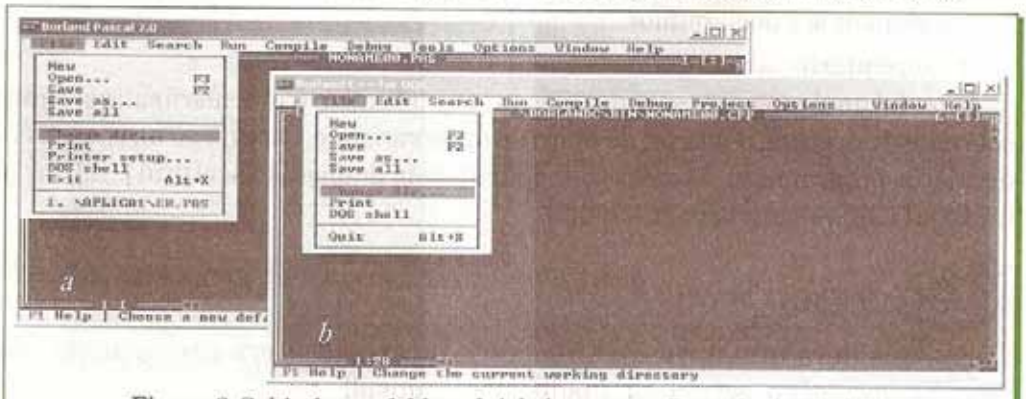


Figura 2. Schimbarea folder-ului de lucru: *a* — Pascal; *b* — C/C++

Pas 3. Accesul la fișiere

Crearea unui nou fișier sursă se realizează alegând calea File → New, iar deschiderea unui fișier sursă existent, alegând File → Open. În același mod, pot fi create/deschise și fișiere text cu date de intrare pentru testarea programelor.

Pas 4. Salvarea

Salvarea fișierului în folderul ales se realizează prin File → Save (sau apăsând tasta funcțională F2). Dacă se dorește redenumirea fișierului, se alege File → Save as ...

Pas 5. Editarea programului

Editarea liniilor de program (figura 3) se face de la tastatură. Nu uitați să salvați permanent cu tasta F2!

Figura 3. Editarea liniilor de program: *a* — Pascal; *b* — C/C++

Pas 6. Compilarea

Compilarea liniilor de program (figura 4) se realizează alegând calea Compile → Compile sau prin activarea simultană a tastelor ALT+F9. Nu uitați să salvați permanent cu tasta F2!

Figura 4. Compilarea: *a* — Pascal; *b* — C/C++

Pas 7. Lansarea în execuție

Lansarea în execuție a programului (figura 5) se realizează alegând calea Run → Run sau prin activarea simultană a tastelor CTRL+F9.

Dacă, după execuție, se observă că rezultatele obținute nu sunt

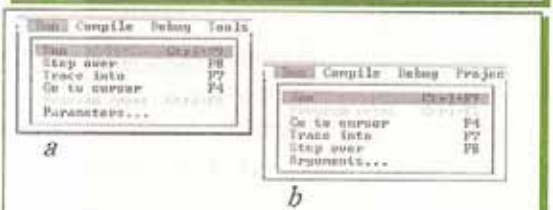


Figura 5. Lansarea în execuție: *a* — Pascal; *b* — C/C++

corecte, se verifică algoritmul de rezolvare a problemei; se reiau pașii 5-7 până când programul furnizează date de ieșire corecte.

Pentru a înțelege cum sunt executate liniile de program (instrucțiuni, operații, apeluri de subprograme etc.), se poate executa programul pas cu pas din opțiunea Run → Trace into (F7) sau Run → Step over (F8).

În tratarea erorilor de concepție a programului, este utilă opțiunea din meniul sistem Debug → Add Watches (în ferestrele de dialog sunt trecuți identificatorii variabilelor ale căror modificări interesează la rularea programului).

Pas 8. Închiderea și părăsirea mediului de programare

Închiderea și părăsirea mediului de programare se realizează alegând combinația de taste ALT + X sau: File → Exit pentru Pascal și File → Quit pentru C/C++.

TEME

Deschideți o sesiune de lucru în mediul de programare al limbajului studiat și rezolvați următoarele cerințe:

1. Editați un fișier sursă cu textul de program de la pasul 5 și executați operațiile descrise la pașii 6 și 7.

2. Inspectați meniul mediului de programare și descrieți succint opțiunile și tastele funcționale.

3. Explicați semnificația următoarelor operații: editare, compilare, execuție.

4. Creați și salvați fișierul text date.in cu date personale (note, agendă).

2. DESCRIEREA ȘI PRELUCRAREA DATELOR

2.1. Tipuri standard de date

Datele prelucrate de un program corespund datelor identificate în etapa de analiză a problemei. În schema din figura 6, este prezentată clasificarea datelor după natura și stabilitatea lor.

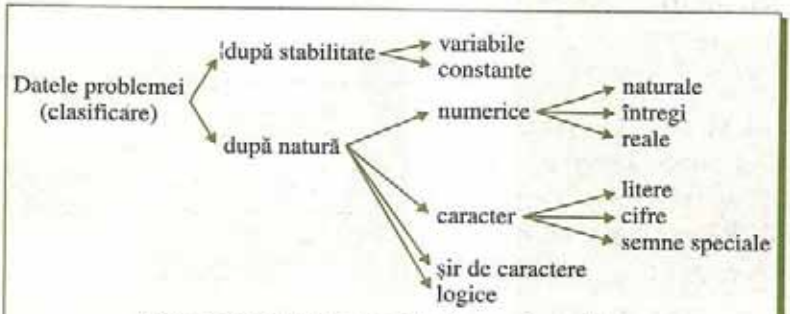


Figura 6. Clasificarea datelor unei probleme

Un tip de dată este caracterizat prin mulțimea de valori și dimensiunea zonei de memorie alocată (numărul de octeți).

Pentru început, vor fi studiate doar câteva tipuri de date necesare unui programator începător.

Din punct de vedere al complexității, există două categorii de date (tabelul 3):

➢ date simple (sau elementare);

➢ date structurate (sau compuse) — se obțin din tipurile simple prin:

a) compunerea, în memoria internă, a unui număr finit de elemente de același tip (tablouri de date);

b) compunerea, în memoria externă, a unui număr nelimitat de componente (fișiere).

Tabelul 3

Tipuri standard de date

Tipuri de date	LIMBAJUL PASCAL		LIMBAJUL C/C++	
	Date	Număr de octeți	Date	Număr de octeți
Tipuri simple	• întregi — **integer**	2	• întregi — **int**	2
	— **shortint**	1	— **shortint**	2
	— **longint**	4	— **unsigned int**	2
	— **byte**	1	— **long**	4
	— **word**	2	— **unsigned long**	4
	• reale — **real**	6	• reale — **float**	4
	— **single**	4	— **double**	8
	— **double**	8	— **long double**	10
	— **extended**	10		
	• caracter — **char**	1	• caracter — **char**	1
	• logice — **boolean**	1	*Precizare:* Orice valoare de tip întreg diferită de zero are semnificația de valoare logică **adevărat**; zero semnifică valoarea logică **fals**.	
Tipuri structurate	• tablouri de date (vectori, șiruri)		• tablouri de date (vectori, șiruri)	
	• fișiere text — **text**		• fișiere text — **fstream**	

Dimensiunea zonei de memorie limitează domeniul de valori. De exemplu: pentru întregi cu semn avem [–32786, 32767], iar pentru întregi fără semn, [0, 65535].

2.2. Constante, variabile, expresii

2.2.1. Constante și variabile

Valorile atribuite *constantelor* nu se pot modifica prin operațiile sau instrucțiunile din program. Tipurile de constante și definirea acestora sunt prezentate în tabelul 4.

Zonele de memorie al căror conținut poate fi modificat în timpul execuției programului se numesc *variabile*. O variabilă poate fi declarată pentru orice tip standard de date.

O variabilă poate primi valori prin citire direct de la tastatură.

O variabilă poate fi afișată prin scriere pe ecranul calculatorului.

Tabelul 4

Tipuri de constante/definirea constantelor și a variabilelor

LIMBAJUL PASCAL	LIMBAJUL C/C++
Tipuri de constante	
• **întregi** — numere întregi *pozitive* care nu pot depăși valoarea 32767, valoare identificată de compilator sub numele de MAXINT; valorile întregi pot fi numere: — *zecimale* (baza 10); Exemple: 43, 187, 2001. — *hexazecimale* (baza 16), care sunt precedate de caracterul special $. Exemplu: $2F4 sau $2F4 → $2F4_{(16)}$. • **reale** — numere reale cu valori absolute cuprinse între 5.9×10^{-39} și 3.4×10^{38}; atât partea întreagă, cât și cea zecimală trebuie să conțină cel puțin o cifră, chiar dacă aceasta este 0. • **caracter** — orice caracter scris între apostrofuri. Exemplu: 'A' • **șiruri de caractere** — o succesiune de caractere, delimitată tot prin apostrofuri. • **constante definite prin cuvinte-cheie** Exemple: true/false, MAXINT.	• **întregi** — numere întregi *pozitive* cu valori între 0 și 4.294.967.295 și care sunt de trei tipuri: — *zecimale* (baza 10); Exemple: 43, 152, 7564. — octale (baza 8), care sunt precedate de un zero nesemnificativ; Exemplu: 0354 → $345_{(8)}$. — *hexazecimale* (baza 16), care sunt precedate de 0x sau 0X. Exemplu: 0x2F4 sau 0X2F4 sau 0x2f4 - → $2F4_{(16)}$. • **reale** — orice valoare reală Exemple: 32.45, 0.5. • **caracter** — orice caracter scris între apostrofuri. Exemplu: 'A' • **șiruri de caractere** — o succesiune de caractere, delimitată tot prin ghilimele. • **constante definite prin cuvinte-cheie** Exemplu: NULL
Definirea constantelor și a variabilelor	
const id_constanta = valoare; **var** id_variabilă: tip standard; Exemplu: **const** pi = 3.14; **const** unu = '1'; **var** s : **integer**; p : **real**;	**const** [tip] id_constantă = valoare; tip standard id_variabilă; Exemplu: **const** pi = 3.14; **const** unu = '1'; **int** s; **float** p;

TEME

Stabiliți tipul constantelor din tabelul de mai jos.

LIMBAJUL PASCAL	132	$5BF	–2.34	512E+23	'Succesiune'	MAXLONGINT	'A'
LIMBAJUL C/C++	132	0x5BF	–2.34	512E+23	"Succesiune"	NULL	'A'

2.2.2. Expresii

O *expresie* este formată din unul sau mai mulți operanzi (constante, variabile, funcții), legați prin opratori. O expresie are o valoare și un tip.

În evaluarea unei expresii, ordinea efectuării operațiilor respectă regulile de prioritate ale operatorilor. Pentru a schimba o prioritate se utilizează paranteze rotunde. În tabelele 5 și 6, sunt prezentate principalele categorii de operatori și prioritatea acestora, corespunzător fiecărui limbaj de programare.

Tabelul 5

Tipuri de operatori

LIMBAJUL PASCAL	LIMBAJUL C/C++
1. Operatori aritmetici: — unari: +, - — binari multiplicativi: *, /, **div** (cât), **mod** (rest) — binari aditivi: +, - 2. Operatori relaționali: <, <=, >, >= 3. Operatori de egalitate: = (egal), <> (diferit) 4. Operatori logici: **not** (negare logică, operator unar), **and** (*și* logic), **or** (*sau* logic), **xor** (*sau exclusiv*)	1. Operatori aritmetici: — unari: +, - — binari multiplicativi: *, /, % (rest) — binari aditivi: +, - 2. Operatori relaționali: <, <=, >, >= 3. Operatori de egalitate: == (egal), != (diferit) 4. Operatori logici: ! (negare logică, operator unar), && (*și* logic), || (*sau* logic) 5. Operatori logici pe biți: ~ (complement față de 1, operator unar), << >> (deplasare pe biți la stânga/la dreapta), & (*și* pe biți), ^ (*xor* – sau exclusiv pe biți), | (*sau* pe biți)
not(0) → 1 not(1) →0 **and** 0 1 0 0 0 1 0 1 **or** 0 1 0 0 1 1 1 1 **xor** 0 1 0 0 1 1 1 0	~ 0 → 1 ~ 1 → 0 & 0 1 0 0 0 1 0 1 | 0 1 0 0 1 1 1 1 ^ 0 1 0 0 1 1 1 0
5. Operatori de atribuire: : = id_v : = valoare/expresie	6. Operatori de atribuire: id_v = valoare/expresie Pentru operația de atribuire se pot utiliza și următoarele asocieri de operatori: *=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=
6. Operatorii de incrementare/decrementare: **inc**(id_v) ⇔ id_v : = id_v + 1 **dec**(id_v) ⇔ id_v : = id_v – 1 unde id_v este o variabilă de tip întreg.	7. Operatorii de incrementare/decrementare: postfixat: id_v++ / id_v- - prefixat : ++id_v / -- id_v Dacă operatorul este postfixat, atunci variabila este incrementată/decrementată după evaluarea expresiei din care face parte; dacă operatorul este prefixat, atunci variabila este incrementată/ decrementată înainte de evaluarea expresiei.
7. Operatorul **sizeof(expresie)/ sizeof(tip)** returnează numărul de octeți alocați pentru memorarea unei expresii sau a unui tip de dată.	8. Operatorul **sizeof(expresie)/ sizeof(tip)** returnează numărul de octeți alocați pentru memorarea unei expresii sau a unui tip de dată. 9. Operatorul condițional: e1 ? e2 : e3 unde e1, e2, e3 sunt expresii. Dacă e1 are valoare nenulă, atunci se prelucrează e2, altfel, se prelucrează e3. 10. Operatorul de conversie explicită **(tip)operand** convertește valoarea operandului la tipul indicat.

Tabelul 6

Operatori — reguli de prioritate și evaluare

LIMBAJUL PASCAL			LIMBAJUL C/C++		
Prioritate	**Operator**	**Evaluare**	**Prioritate**	**Operator**	**Evaluare**
1	()	s→d	1	()	s→d
2	**not** + - **sizeof()**	d→s	2	! ~ + - ++ — **sizeof()**	d→s
3	* / **div mod**	s→d	3	* / %	s→d
4	+ -	s→d	4	+ -	s→d
5	< <= > >=	s→d	5	< <= > >=	s→d
6	= <>	s→d	6	(egal) == != (diferit)	s→d
7	**and** (*și* logic)	s→d	7	& (și pe biți)	s→d
8	**xor**	s→d	8	^ (**XOR** pe biți)	s→d
9	**or** (*sau* logic)	s→d	9	| (**OR** pe biți)	s→d
10	:= (atribuire)	s→d	10	**&&** (*și* logic)	s→d
			11	|| (*sau* logic)	s→d
			12	= (atribuire)	d→s

Exemplu:
Avem următoarele expresii:

LIMBAJUL PASCAL	LIMBAJUL C/C++
expresii numerice — determinantul ecuației de gradul II	
D := b * b – 4 * a * c	D = b * b – 4 * a * c
expresii logice — x ∈ [a, b]	
(x >= a) and (x <= b)	(x >= a) && (x <= b)
	expresii la nivel de bit
	a = 3 b = 2 c = a & b = 2 a → 0 1 1 b → 0 1 0 a & b → 0 1 0 → 2

TEME

1. Scrieți, în limbajul de programare studiat, următoarele expresii:
   E1 = „x este număr par și y nu se divide la 3, 5 și 7"
   E2 = „x este mai mic sau cel mult egal cu y și y este multiplu de 11 și 9".
2. Evaluați următoarele expresii pentru a = 5, b = 2, c = 3:
   E1 = a + b/2 + c * a + b
   E2 = -c + b * a + (c * b / a + b + c) / (a * b).

3. Fie numere reale a, b, c, d, și x, unde a < b și c < d. Identificați expresia corectă pentru ca x ∈ [a, b) sau x ∈ (c, d].

LIMBAJUL PASCAL	LIMBAJUL C/C++
a) (x>=a or x <b) and (x>c or x<=d) b) ((x>=a) and (x<b)) or ((x>c) and (x<=d)) c) (x>=a or x <b) or (x>c or x<=d) d) ((x>=a) and (x<=b)) or ((x>=c) and (x<=d))	a) (x>=a || x <b) && (x>c || x<=d) b) ((x>=a) && (x<b)) || ((x>c) && (x<=d)) c) (x>=a || x <b) || (x>c || x<=d) d) ((x>=a) && (x<=b)) || ((x>=c) && (x<=d))

2.3. Citirea și scrierea datelor

Operațiile de citire a datelor de intrare și de scriere a datelor de ieșire se pot realiza prin două căi:

1. de la tastatură/pe ecran; simbolizate prin (1) pe figura 7;
2. din fișiere text/în fișiere text; simbolizate prin (2) pe figura 7.

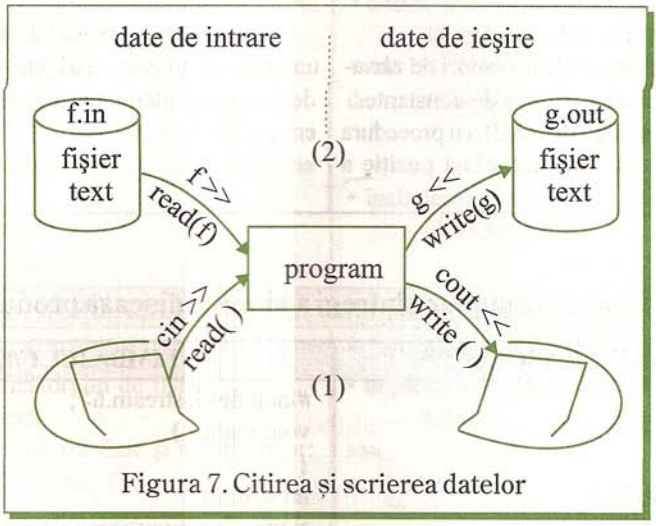


Figura 7. Citirea și scrierea datelor

Pentru fiecare tip de operație de citire sau scriere, există comenzi specifice implementate în bibliotecile limbajelor de programare.

Biblioteca limbajului Pascal se află în unit-ul System.

Limbajul C/C++ are mai multe astfel de biblioteci; în manual, vom folosi iostream.h pentru operațiile standard și fstream.h pentru fișiere.

2.3.1. Operații cu tastatura și ecranul

Operațiile de citire a datelor de intrare de la tastatură și de scriere (afișare) a celor de ieșire pe ecran sunt prezentate în tabelul 7. Valorile afișate pot fi cele reținute în zonele de memorie sau cele rezultate din evaluarea unor expresii.

Tabelul 7

Operații cu dispozitivele standard

LIMBAJUL PASCAL	LIMBAJUL C/C++
CITIREA DE LA INTRAREA STANDARD (TASTATURĂ) se face cu:	
procedurile **read** și **readln** ***read***(id_v1[, idv2,.., id_vn]); ***readln***(id_v1[, idv2,.., id_vn]); unde id_v1, id_v2, ..., id_vn sunt identificatori de variabile elementare de orice tip, cu excepția celor logice. Procedura **readln** are același rol cu cel al procedurii **read**, cu diferența că, la terminarea citirii, cursorul este poziționat pe linia următoare a ecranului.	funcția **cin** #include <iostream.h>; ***cin***>>id_v1[>>id_v2>>...>>id_vn]; unde id_v1, id_v2, ..., id_vn sunt identificatori de variabile elementare de orice tip.
SCRIEREA (AFIȘAREA) LA IEȘIREA STANDARD (ECRAN) se face cu:	
procedurile **write** și **writeln** **write** (id1[, id2,.., idn]); **writeln** (id1[, id2,.., idn]); unde id1, id2, ..., idn sunt identificatori de variabile elementare de orice tip sau de constante. La terminarea scrierii (afișării), cu procedura **writeln**, cursorul este adus pe prima poziție a rândului următor.	funcția **cout** #include<iostream.h>; **cout**<<id_v1[<<id_v2<<..<<id_vn]; unde id_v1, id_v2, ..., id_vn sunt identificatori de variabile elementare de orice tip sau de constante.

Exemplu:

Se citesc două numere întregi a și b; se afișează produsul lor.

LIMBAJUL PASCAL	LIMBAJUL C/C++

program produs;
var a, b, p: integer;
begin
write ('a='); **readln** (a);
write ('b='); **readln** (b);
p:= a * b;
writeln ('a*b=', p);
end.
	#include<iostream.h>;
	void main ( )
	{
	int a, b, p;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;
	p=a*b;
	cout<<"a*b="<<p;
	}

2.3.2. Operații cu fișiere text

Datele cu care lucrează un program pot fi memorate în fișiere text. Astfel se asigură păstrarea datelor și reducerea timpului de operare la testarea/utilizarea programului.

Un *fișier text* este format dintr-o succesiune de caractere ASCII scrise pe una sau mai multe linii (rânduri), pe memorie externă (disc). Sfârșitul fișierului este

marcat de o „etichetă" specială: EOF (End Of File). Prelucrarea unui fișier se face prin mai multe operații: *deschidere* (pentru citire sau scriere), *citire*, *scriere*, *închidere* (tabelul 8). La nivel de program, fișierul este recunoscut printr-o variabilă asociată, numită *identificator de fișier* (id_f).

Tabelul 8

Operații cu fișiere de text

LIMBAJUL PASCAL	LIMBAJUL C/C++
CITIREA DATELOR DIN FIȘIERE TEXT are următoarele etape:	
• definirea identificatorului de fișier ***var*** id_f:***text***; • asocierea dintre fișierul fizic și identificator: ***assign***(id_f,'nume_f'); • deschiderea fișierului pentru citire: ***reset***(id_f); • citirea variabilelor din fișier se face cu ajutorul procedurilor: ***read*** și ***readln***. ***read***(id_f,id_v1[, idv2,.., id_vn]); ***readln***(id_f,id_v1[, idv2,.., id_vn]); unde id_v1, id_v2, ..., id_vn sunt identificatori de variabile elementare de orice tip, cu excepția celor logice. • închiderea fișierului: ***close***(id_f).	#include<fstream.h> • implementarea fișerelor de tip text: **fstream** id_f([cale\\]nume_f,ios::in); sau **ifstream** id_f("nume_f"); Prin implementare, fișierul este deschis pentru citire. • citirea se face astfel: ***id_f***>>id_v1[>>id_v2>>...>>id_vn]; unde id_v1, id_v2, ..., id_vn sunt identificatori de variabile elementare de orice tip. • închiderea fișierului : id_f.**close**();
SCRIEREA ÎN FIȘIERE TEXT are următoarele etape:	
• definirea identificatorului de fișier ***var*** id_f:***text***; • asocierea dintre fișierul fizic și identificator: ***assign***(id_f,'nume_f'); • deschiderea fișierului pentru scriere: ***rewrite***(id_f); • scrierea variabilelor în fișier se face cu ajutorul procedurilor **write** și **writeln**: ***write***(id_f, id1[, id2,.., idn]); ***writeln***(id_f, id1[, id2,.., idn]); unde id1, id2, ..., idn sunt identificatori de variabile elementare de orice tip sau de constante. • închiderea fișierului: ***close***(id_f);	#include<fstream.h> • implementarea fișerelor de tip text: **fstream** id_f([cale\\]nume_f,ios::out); sau **ofstream** id_f([cale\\]"nume_f"); Prin implementare, fișierul este deschis pentru scriere. • scrierea în fișier se face astfel: ***id_f***<<id_v1[<<id_v2<<..<<id_vn]; unde id_v1, id_v2, .., id_vn sunt identificatori de variabile elementare de orice tip sau de constante. • închiderea fișierului: id_f.**close**();

Un fișier text poate fi creat direct din mediul de programare File → New, salvat cu extensia dorită (exemplu: date.in) și folosit ("citit") în program.

Exemplu:

Din fișierul date.in (creat în directorul curent), se citesc variabilele reale a, b și c, separate prin câte un spațiu.

În fișierul date.out se vor afișa valorile citite, pe prima linie, cu câte un spațiu între ele, iar pe următoarele linii, suma și produsul lor.

date.in	date.out
3 4 5	3 4 5 12 60

LIMBAJUL PASCAL	LIMBAJUL C/C++

var f , g : text;
a, b, c , s, p : real;
begin
assign(f , 'date.in'); reset(f);
assign(g , 'date.out'); rewrite(g);
read(f , a , b , c);
s: = a + b + c ; prod : = a*b*c;
writeln(g, a, ' ',b,' ',c); writeln(g, 'Suma = ',s);
writeln(g, ' Produsul = ',p);
close(f); close(g);
end.
	#include<fstream.h>
	ifstream f("date.in");
	ofstream g("date.out");
	void main()
	{float a, b, c, s, p;
	f>>a>>b>>c;
	s=a+b+c; p=a*b*c;
	g<<a<<" "<<b<<" "<<c<<endl;
	g<<"Suma = "<<s<<endl;
	g<<"Produsul= "<<p<<endl;
	f.close(); g.close();
	}

TEME

1. Analizați exemplul corespunzător limbajului de programare studiat și explicați operațiile de lucru cu fișiere.

2. Modificați secvența din exemplu astfel încât introducerea datelor să se facă de la tastatură.

3. Modificați secvența din exemplu astfel încât scrierea datelor să se facă pe ecran.

3. Creați fișierul date.in și testați programul. Pentru verificarea rezultatelor, deschideți fișierul date.out.

2.4. Funcții matematice uzuale

Limbajele de programare au biblioteci în care se află subprograme pentru determinarea valorilor celor mai frecvente funcții matematice. Biblioteca limbajului Pascal se află în unit-ul System, iar cea a limbajului C/C++, în fișierul header <math.h>.

Pentru funcțiile matematice prezentate în manual (tabelul 9), sunt definite tipurile de date ale domeniilor de definiție și tipurile de date ale mulțimilor de valori.

Tabelul 9

Funcții matematice

LIMBAJUL PASCAL		LIMBAJUL C/C++	
		#include<math.h>	
abs : R → R_+	**abs**(x) — modulul lui x	abs : Z → N	**abs**(x) — modulul lui x
abs : Z → N		(vezi labs, fabs)	
sqr : R → R_+	**sqr**(x) — x la puterea a doua	pow : R → R_+	**pow**(x,2) — x la puterea a doua
sqrt : R_+ → R_+	**sqrt**(x) — radical din x	sqrt : R_+ → R_+	**sqrt**(x) — radical din x
sin : R → [-1,1]	**sin**(x), **cos**(x), **arctan**(x) —	sin : R → [-1,1]	**sin**(x), **cos**(x), **arctan**(x) —
cos : R → [-1,1]	funcții trigonometrice	cos : R → [-1,1]	funcții trigonometrice
arctan : R → R		atan : R → R	
ln : R_+ → R	**ln**(x) — logaritm natural	log : R_+ → R	**log**(x) — logaritm natural (ln x)
		log10 : R_+ → R	**log10**(x) — logaritm în baza 10 (lg x)
exp : R → R_+	**exp**(x) — e^x (e = 2,71 constanta Euler)	exp : R → R_+	**exp**(x) — e^x (e = 2,71 constanta Euler)
int : R → Z	**int**(x) — partea întreagă din x	ceil : R → Z	**ceil**(x) — rotunjește la cel mai apropiat întreg mai mare decât x
trunc : R → Z	**trunc**(x) — rotunjește la cel mai apropiat întreg mai mic decât x		
round : R → Z	**round**(x) — rotunjește la cel mai apropiat întreg mai mare decât x	floor : R → Z	**floor**(x) — rotunjește la cel mai apropiat întreg mai mic decât x
frac : R → (-1,1)	**frac**(x) — returnează partea fracționară a lui x	pow : R × R → R	**pow**(x,y) — calculează x^y

TEME

1. Stabiliți ce afișează fiecare dintre instrucțiunile din tabelul alăturat.

LIMBAJUL PASCAL	LIMBAJUL C/C++

writeln(round (5.62)) ;
writeln(trunc(4.32)) ;
writeln(round(-7.28));
writeln(trunc(-4.33));
writeln(abs(-14));
	cout<<ceil(5.62)<<endl;
	cout<<floor(4.32)<<endl;
	cout<<ceil(-7.28)<<endl;
	cout<<floor(-4.33);
	cout<<abs(-14)<<endl;

2. Transformați fiecare instrucțiune într-o secvență care să cuprindă declararea, citirea și scrierea datelor, ca în exemplul din tabelul alăturat.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var x: **real;** **read**(x); **write**(**round** (x));	**float** x; **cin**>>x; **cout**<<**ceil**(x);

3. INSTRUCȚIUNI PENTRU CODIFICAREA STRUCTURILOR DE CONTROL

3.1. Structuri liniare

Structura liniară este o secvență de instrucțiuni care se execută necondiționat, o singură dată. O astfel de structură poate să conțină instrucțiuni pentru citirea și scrierea datelor, calcule matematice (expresii) — *instrucțiuni de atribuire.*

Prin atribuire, o variabilă primește valoarea unei alte variabile, a unei valori sau a unei expresii. Instrucțiunea de atribuire este reprezentată prin operatorul de atribuire specific limbajului de programare (tabelul 10).

Tabelul 10

Sintaxa instrucțiunii de atribuire

LIMBAJUL PASCAL	LIMBAJUL C/C++
id_variabila := valoare/expresie; *Precizare:* Tipul variabilei trebuie să coincidă cu tipul valorii/expresiei.	id_variabila = valoare/expresie; *Precizare:* Atribuirea este precedată de conversia implicită a valorii/expresiei la tipul variabilei.

Exemplu:
Se interschimbă conținutul a două variabile întregi u și v.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var u, v, aux : integer;
begin
u := 5; v := 9; {atribuire/inițializare cu valori}
u := u + v; {atribuirea valorii unei expresii}
v := u - v; {atribuirea valorii unei expresii}
u := u - v; {atribuirea valorii unei expresii}
writeln('u = ', u, ' v = ', v)
end.
	#include<iostream.h>
	void main()
	{ int u = 5, v = 9 ; //atribuire/inițializare cu valori
	u = u + v; {atribuirea valorii unei expresii}
	v = u - v; {atribuirea valorii unei expresii}
	u = u - v; {atribuirea valorii unei expresii}
	cout<<" u = "<<u<<" v = "<<v;
	}

TEME

1. De la intrarea standard se citesc trei valori reale x, y și z. Să se scrie un program, pentru afișarea, pe ecran, a valorilor următoarelor expresii:

$$A = 5x + 3B - 21y + z^4; \quad B = C - 3x + 5y - 45z; \quad C = \frac{12x+1}{y^2+2} - \sqrt{z^2+27}.$$

2. Se consideră două măsuri de unghiuri exprimate în grade, minute și, respectiv, secunde: g1, m1, s1; g2, m2, s2. Să se calculeze și să se afișeze suma măsurilor celor două unghiuri, exprimată tot în grade, minute, respectiv, secunde. Datele de intrare se citesc din fișierul text unghi.in (informațiile fiecărui unghi sunt citite de pe linii distincte, valorile fiind separate printr-un spațiu). Măsura unghiului sumă se va afișa în fișierul unghi.out.

Exemplu:

unghi.in	unghi.out
2 45 55	15° 41′ 40″
12 55 45	

3. Fie s un număr natural nenul ce reprezintă un interval de timp exprimat în secunde. Să se transforme acest interval de timp în ore, minute și secunde.

Exemplu: s = 4536 → 1 oră 25 min 36 secunde

4. Din fișierul numar.in se citește un număr natural nenul, format exact din trei cifre. Realizați un program prin care numărul și suma cifrelor sale să fie afișate pe linii distincte, în fișierul numar.out.

5. De la tastatură se citesc coordonatele punctelor A(x1,y1), respectiv B(x2,y2). Să se afișeze lungimea segmentului AB și coordonatele mijlocului segmentului AB.

6. Un produs are prețul unitar de x lei, la care se aplică TVA în valoare de t% (x și t sunt numere naturale și sunt citite de la tastatură). Să se afișeze, pe ecran, prețul de vânzare.

7. Un cerc are raza r (număr real citit de la tastatură). Să se determine lungimea și aria cercului și lungimea sectorului de cerc care subîntinde un unghi de 40°. Datele de ieșire vor fi afișate pe linii destincte în fișierul cerc.out.

3.2. Structuri alternative

3.2.1. Instrucțiunea if

Structurile alternative dirijează execuția unei secvențe de instrucțiuni (s 1 sau s 2) în funcție de valoarea unei condiții plasate în blocul de decizie. Aceste structuri se codifică prin instrucțiunea if (figura 8 și tabelul 11). Dacă secvența s 1 sau s 2 conține mai multe instrucțiuni, acestea sunt introduse într-un bloc de instrucțiuni. *Dacă secvența s2 este vidă (structura pseudoalternativă),* instrucțiunea if nu are ramura else.

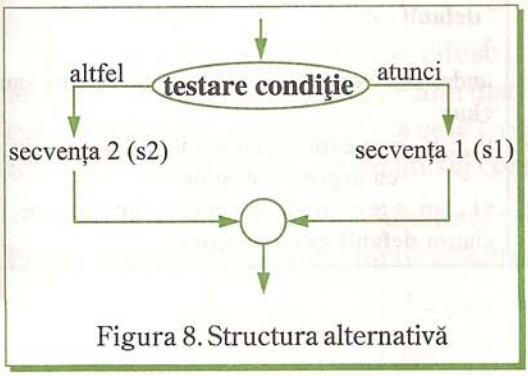


Figura 8. Structura alternativă

Tabelul 11

Sintaxa instrucțiunii if

LIMBAJUL PASCAL	LIMBAJUL C/C++
if cond_logica **then** s1 **else** s2;	**if** (cond_logica) s1; **else** s2;

Exemplu:
Rezolvarea ecuației de gradul I, de forma $ax + b = 0$, cu coeficienți reali.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a,b,x:real;
begin
write(' a = '); readln(a); write(' b = '); readln(b);
if a<>0 then
begin
x := -b/a;
write('ec. compatibila determinata x = ',x)
end
else
if b<>0 then
write('ec. incompatibila')
else
write('ec. compatibila nedeterminata');
end.
	#include<iostream.h>
	void main()
	{ float a,b, x;
	cout<<" a = ";cin>>a; cout<<" b = ";cin>>b;
	if(a!=0)
	{ x=-b/a;
	cout<<" ec. compatibila determinata x = "<<x;
	}
	else
	if(b!=0)
	cout<<" ec. incompatibila ";
	else
	cout<<" ec. compatibila nedeterminata";
	}

3.2.2. Instrucțiunea de selecție

Structura alternativă de selecție dirijează execuția unei secvențe de instrucțiuni (s1, s2, ..., sn) în funcție de valoarea unui *selector* (tabelul 12).

Tabelul 12

Sintaxa instrucțiunii de selecție

LIMBAJUL PASCAL	LIMBAJUL C/C++
case selector **of** v_{11}[,v_{12}..v_{1n}] : s1; v_{21}[,v_{22}..v_{2n}] : s2; ... v_{m1}[,v_{m2}..v_{mn}] : sm **else** s **end**; unde: **selector** este o expresie de tip **integer** sau **char** v_{11},...v_{mn} sunt expresii constante de același tip cu expresia **selector** s1,..sm, s reprezintă secvențe de instrucțiuni clauza **else** este opțională	**switch** (selector) { **case** v_1 : s1; **break**; **case** v_2 : s2 ; **break**; ... **case** v_m : sm; **break**; **default** : s; } unde: **selector** este o expresie de tip **int** sau **char** v_1,...v_m sunt expresii constante de același tip cu expresia **selector** s1,..sm, s reprezintă secvențe de instrucțiuni clauza **default** este opțională

Exemplu:
Se citesc de la tastatură două numere întregi x și y. Programul realizează una dintre următoarele operații aritmetice, potrivit opțiunii utilizatorului: suma, diferența, produsul, câtul împărțirii întregi.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a, b, selector : integer;
begin
write('a= ');readln(a);write('b = ');readln(b);
writeln('Tastati una dintre cifre ');
writeln('1 – suma / 2 — diferenta / 3 — produs / 4 – cat');
write('selector = ');readln(selector);
case selector **of**
1 : write(' suma = ',a+b);
2 : write(' diferenta = ',a-b);
3 : write(' produsul = ',a*b);
4 : write(' catul impartirii intregi = ',a div b);
else write(' Ati tastat o optiune inexistenta');
end;
end.
	#include<iostream.h>
	void main()
	{int a, b, selector;
	cout<<" a = ";cin>>a; cout<<" b = ";cin>>b;
	cout<<" Tastati una dintre cifre "<<endl;
	cout<<"1– suma / 2 — diferenta / 3— produs / 4 — cat ";
	cout<<" selector = ";cin>>selector;
	switch (selector)
	{**case** 1: cout<<" suma = "<<a+b;break;
	case 2 : cout<<"produsul = "<<a*b;break;
	case 3 : cout<<"diferenta = "<<a-b;break;
	case 4 : cout<<"catul = "<<a/b;break;
	default:cout<<"Ati tastat o optiune inexistenta";
	} }

TEME

1. De la intrarea standard (tastatură) se citesc trei numere reale a, b și c. Dacă cele trei numere pot forma laturile unui triunghi, să se afișeze pe ecran un mesaj corespunzător: a) echilateral; b) isoscel; c) dreptunghic.

2. Din fișierul max_min.in se citesc patru numere întregi x, y, u și v, separate prin câte un spațiu. Să se determine maximul și minimul valorilor citite și apoi să se scrie aceste valori în fișierul max_min.out, pe linii distincte.

Exemplu:

max _min.in	max_min.out
12 -24 5 3	maximul = 12
	minimul = -24

3. De la intrarea standard se citesc trei numere reale m, n și p. Să se calculeze și să se afișeze pe ecran, pe linii distincte, media aritmetică și media geometrică a celor trei numere. Dacă cele trei numere pot forma un triplet pitagoric ($c^2 = a^2 + b^2$), să se afișeze un mesaj corespunzător.

4. Să se afișeze valoarea funcției $f: \mathbb{R} \to \mathbb{R}$ în punctul $x = b$ (b este o valoare introdusă de la tastatură), funcția având următoarea lege de corespondență:

$$f(x) = \begin{cases} x^2 - 6x, & \text{dacă } x < -12 \\ \sqrt{x^4 + 12}, & \text{dacă } -12 \le x < -5 \\ 2x + 12, & \text{dacă } -5 \le x < 2 \\ 14, & \text{altfel} \end{cases}.$$

5. a) Ce afișează următorul program?

LIMBAJUL PASCAL	LIMBAJUL C/C++

{1} var a, b, x, y, z, t : real;
{2} begin
{3} x:=29; y:=13; z:=4.5; t:=2.3;
{4} a:= x/y;
{5} b:=z/t;
{6} if a=b then write('Valori egale')
else write('Valori diferite')
{7} end.
	#include<iostream.h> // 1
	void main() // 2
	{ float x=29, y=13, z=4.5, t=2.3; // 3
	float a, b; // 4
	a = x/y; // 5
	b = z/t ; // 6
	if(a==b)cout<< "Valori egale "; // 7
	else cout<< "Valori diferite "; }

b) ➢Dar în cazul înlocuirii liniilor 4 și 5 cu liniile {4'} *a: = x div y*
{5'} *b := z div t* ?

➢Dar în cazul înlocuirii liniei 4 cu linia {4'} *int a, b* ?

6. Se consideră două cercuri date fiecare prin coordonatele centrului (x, y) și rază (r). Să se afișeze, pe ecran, poziția celor două cercuri unul față de celălalt. Informațiile despre cercuri se citesc de la tastatură.

3.3. Structuri repetitive

Dacă, într-un algoritm, se repetă o secvență de operații, atunci se utilizează structuri repetitive cu număr cunoscut de pași (*cu contor*) sau cu număr necunoscut de pași (*cu condiție*).

3.3.1. Structuri repetitive cu contor

Structura repetitivă *cu contor* (cu număr cunoscut de pași) este codificată prin instrucțiunea for (tabelul 13).

Tabelul 13

Sintaxa instrucțiunii for

LIMBAJUL PASCAL	LIMBAJUL C/C++
for crescător **for** contor:=val_i **to** val_f **do** S; for descrescător **for** contor:=val_i **downto** val_f **do** S; unde **contor** este o variabilă ce pornește de la o valoare inițială **val_i** și se modifică crescător/ descrescător până la o valoare finală **val_f**. Dacă secvența (S) conține mai multe instrucțiuni, atunci acestea sunt cuprinse între **begin** și **end**	**for** (e1; e2; e3) S; unde e1, e2, e3 sunt expresii de forma: e1 — inițializare variabilă contor **contor** = valoare_inițială e2 — condiția de continuare **contor** <= valoare_finală **for** crescător **contor** >= valoare_finală **for** descrescător e3 — modificare contor **contor** ++ **for** crescător **contor** -- **for** descrescător Dacă secvența (S) conține mai multe instrucțiuni, atunci acestea sunt cuprinse între acolade {}.

Exemplu:

Se afișează pe ecran primele n numere naturale astfel: pe prima linie, în ordine crescătoare, iar pe următoarea linie, în ordine descrescătoare. Numărul n se citește de la tastatură.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var i, n : integer;
begin
write('n = ');readln(n);
for i:=1 **to** n **do**
write(i ,' ');
writeln;
for i := n **downto** 1 **do** write(i, ' ');
end.
	#include<iostream.h>
	void main()
	{int i, n;
	cout<<" n = ";cin>>n;
	for(i = 1; i<=n ; i++) cout<<i<<" ";
	cout<<endl;
	for(i = n; i>=1; i- -) cout<<i<<" ";
	}

TEME

1. Se citesc n numere întregi. Să se determine și să se afișeze:

a) câte dintre ele sunt pare; b) suma elementelor pozitive; c) valoarea maximă.

Datele de intrare se citesc de la tastatură, iar datele de ieșire se vor afișa pe ecran, pe linii distinte.

2. Fie un șir cu p numere naturale. Să se afișeze suma divizorilor fiecărui număr. Datele de intrare se citesc din fișierul date.in astfel: de pe prima linie se citește numărul p, iar de pe următoarea linie se citesc p numere separate prin câte un spațiu. Datele de ieșire se vor afișa, pe linii distincte, în fișierul date.out.

3. Să se afișeze, pe ecran, toate numerele naturale de patru cifre cu proprietatea că suma inverselor cifrelor lor este un număr subunitar, iar cifrele sunt în ordine descrescătoare.

Exemplu: pentru numărul 7543 avem 1/7+1/5+1/4+1/2= 0.92619047 (fracție subunitară).

4. De la tastatură se citesc două numere naturale nenule m și z. Să se afișeze, pe ecran, primele z cifre din simetricul numărului m și suma acestora.

Exemplu: m = 13745, z = 3; se afișează 547 și 16.

5. Fie n un număr natural nenul. Calculați valoarea expresiei:

$$E = \left(1-\frac{1}{2^3}\right)\left(1-\frac{1}{3^3}\right)\cdots\left(1-\frac{1}{n^3}\right)$$

6. Să se scrie un program pentru afișarea triunghiului de numere alăturat, unde n este număr natural (n >= 5), citit de la intrarea standard (tastatură).

```
1
1 2
1 2 3
1 2 3 4
.....
1 2 3 4 5...n
```

3.3.2. Structuri repetitive cu condiție

În cazul în care o secvență de operații se repetă *cât timp/până când* este îndeplinită o condiție, atunci se folosesc structuri repetitive, cu test inițial sau cu test final.

♦ Structuri repetitive cu test inițial

Structura repetitivă cu test inițial este codificată prin instrucțiunea while (figura 9 și tabelul 14).

Evaluarea condiției precede secvența de operații; secvența se repetă cât timp condiția este îndeplinită.

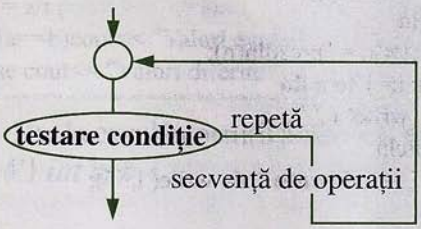


Figura 9. Structuri repetitive cu test inițial

Tabelul 14

Sintaxa instrucțiunii repetitive cu test inițial

LIMBAJUL PASCAL	LIMBAJUL C/C++
while(cond_logica) **do** S; Secvența (S) se execută numai în cazul în care condiția logică este adevărată (expresia generează valoarea logică **true**). Dacă secvența conține mai multe instrucțiuni, atunci acestea sunt cuprinse între **begin** și **end**.	**while**(cond_logica) S; Secvența (S) se execută numai în cazul în care condiția logică este adevărată (expresia generează o valoare nenulă). Dacă secvența conține mai multe instrucțiuni, atunci acestea sunt cuprinse între acolade {}.

Exemple:

1. Se afișează, pe ecran, suma cifrelor unui număr natural nenul x. Numărul x se citește de la tastatură.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var x , s : word;
begin
write(' x = ');readln(x);
s : = 0;
while(x<>0) **do**
begin
s : = s + x mod 10 ; x : = x div 10;
end;
writeln('Suma cifrelor = ',s);
end.
	#include<iostream.h>
	void main()
	{unsigned int x, s = 0;
	cout<<" x = ";cin>>x;
	while (x != 0)
	{ s = s + x%10; x = x/10;
	}
	cout<<" Suma cifrelor = "<<s<<endl;
	}

2. Se afișează, pe ecran, produsul primelor m numere naturale. Numărul natural nenul m se citește din fișierul date.in, iar produsul se va afișa în fișierul date.out. Se utilizează o structură repetitivă cu test inițial.

Observație: În rezolvarea abordată, se prezintă echivalența dintre instrucțiunea repetitivă cu contor și cea repetitivă cu test inițial.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var m, i, p : word;
f, g : text;
begin
assign(f,'date.in') ; reset(f);
assign(g,'date.out'); rewrite(g);
read(f, m);
p:=1; **i : = 1**;
while(**i < = m**) do
begin
p : = p * i;
i : = i + 1; { inc(i) }
end;
write(g, 'produsul primelor ',m);
writeln(g, ' numere naturale = ', p);
close(f); close(g);
end.
	#include<fstream.h>
	
	void main()
	{ ifstream f("date.in");
	ofstream g("date.out");
	unsigned m , i , p = 1;
	f>>m;
	i = 1;
	while (**i <= m**)
	{ p = p * i ;
	i = i + 1; // i ++
	}
	g<<" produsul primelor "<<m;
	g<<" numere naturale = " <<p;
	f.close(); g.close();
	}

♦ Structuri repetitive cu test final

Structura repetitivă cu test final este codificată, în limbajul Pascal, prin instrucțiunea repeat_until, iar în limbajul C/C++, prin instrucțiunea do_while (figura 10 și tabelul 15).

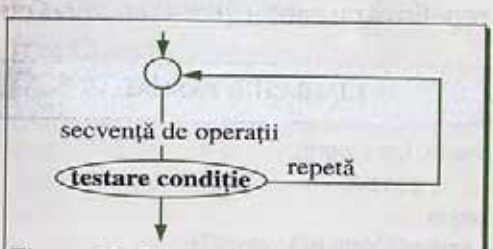


Figura 10. Structuri repetitive cu test final

Evaluarea condiției se face după execuția secvenței de operații și determină repetarea secvenței sau ieșirea din structură.

Tabelul 15

Sintaxa instrucțiunii repetitive cu test final

LIMBAJUL PASCAL	LIMBAJUL C/C++
repeat S; **until** (cond_logica); Secvența (S) se execută **până când** condiția logică este adevărată (expresia generează valoarea logică **true**). Secvența se execută cel puțin o dată, chiar dacă condiția logică este îndeplinită (adevărată).	**do** { S; }**while**(cond_logica); Secvența (S) se execută **cât timp** condiția logică este adevarată (expresia generează o valoare nenulă). Secvența se execută cel puțin o dată, chiar dacă condiția logică nu este îndeplinită.

Exemple:

1. Se afișează, pe ecran, cifra minimă a unui număr natural nenul a. Numărul a se citește de la tastatură.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a , min , c : word;
begin
write(' a = ');readln(a);
min : = 9;
repeat
c : = a mod 10;
if c< min then min := c;
a : = a div 10;
until (a = 0);
writeln('Cifra minima = ', min);
end.
	#include<iostream.h>
	void main()
	{unsigned int a , min = 9, c;
	cout<<" a = "; cin>>a;
	do
	{ c = a % 10;
	if (c < min) min = c;
	a = a/10;
	}**while**(a != 0) ;
	cout<<" Cifra minima = "<<min<<endl;
	}

2. Se calculează produsul primelor m numere naturale. Numărul natural nenul m se citește din fișierul date.in, iar produsul se va afișa în fișierul date.out. Se folosește o structură repetitivă cu test final.

Observație: În rezolvarea abordată se prezintă echivalența dintre instrucțiunea repetitivă cu contor și cea repetitivă cu test final.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var m, i, p : word;
f, g : text;
begin
assign(f,'date.in') ; reset(f);
assign(g,'date.out'); rewrite(g);
read(f, m);
p:=1; **i : = 1**;
repeat
p : = p * i;
i : = i + 1; { inc(i) }
until (i>m)
write(g, 'produsul primelor ', m);
write(g, 'numere naturale = ', p);
close(f); close(g);
end.
	#include<fstream.h>
	
	void main()
	{ ifstream f("date.in");
	ofstream g("date.out");
	unsigned m , i , p = 1;
	f>>m;
	i = 1;
	do
	{ p = p * i ;
	i = i + 1; // i ++
	}**while (i <= m)**;
	g<<" produsul primelor "<<m;
	g<<" numere naturale = "<<p;
	f.close(); g.close();
	}

TEME

Recomandare: Toate aplicațiile propuse pentru rezolvare la instrucțiuni repetitive cu contor (vezi pagina 25) se pot rezolva utilizând structuri repetitive cu condiție.

1. Se citește, de la intrarea standard, un șir de numere reale până când se introduce valoarea zero. Să se determine media aritmetică a elementelor negative și media aritmetică a celor pozitive.

2. Din fișierul palindrom.in se citește un număr natural nenul n, iar de pe următoarea linie se citesc n numere naturale nenule separate prin câte un spațiu. În fișierul palindrom.out, să se afișeze pe prima linie șirul citit, pe următoarele linii numerele care sunt palindromuri, iar pe ultima linie câte palindromuri s-au găsit sau un mesaj corespunzător. (Un *palindrom* este un număr simetric; de exemplu, 15451 este palindrom, 15452 nu este palindrom.)

3. De la tastatură se citește un număr natural nenul t. Prin descompunerea în factori primi, să se determine suma ordinelor de multiplicitate ale fiecărui factor. Pe baza rezultatului obținut, se poate preciza dacă numărul este prim?

4. De pe prima linie a fișierului fractii.in se citește un număr natural nenul n care reprezintă un număr de fracții raționale. De pe următoarele n linii se citesc perechi de numere naturale nenule, separate prin câte un spațiu.
În fișierul fractii.out, să se afișeze cele n fracții simplificate.

Exemplu:

fractii.in	fractii.out
4 12 27 17 23 54 28 125 275	4/9 17/23 27/14 5/11

5. Se citește un șir de numere întregi până la citirea valorii -5. Să se determine cel mai mare divizor comun și cel mai mic multiplu comun al numerelor citite. Datele de intrare vor fi citite de la tastatură, iar datele de ieșire se vor afișa pe linii distincte în fișierul cmmdcmc.out.

6. De la intrarea standard se citesc u numere întregi. Să se afișeze, pe ecran, pe linii distincte, suma cifrelor fiecărui număr.

7. Fie două numere naturale a și b (a < b). Să se afișeze toate numerele *rotunde* din intervalul [a, b) și numărul lor. (Un număr este *rotund* dacă în reprezentarea în baza 2 are numărul de cifre de 1 egal cu numărul de cifre de 0.)

Exemplu: $12_{(10)} \rightarrow 1100_{(2)}$ este rotund.

4. ALGORITMI ELEMENTARI — IMPLEMENTARE

4.1. Rezolvarea ecuației de gradul II

Ecuația de gradul II are forma generală $ax^2 + bx + c = 0$, $a, b, c \in \mathbb{R}$. Din fișierul date.in se citesc trei numere reale (separate prin câte un spațiu), care reprezintă coeficienții ecuației. În fișierul date.out se va afișa tipul rădăcinilor și valoarea lor (tabelul 16).

Tabelul 16

Rezolvarea ecuației de gradul II

LIMBAJUL PASCAL

```
var a,b,c,x1,x2,delta : real;
    f,g:text;
begin
 assign(f, 'date.in'); reset(f);
 assign(g, 'date.out'); rewrite(g);
 read(f,a,b,c);
  if(a=0) then writeln(g, 'ec. de gradul I ')
  else
   begin
    delta:=b*b-4*a*c;
    if(delta>0)then
      begin
        x1:=(-b-sqrt(delta))/(2*a);
        x2:=(-b+sqrt(delta))/(2*a);
        write(g,'ec. are doua radacini reale ');
        write(g,'x1=',x1,' x2=',x2 );
      end
     else
      if(delta=0)then
        begin
         write(g,' ec. are doua radacini reale');
          write(g,' x1=x2 = ',-b/(2*a));
        end
       else
              write(g,'ec. are doua radacini
complexe');
   end;
 close(f);close(g);
end.
```

LIMBAJUL C/C++

```
#include<fstream.h>
#include<math.h>
void main()
{ float a,b,c,x1,x2, delta ;
  ifstream f("date.in"); ofstream g("date.out");
  f>>a>>b>>c;
  if(a==0) g<<"ec. de gradul I ";
  else
  {
     delta= b*b-4*a*c;
    if(delta>0)
       {
         x1=(-b-sqrt(delta))/(2*a);
         x2=(-b+sqrt(delta))/(2*a);
         g<<"ec. are doua radacini reale ";
         g<<"x1 = "<<x1<<" x2 = "<<x2;
       }
      else
      if(delta==0)
        {
          g<<"ec. are doua radacini reale ";
          g<<" x1 = x2 = "<<-b/(2*a);
        }
       else
         g<<"ec. are doua radacini complexe ";
  }
 f.close();g.close();
}
```

4.2. Cel mai mare divizor comun a două numere naturale

Din fișierul euclid.in se citesc două numere întregi a și b, separate prin spațiu. În fișierul euclid.out se va afișa cel mai mare divizor comun (tabelul 17).

Tabelul 17

Cel mai mare divizor comun a două numere naturale

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a, b, r : integer;
f,g:text;
begin
assign(f, 'euclid.in');reset(f);
assign(g, 'euclid.out');rewrite(g);
readln(f, a, b);
while(a mod b<>0) do
begin
r:= a mod b; a : = b; b : = r;
end;
writeln(g, b);
close(f); close(g) ;
end.
	#include<fstream.h>
	void main()
	{ int a, b, r;
	ifstream f("euclid.in");
	ofstream g("euclid.out");
	f>>a>>b;
	while(a%b)
	{ r = a%b; a = b ; b = r; }
	g<<b;
	f.close();
	g.close();
	}

4.3. Numere prime

Se citesc n numere naturale. Se afișează, pe linii distincte, numerele care sunt prime (tabelul 18).

Tabelul 18

Numere prime

LIMBAJUL PASCAL	LIMBAJUL C/C++

var n, x, i , j : word;
prim : boolean;
begin
write(' n = '); readln(n);
for i:=1 to n do
begin
write(' numarul = '); readln(x);
if(x=1) then prim:=false
else
begin
prim:=true; j:=2;
while(prim and (j<=trunc(sqrt(x)))) do
if(x mod j = 0) then prim := false
else inc(j);
end;
if(prim) then writeln(x)
end; end.
	#include<iostream.h>
	#include<math.h>
	void main()
	{ unsigned int n, i , j, x, prim;
	cout<<" n = ";cin>>n;
	for(i=1; i<=n; i++)
	{ cout<<" numarul = "); cin>>x;
	if(x= =1) prim=0;
	else
	{ prim=1; j=2;
	while (prim && j<=sqrt(x))
	if(x%j = = 0) prim=0;
	else j++;
	}
	if(prim) cout<<x<<endl;
	}
	}

4.4. Descompunerea în factori ireductibili a unui număr natural

Se consideră un număr natural nenul a. Se descompune acest număr în factori ireductibili (tabelul 19).

Tabelul 19

Descompunerea în factori ireductibili a unui număr natural

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a, d , m: word;
begin
write(' a= ');readln(a);
d:=2;
while(a>1) do
begin
m:=0;
while(a mod d = 0) do
begin
inc(m); a : = a div d;
end;
if(m>0)then write(d,'^',m,' ');
inc(d);
end
end.
	#include<iostream.h>
	void main()
	{ unsigned int a, d, m;
	cout<<" a = ";cin>>a;
	d=2;
	while(a>1)
	{ m=0;
	while(a%m==0)
	{ m++; a = a/d; }
	if(m) cout<<d<<"^"<<m<<" ";
	d++;
	}
	}

4.5. Numere aleatoare

Se generează m numere naturale mai mici decât 121. Se determină care este cel mai mare număr generat și poziția sa în șir (tabelul 20).

Tabelul 20

Numere aleatoare

LIMBAJUL PASCAL	LIMBAJUL C/C++

var m, i, z, max, poz : word;
begin
write(' m = ') ; readln(m);
randomize;
max : = **random**(122) ; poz :=1;
for i := 2 to m do
begin
z : = **random**(122) ;
if(z>max) then
begin
max : = z ; poz := i ;
end;
end;
writeln(' maxim = ',max,' pe pozitia = ',poz); end.
	#include<iostream.h>
	#include<stdlib.h>
	void main()
	{int m, i, z, max, poz;
	cout<<" m = "; cin>>m;
	randomize();
	max=**rand**()%122; poz=1;
	for(i=2;i<=m;i++)
	{ z = **rand**() % 122;
	if(z>max)
	{ max=z; poz = i ; }
	}
	cout<<"maxim = "<<max;
	cout<<" pe pozitia = "<<poz; }

TABLOURI UNIDIMENSIONALE

1. PRELUCRAREA DATELOR CU ACEEAȘI SEMNIFICAȚIE

Aplicația 1: *salvamont1*

Un echipaj Salvamont decide să monteze corzi de siguranță pe traseele turistice cu risc mare de accidentare. Deoarece corzile de siguranță sunt foarte scumpe, acestea vor fi montate doar între punctele a căror diferență de nivel este mai mare de 5 m.

Cunoscând numărul de puncte din traseu și cotele de nivel ale acestora, echipa dorește să determine numărul de corzi ce vor fi instalate și cât de mare este cea mai lungă dintre corzi.

Pentru un traseu cu 10 puncte în observație, având cotele montane 3, 7, 15, 20, 28, 32, 46, 50, 53 și 59, sunt necesare 4 corzi, cea mai lungă fiind de 14 m.

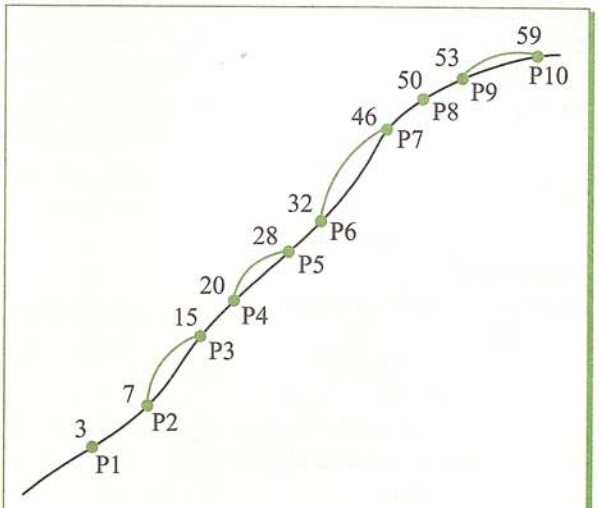


Figura 11. Trasee turistice — coarda cea mai lungă

1. *Analiza problemei*

- *Date de intrare*
  - — numărul de puncte (N)
  - — N valori reprezentând cotele (H)
- *Date de ieșire*
  - — numărul de corzi (C)
  - — coarda cea mai lungă (L)
- *Condiții și relații importante*

Se calculează diferența D dintre două puncte consecutive H 2 și H 1: D = H 2 – H 1. Interesează cazul în care D > 5.

2. *Raționamentul problemei*

Pas 1. Se introduce numărul de puncte, N.

Pas 2. Se citesc primele două valori, H 1 și H 2.

Pas 3. Se calculează D și se verifică condiția.

Pas 4. Dacă este îndeplinită condiția, crește numărul de corzi și lungimea corzii instalate se memorează în L.

Pas 5. Se păstrează valoarea din H 2 în H 1.

Pas 6. Se citește o nouă valoare în H 2.

Pas 7. Se calculează D și se verifică condiția.

Pas 8. Dacă este îndeplinită condiția, crește numărul de corzi și se compară ultima coardă cu cea memorată. În memorie, se păstrează valoarea cea mai mare.

Pas 9. Se repetă operațiile P5, P6, P7, P8 pentru toate valorile.

Pas 10. Se afișează numărul de corzi, C, și coarda cea mai lungă, L.

3. *Reprezentarea algoritmului*

```
început salvamont1
    citește N
    citește H1, H2
    C ← 0
    L ← 0
    D ← H2 – H1
    dacă D > 5 atunci
                bloc
                    C ← C + 1
                    L ← D
                sfârșit bloc
    sfârșit dacă
    pentru i ← 1, N–2 execută
        bloc
            H1 ← H2
            citește H2
            D ← H2 – H1
            dacă D > 5 atunci
                bloc
                    C ← C + 1
                    dacă D > L atunci L ← D
                    sfârșit dacă
                sfârșit bloc
            sfârșit dacă
        sfârșit bloc
    sfârșit pentru
    scrie C, L
sfârșit salvamont1
```

4. *Verificarea algoritmului*

Pentru verificarea algoritmului, completați tabelul de variație:

N	H1	H2	D	D > 5	C	L	I

TEME

1. Modificați algoritmul prezentat astfel încât să nu mai fie necesare operațiile de la Pas 3 și Pas 4.

2. Câte zone de memorie folosește algoritmul prezentat?

3. Codificați algoritmul prezentat, în limbajul de programare studiat.

4. Stabiliți care dintre următoarele cerințe pot fi rezolvate folosind aceleași date de intrare și aceleași zone de memorie ca în algoritmul prezentat:

a) Să se determine dimensiunea celei mai scurte corzi.

b) Să se afișeze corzile care se vor monta înainte și după cea mai lungă coardă.

c) Să se determine câte corzi de aceeași lungime se vor folosi.

d) Să se determine punctele care formează cea mai periculoasă zonă din traseu (cea mai lungă secvență de corzi consecutive). În exemplul din figura12, există două astfel de zone: prima zonă este formată din punctele P1, P2, P3 și P4, iar a doua din punctele P5, P6, P7 și P8.

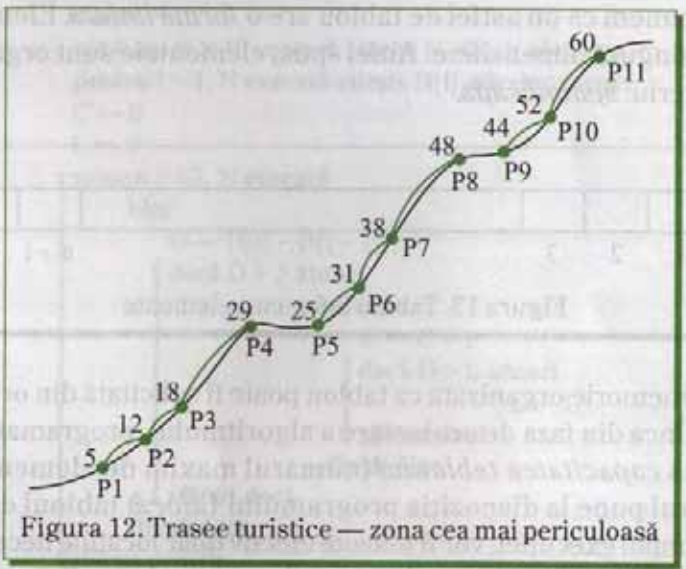


Figura 12. Trasee turistice — zona cea mai periculoasă

Rețineți!

În rezolvarea aplicației *salvamont1* s-a folosit raționamentul pentru determinarea valorii maxime dintr-un șir de numere.

2. ORGANIZAREA DATELOR ÎN TABLOURI UNIDIMENSIONALE

În aplicația *salvamont1* sunt prelucrate mai multe valori cu aceeași semnificație: cotele montane ale punctelor din traseu. Rezolvarea problemei a răspuns strict la cerința determinării numărului de corzi și a corzii de lungime maximă. De aceea, a fost suficientă păstrarea în memorie, la un moment dat, doar a ultimelor două valori consecutive H 1 și H 2. Dacă sunt necesare și alte prelucrări, de genul: *Câte corzi de aceeași lungime H se vor folosi* sau *câte corzi egale între ele se vor instala*? Rezolvarea unor astfel de cerințe necesită păstrarea tuturor valorilor în n zone de memorie ale calculatorului: H 1, H 2, H 3, ... , Hn. Doar pentru introducerea datelor ar fi necesare n comenzi distincte de citire, câte una pentru fiecare zonă de memorie: *citește H 1, citește H 2* etc. Valoarea lui n variază de la un traseu la altul, de la o execuție la alta a programului. Ar trebui să adăugăm sau să ștergem comenzi înaintea fiecărei execuții.

Situația pe care o analizăm se rezolvă organizând datele cu aceeași semnificație într-un ansamblu numit *tablou*. Ansamblul primește un nume și fiecare element are o adresă compusă din numele tabloului și poziția elementului în tablou — ca o adresă poștală cu stradă și număr.

Pentru cele n cote montane, tabloul s-ar putea numi H, iar adresele elementelor ar fi H[1], H[2], ..., H[n].

Intuitiv, spunem că un astfel de tablou are o *formă liniară*. Elementele sunt dispuse pe o singură dimensiune. Altfel spus, elementele sunt organizate după un singur criteriu: *semnificația*.

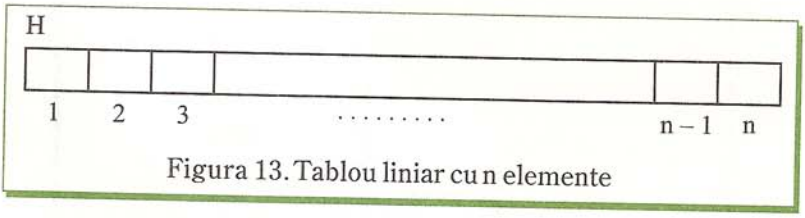


Figura 13. Tablou liniar cu n elemente

O zonă de memorie organizată ca tablou poate fi solicitată din orice limbaj de programare. Încă din faza de proiectare a algoritmului, programatorul trebuie să stabilească *capacitatea tabloului* (numărul maxim de elemente).

Calculatorul pune la dispoziția programului (*alocă*) tabloul de capacitate maximă. În timpul execuției, vor fi folosite efectiv doar locațiile necesare potrivit valorii lui n. Identificarea elementelor prin adresă permite realizarea de secvențe scurte de operații care se repetă, pentru fiecare element al tabloului, doar prin schimbarea adresei.

Spre exemplu, citirea și memorarea cotelor montane în tabloul liniar H se face prin secvența:

```
început citire_date1
    alocă H[10]
    citește n
    cât timp n > 10 execută citește n sfârșit cât timp
    pentru i = 1, n execută citește H[i] sfârșit pentru
sfârșit citire_date1
```

Secvența solicită spațiu pentru cel mult 10 elemente, de aceea structura repetitivă cât timp controlează valoarea lui n. Adresa fiecărui element se generează în structura repetitivă cu contor și este dată chiar de valoarea contorului.

Aplicația 2: *salvamont2*

Reluăm problema traseelor turistice în scopul rezolvării prin memorarea cotelor montane într-un tabloul liniar **H**. *Raționamentul de rezolvare rămâne același. Elementele consecutive vor fi de forma* ***H[i]*** *și* ***H[i – 1]***.

Reprezentarea algoritmului

```
început salvamont2
    alocă H[10]
    citește N
    cât timp N > 10 execută citește N sfârșit cât timp
    pentru i = 1, N execută citește H[i] sfârșit pentru
    C ← 0
    L ← 0
    pentru i = 2, N execută
        bloc
            D ← H[i] – H[i – 1]
            dacă D > 5 atunci
                        bloc
                            C ← C + 1
                            dacă D > L atunci
                                        L ← D
                            sfârșit dacă
                        sfârșit bloc
            sfârșit dacă
        sfârșit bloc
    sfârșit pentru
  scrie C, L
sfârșit salvamont2
```

Rețineți! Cum organizăm datele în tablouri

1. Dacă într-o problemă este necesară memorarea mai multor valori cu aceeași semnificație, aceste valori vor fi grupate într-un ansamblu de tip tablou.

2. Elementele unui tablou se identifică printr-o adresă formată din numele tabloului și un indice.

3. Operațiile care se repetă pentru fiecare element din tablou pot fi grupate în structuri repetitive cu contor. Contorul genereză chiar indicele de adresă.

TEME

1. Propuneți o soluție pentru organizarea și memorarea datelor specifice următoarelor probleme:

a) Se cunoaște numărul de cărți așezate pe cele n rafturi dintr-o bibliotecă. Să se determine câte rafturi au același număr de cărți.

b) Un copil își notează câți bani primește și câți cheltuiește în fiecare zi. După n zile, copilul este curios să afle în ce zile a reușit să facă economii.

c) Se cunoaște înălțimea și greutatea fiecărui elev din clasă. Să se determine înălțimea medie a băieților și greutatea medie a fetelor.

2. Formulați exemple de situații (probleme) care să necesite organizarea și memorarea datelor în tablouri unidimensionale.

STUDIU DE CAZ: *HARTA ELECTRONICĂ*

O hartă electronică afișează pe ecranul calculatorului imaginea unui teritoriu (oraș, județ, țară, continent). Pe hartă există puncte evidențiate prin simboluri speciale. Prin selectarea unui astfel de punct se afișează, pe ecran, informații specifice.

Exemplu:

- Pe o hartă administrativ-teritorială, pentru un *punct-oraș* se afișează numărul de locuitori.
- Pe o hartă geografică, pentru un *punct-vârf montan* se afișează înălțimea.

Cerințe:

1. Propuneți și alte puncte — cu informațiile specifice — care ar putea fi selectate dintr-o hartă electronică.

2. Identificați datele cu aceeași semnificație necesare pentru realizarea unei hărți electronice.

3. Propuneți o soluție pentru organizarea și memorarea datelor specifice unui teritoriu de mici dimensiuni (cartier, zona școlii, oraș).

4. Propuneți o soluție pentru organizarea și memorarea datelor specifice unui teritoriu larg (capitală, județ, țară etc.).

Sugestii de lucru: Clasa poate fi împărțită în grupe; fiecare grupă își alege o hartă tematică, pentru un teritoriu specificat. Prezentarea soluțiilor găsite de fiecare grupă va fi urmată de discuții și aprecieri.

3. IMPLEMENTAREA TABLOURILOR UNIDIMENSIONALE

Pentru memorarea și prelucrarea datelor organizate ca tablouri unidimensionale, înainte de scrierea unui program, trebuie să cunoaștem:

— tipul elementelor din tablou (datele cu aceeași semnificație);

— numărul maxim de elemente din tablou: *nrmax* (capacitatea tabloului).

Aceste elemente rezultă din analiza problemei și sunt folosite de compilator pentru determinarea zonei de memorie alocată tabloului.

Tabloul ocupă, în memoria calculatorului, o „suprafață" *(array)* compusă din locații învecinate *(adiacente)*. În fiecare locație, este memorată valoarea unui element. Adresa locațiilor începe de la o *valoare de referință* specifică limbajului de programare și se construiește prin valori succesive pentru fiecare element din tablou (figura 14). Numărul elementelor (n) memorate la un moment dat în tablou nu poate depăși valoarea maximă *nrmax*. În consecință, tipul variabilei folosite ca indice de adresă trebuie să accepte valori în domeniul [*valoare de referință, nrmax*]. Adresarea elementelor se face, cel mai frecvent, prin structura repetitivă de tip for, indicele de adresă fiind chiar contorul structurii, în general având valori naturale sau de tip caracter (tabelul 21).

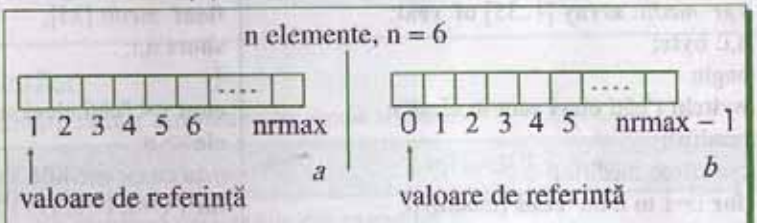


Figura 14. Implementarea tablourilor unidimensionale: *a* — limbajul Pascal; *b* — limbajul C/C++

Tabelul 21

Elemente de sintaxă specifice tablourilor unidimensionale

LIMBAJUL PASCAL	LIMBAJUL C/C++
Declararea tabloului	
var *numetablou* : **array** [1..*nrmax*] **of** *tipelement*	*tipelement numetablou* [*nrmax*]
Declararea indicelui de adresă	
var *numeindice:tipindice*	*tipindice numeindice*
Adresarea unui element din tablou	
numetablou [*numeindice*]	*numetablou* [*numeindice*]

Exemplu:

Pentru tabloul *cote* cu 100 de elemente de tip întreg în care sunt inițializate cu 0 primele 10 elemente, scriem următoarele instrucțiuni:

LIMBAJUL PASCAL	LIMBAJUL C/C++

var *cote*: **array** [1..100] **of integer**;
i: **byte**;
begin
for i:=1 to 10 **do** *cote*[i] =0;
end
	int *cote*[100];
	short i;
	{
	for (i =0; i< 10; i++)
	cote[i]=0;
	}

Datele organizate în tablouri sunt prelucrate element cu element.

Iată câteva dintre cele mai frecvente prelucrări:

- introducerea valorilor direct de la tastatură sau dintr-un fișier;
- afișarea valorilor pe ecran sau într-un fișier;
- verificarea unor proprietăți;
- determinarea unor valori medii;
- compararea valorilor (de exemplu, pentru aflarea elementului maxim sau minim).

Exemple:

1. Introducerea și afișarea valorilor

Profesorul diriginte cunoaște media anuală a fiecărui elev și dorește un program care să realizeze introducerea mediilor de la tastatură și afișarea lor pe ecran.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var *medii*: **array** [1..35] **of real**;
n,i: **byte**;
begin
writeln ('câți elevi sunt în clasă');
read(n);
{se citesc mediile}
for i:=1 to n do **read** (medii[i]);
{se afișează mediile}
for i:=1 to n do **writeln** (medii[i]);
end
	float *medii* [35];
	short n,i;
	{
	cout << "câți elevi sunt în clasă";
	cin>> n;
	// se citesc mediile
	for (i =0; i< n; i++)
	cin>> medii[i];
	// se afișează mediile
	for (i =0; i< n; i++)
	cout << medii[i];
	}

2. Verificarea unor proprietăți

Profesorul diriginte dorește să afle numărul de elevi nepromovați (media < 5).

LIMBAJUL PASCAL	LIMBAJUL C/C++
Se păstrează declarațiile din exemplul 1.	

var *nr:* **integer;**
begin
{se numără elevii nepromovați}
nr: = 0;
for i: = 1 to n **do**
if medii[i]<5
then nr: = nr + 1;
{se afișează numărul de nepromovați}
 	
writeln (nr, 'elevi nepromovati');
end
	int nr = 0;
	{
	// se numără elevii nepromovați
	for (i = 0; i < n; i ++)
	if (medii[i] < 5)
	nr = nr + 1;
	
	// se afișează numărul de nepromovați
	cout << nr << "elevi nepromovati";
	}

3. Determinarea mediei

Profesorul diriginte doreşte să afle media generală a clasei.

LIMBAJUL PASCAL	LIMBAJUL C/C++
Se păstrează declaraţiile din exemplul 1.	

var *m: real*;
begin
m: = 0;
for i: = 1 to n **do**
m: = m+ medii[i];
 	
{se calculează media clasei, în cazul general se testează n <> 0}
m: = m/n;
{se afişează media clasei}
writeln ('media clasei este', m);
end
	float m=0;
	{
	for (i = 0; i < n; i ++)
	m = m + medii[i];
	
	// se calculează media clasei, în cazul general se testează n <> 0}
	
	m = m/n;
	cout << "media clasei este" << m;
	}

4. Compararea valorilor

Profesorul diriginte doreşte să afle cea mai mare medie a clasei.

LIMBAJUL PASCAL	LIMBAJUL C/C++
Se păstrează declaraţiile din exemplul 1.	

var *max: real*;
begin
max: = medii[1];
for i: = 2 to n **do**
if medii[i] > max **then**
max: = medii[i];
{se afişează cea mai mare medie}
writeln ('media este', max);
end
	float max;
	{
	max = medii[0];
	for (i = 1; i < n; i ++)
	if (medii[i] > max)
	max = medii[i] ;
	//se afişează cea mai mare medie
	cout << "media este" << max;
	}

TEME

1. Realizaţi un program care să rezolve toate cerinţele din exemplele 1, 2, 3 şi 4.

2. Un tren de marfă poate avea cel mult 20 de vagoane. Fiecare vagon poate transporta o încărcătură de maxim 5 000 kg. Realizaţi un program pentru rezolvarea următoarelor cerinţe:

a) introducerea şi afişarea încărcăturii fiecărui vagon;

b) afişarea vagoanelor încărcate la întreaga capacitate;

c) afişarea vagoanelor cu cea mai mică încărcătură.

Construiţi date de test pentru verificarea fiecărui program.

3. Scrieți un program pentru introducerea și afișarea caracter cu caracter a codului numeric personal.

4. Se cunoaște temperatura înregistrată zilnic la stația meteo locală, timp de n zile. Scrieți un program care să determine temperatura maximă, temperatura minimă și zilele în care a fost atinsă temperatura medie. Construiți date de test pentru verificarea programului.

Test

I.

1. Specificați ce valori afișează secvența de operații alăturată, pentru vectorul v, unde v = 0, 3, 2, 4, 1, 6, 8:

a) 1, 2, 3; b) 4, 6; c) 2, 4, 6, 8;
d) nici o valoare.

```
pentru i = 1, 7 execută
    dacă v[i] = i atunci scrie i
    sfârșit dacă
sfârșit pentru
```

2. Precizați care dintre următoarele secvențe de operații calculează suma numerelor pare din vectorul v:

a)
```
S ← 0
pentru i = 1, N execută
    S ← S + v[i]
    dacă S mod 2 = 0
        atunci S ← 0
    sfârșit dacă
sfârșit pentru
```

b)
```
S ← 0
repetă
    dacă v[i] mod 2 = 0
        atunci S ← S + v[i]
    sfârșit dacă
până când i > N
```

c)
```
S ← 0
pentru i = 1, N execută
    dacă v[i] mod 2 = 0
        atunci S ← S + v[i]
    sfârșit dacă
sfârșit pentru
```

3. Construiți un vector v cu 10 elemente pentru care secvența de operații alăturată afișează valoarea 3.

```
z ← 0
pentru i = 1, 7 execută
    dacă v[i] = 0
        atunci z ← z + 1
    sfârșit dacă
sfârșit pentru
scrie z
```

4. Ce valori afișează secvența de operații alăturată, pentru vectorul v, unde v = 2, 6, 4, 2, 5, 7, 1, 8, 2, 6:

a) 2, 6, 4, 2, 5; b) 3, 7, 5, 3, 6; c) 1, 2, 3, 4, 5?

```
pentru i = 1, 5 execută
    bloc
        v[i] ← v[i] + 1
        scrie v[i]
    sfârșit bloc
sfârșit pentru
```

II. Codificați și testați secvențele de la I în limbajul de programare studiat.

Probleme propuse

Pentru fiecare dintre problemele propuse, parcurgeți următoarea schemă de rezolvare: analiza problemei, organizarea datelor, raționamentul de rezolvare, date de test, scrierea și testarea programului.

1. Cunoscând numărul de locuitori din n orașe, se dorește determinarea numărului de orașe a căror populație depășește o valoare p specificată. La o singură execuție a programului, pot fi specificate mai multe valori p până la introducerea valorii 0, având semnificația *sfârșit prelucrare*.

Exemplu: Sugestii pentru 4 orașe.

Date de intrare	Date de ieșire
n = 4	
1500000 750000 2005000 2500000	
p = 2000000	2
p = 1350000	3
p = 3000000	0
p = 0	*sfârșit prelucrare*

2. Se cunosc anii în care au avut loc evenimente istorice importante la nivel mondial. Fie n numărul acestora. Să se determine numărul de evenimente care au avut loc într-o perioadă [p 1, p 2]. La o singură execuție a programului, pot fi specificate mai multe perioade până la introducerea unei perioade de forma [0, 0] cu semnificația *sfârșit prelucrare*.

Date de intrare	Date de ieșire
n = 10	
1859 1457 1877 1804 1648 1763 1920 1526 1848 1917	
p1 = 1450 p2 = 1550	1457 1526
p1 = 1350 p2 = 1300	*perioadă greșită*
p1 = 1800 p2 = 1900	1859 1877 1804 1848
p1 = 1650 p2 = 1700	*nici un eveniment*
p1 =0 p2 = 0	*sfârșit prelucrare*

3. Clienții unui magazin on-line au la dispoziție oferta cu prețurile în euro pentru cel mult 25 de componente electronice. Fiecare cumpărător introduce cantitatea din produsul ales, exprimată în bucăți. Realizați un program care să afișeze valoarea totală a încasărilor, de la toți cumpărătorii dintr-o sesiune on-line (o execuție a programului).

Exemplu: Sugestii pentru 3 clienți și 5 produse.

Date de intrare	Date de ieșire
numărul de clienți = 3 numărul de produse = 5 Prețurile produselor 150 0.85 3.70 83.5 250 Client — Ce produse — Câte bucăți 1 — 1 2 5 — 1 2 1 2 — 2 — 3 3 — 3 4 — 2 3	401.70 2.55 257.9 *Total încasări* *662.15*

Indicație: Sunt necesare două tablouri — unul pentru prețuri și unul pentru produse.

4. Un *cod* poate fi format din cel mult 20 de caractere (cifre și litere mari). Să se formeze un *cod nou* care să înceapă cu literele primului *cod*, urmate de cifrele acestuia.

Exemplu: Sugestii pentru un cod format din 7 caractere.

cod 17A3R9E *cod nou* ARE1739

5. După n aruncări cu două zaruri, se cunoaște numărul de puncte obținute la fiecare aruncare. Suma punctelor formează punctajul aruncării. Să se determine numărul de apariții ale fiecărui punctaj.

Exemplu: Sugestii pentru 6 aruncări.

Date de intrare		Date de ieșire	
Zarul 1	Zarul 2	Punctaj	Număr de apariții
6	4	10	2
5	3	8	2
5	5	7	1
4	3	4	1
2	6		
1	3		

6. Într-un grup de n copii, există o regulă de comunicare astfel încât un mesaj să ajungă sigur la toți membrii grupului. Fiecare copil transmite mesajul către un singur copil. Cunoscând perechile de copii care comunică direct, să se determine ordinea în care aceștia primesc mesajul. Pentru confirmare, mesajul ajunge întotdeauna la copilul care a transmis mesajul în grup.

Exemplu: Sugestii pentru un grup de 6 copii.

Date de intrare	Date de ieșire
n = 6 Copilul 1 comunică direct cu copilul 4. Copilul 2 comunică direct cu copilul 5. Copilul 4 comunică direct cu copilul 2. Copilul 5 comunică direct cu copilul 3. Copilul 6 comunică direct cu copilul 1. Copilul 3 comunică direct cu copilul 6. Mesajul e trimis de copilul 3.	Ordinea în care copiii primesc mesajul este: 3, 6, 1, 4, 2, 5, 3

Indicație: Se folosește un singur tablou; pentru exemplul dat, tabloul va păstra următoarele valori: 4, 5, 6, 2, 3, 1.

7. Un număr de n copii, numerotați de la 1 la n, sunt așezați în cerc pe n scaune, numerotate de la 1 la n. Fiecare copil se mută pe scaunul vecinului său din stânga. Schimbul de scaune se repetă până când fiecare ajunge pe scaunul pe care s-a aflat la început. Realizați un program care să afișeze aranjamentul copiilor după fiecare schimb de scaune.

Inițial, fiecare copil i stă pe scaunul i.

Exemplu: Sugestii pentru 5 copii.

Aranjamentul inițial:
1 2 3 4 5

Aranjamentul după primul schimb de scaune:
2 3 4 5 1

Aranjamentul după al doilea schimb de scaune:
3 4 5 1 2

8. Cititorii unei biblioteci electronice primesc, la înscriere, un *user-cod* format din cel mult 25 de cifre. Pentru fiecare categorie de cititori (elevi, studenți, cercetători etc.) există un *cod-sursa*. Să se formeze *cheia-de-acces* știind că aceasta este formată din cifrele comune celor două coduri (*user-cod* și *cod-sursa*), în ordinea în care apar în codul sursă. În cazul în care cheia de acces nu se poate forma, se va afișa mesajul *cititor neautorizat*.

Exemplu: cod-sursă 1778248185
user-cod 2177566001 cheia de acces 1725
9600360963 *cititor neautorizat*

9. Ionuț și Andrei vor să însumeze două *numere foarte mari*. Atât de mari încât nu pot fi memorate nici de calculator, decât cifră cu cifră. Copiii s-au gândit să folosească două tablouri, câte unul pentru fiecare număr și să însumeze „ca la aritmetică" (figura 15).

Realizați programul care însumează cifră cu cifră două *numere foarte mari*.

Indicație: Se lucrează cu trei vectori și o locație pentru transport.

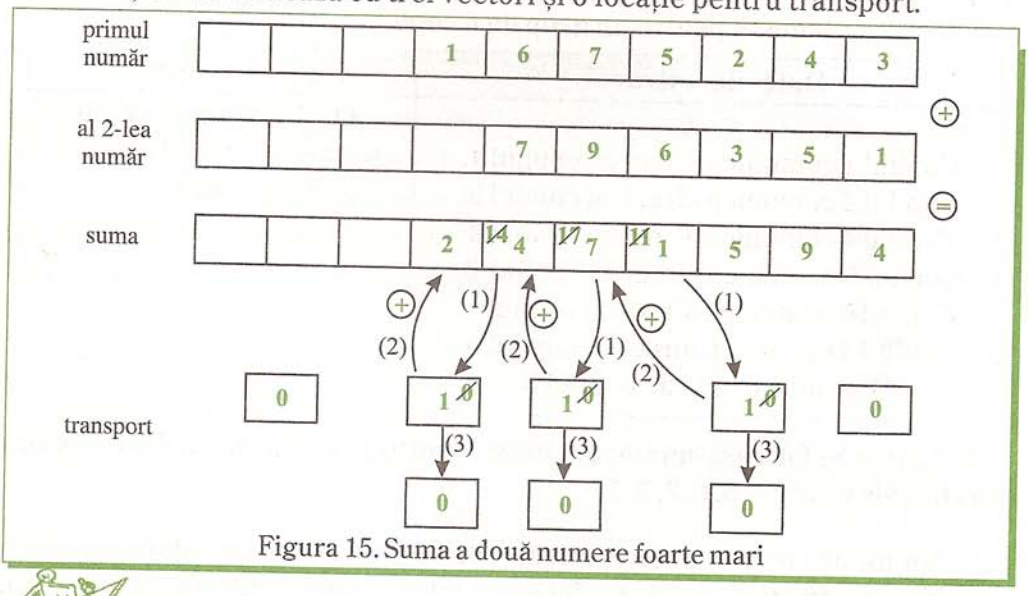


Figura 15. Suma a două numere foarte mari

MINIPROIECT: *Prelucrarea datelor experimentale*

Înregistrați rezultatele unor măsurători experimentale de laborator. Prelucrați aceste rezultate astfel încât să puneți în evidență ***valoarea medie, valoarea maximă, valoarea minimă*** *precum și* ***distribuția valorilor pe intervale semnificative*** *în funcție de specificul experimentului.*

Realizați o documentație completă a proiectului care să cuprindă:

A. Descrierea succintă a experimentului:
   1 — semnificația valorilor măsurate;
   2 — prelucrarea valorilor înregistrate;
   3 — interpretarea rezultatelor.

B. Utilizarea calculatorului în prelucrarea datelor experimentale:
   1 — organizarea datelor;
   2 — raționamentul de prelucrare;
   3 — programul.

C. Datele de test (măsurătorile experimentale).

D. Concluzii privind prelucrarea datelor experimentale cu calculatorul.

Sugestie de lucru. Proiectul poate fi realizat în echipă; prezentarea proiectului poate fi însoțită de o prezentare electronică (în PowerPoint) și de varianta prelucrării tabelare (în Excel) realizate la cursul de TIC (Tehnologia informațiilor și a comunicării).

ALGORITMI FUNDAMENTALI PENTRU PRELUCRAREA DATELOR

1. ALGORITMI DE SORTARE

1.1. Când și de ce ordonăm datele

Aplicația 1: *rucsac*

Trebuie să golim o încăpere plină cu tot felul de obiecte de diferite mărimi, greutăți și chiar utilități. Ni se cere să golim repede camera, din cât mai puține mutări. Avem la dispoziție doar un rucsac destul de încăpător, dar care nu „ține" mai mult de 30 kg. Obiectele au fost cântărite și pe fiecare este înscrisă greutatea (în kilograme): 7, 8, 10, 25, 5, 7, 8, 10, 9, 5, 6, 7, 25, 5.

1. *Raționamentul de rezolvare*

O modalitate ar fi să luăm obiectele la rând și să le încărcăm în rucsac. Fiecare rucsac, o mutare. În cazul nostru, avem 6 mutări (tabelul 22).

Tabelul 22

Rucsac	Mutări
7 + 8 + 10 = 25	1
25 + 5 = 30	2
7 + 8 + 10 = 25	3
9 + 5 + 6 + 7 = 27	4
25	5
5	6

E clar că nu e cea mai bună soluție. Ultima mutare e pentru un rucsac aproape gol. Obiectul de 5 kg ar fi încăput și-n primul rucsac și-n al treilea, dacă era „la rând".

Așadar, să „rânduim" altfel obiectele! Într-o ordine care să ne convină nouă. Cum nu contează decât greutatea, vom așeza obiectele, la rând, în această ordine: 5, 5, 5, 6, 7, 7, 7, 8, 8, 9, 10, 10, 25, 25. Să vedem câte mutări sunt necesare acum (tabelul 23)!

Tabelul 23

Rucsac	Mutări
5 + 5 + 5 + 6 + 7 = 28	1
7 + 7 + 8 + 8 = 30	2
9 + 10 + 10 = 29	3
25	4
25	5

Am redus o mutare și-am obținut numărul minim cu putință, în acest caz, de 5 mutări. Se poate verifica: suma greutăților este 130 kg, capacitatea rucsacului este de 30 kg; sunt necesare 5 rucsacuri.

Problema poate fi rezolvată cu calculatorul.

Pentru calculator, vom înlocui încăperea cu un tablou unidimensional. Pentru început, punem obiectele în ordinea din cameră. Apoi le așezăm după greutate și, în cele din urmă, le „adunăm" în rucsac — o zonă de memorie a cărei valoare nu va fi mai mare de 30.

2. *Reprezentarea algoritmului*

Deși nu e terminat, algoritmul arată astfel:

```
începput rucsac
  alocă G[100]
  repetă citește N până când N < 100
  pentru i=1,N execută citește G[i]
  sfârșit pentru
  ordonează G // prelucrare nerezolvată
  R ← 0 //golim rucsacul
  m ← 0
  i ← 1
  repetă
    dacă G[i]< =30 – R
          atunci
             bloc
                   R ← R+ G[i]
                   i ← i+1
             sfârșit bloc
          altfel
             bloc
                   m ← m+1
                   R ← 0
             sfârșit bloc
    sfârșit dacă
  până când i>N
  m ← m+1
  scrie m //numărul de rucsacuri
sfârșit rucsac
```

Algoritmul e pregătit să golească o cameră cu cel mult 100 de obiecte folosind un rucsac cu capacitate de 30 kg. *Generalizare:* se poate citi și capacitatea rucsacului, Q.

În rezolvarea acestei probleme, s-a folosit *metoda Greedy* (în limba engleză, *greedy* înseamnă *lacom*).

Rețineți! Ordonarea datelor

1. În foarte multe domenii de activitate, care necesită compararea unor valori, este necesară ordonarea convenabilă a acestora (stabilirea plătitorilor restanțieri, realizarea de topuri, prezentarea rezultatelor la un concurs/examen ș.a.).

2. În multe probleme, aranjarea (ordonarea) convenabilă a datelor conduce la găsirea unei soluții sigure și optime.

3. Metoda Greedy — metodă de programare bazată pe ordonarea convenabilă a datelor.

4. În funcție de specificul problemei, criteriul de ordonare a datelor poate fi crescător (de la valori mici la valori mari) sau descrescător (de la valori mari la valori mici).

TEME

1. Pentru fiecare dintre următoarele situații, justificați necesitatea ordonării datelor precizând criteriul de ordonare (crescător, descrescător).

a) Mai mulți elevi s-au documentat asupra evenimentelor istorice desfășurate în Europa în perioada Evului Mediu și le-au prezentat la ora de istorie în următoarea ordine:

493 838 771 584 1108 862 496 728

Deși a apreciat munca elevilor, profesorul nu a fost mulțumit de prezentare. De ce?

b) La sfârșitul fiecărei luni, directorul unei firme comerciale afișează o listă cu încasările realizate de fiecare dintre agenții săi de vânzări, în ordinea în care aceștia au fost angajați. Agenții de vânzări sunt nemulțumiți de modul de afișare întrucât ar dori să-și aprecieze cu ușurință nivelul încasărilor personale față de încasările celorlalți colegi. (Construiți și un exemplu numeric.)

c) Operatoarele de la serviciul *Informații* trebuie să găsească într-un timp scurt numărul de telefon al unui abonat cunoscând numele și adresa acestuia. Un informatician distrat a realizat o agendă telefonică în care numerele de telefon sunt așezate în ordine crescătoare. Operatoarele sunt nemulțumite. De ce?

d) La un club sportiv au început înscrierile pentru echipa de baschet. Deși sunt înscriși doar cei cu înălțimea peste 1,70 m, numărul candidaților este mult mai mare decât numărul locurilor. Cum procedează antrenorul pentru a-i alege cu ușurință pe cei mai dotați candidați?

2. Formulați exemple de probleme (situații) care necesită ordonarea datelor. Precizați criteriul de ordonare.

1.2. Sortarea prin metoda bulelor — Bubble Sort

În situația analizată în aplicația *rucsac*, pentru a umple eficient rucsacul, obiectele au fost reașezate în ordinea de la cel mai ușor la cel mai greu.

Rezolvarea problemei cu calculatorul nu este încă posibilă întrucât nu am prezentat algoritmul prin care se obține această reașezare a elementelor.

Inițial, vectorul greutăților, G, avea următoarele elemente:

G	7	8	10	25	5	7	8	10	9	5	6	7	25	5

Prin ordonare, elementele și-au schimbat poziția astfel încât la stânga oricărui element se află toate valorile mai mici decât el, iar la dreapta sa, toate valorile mai mari:

G	5	5	5	6	7	7	7	8	8	9	10	10	25	25

Trecerea vectorului de la configurația inițială (*vector neordonat*) la configurația finală (*vector ordonat*) se face prin compararea elementelor învecinate și schimbarea poziției lor astfel încât valorile mici să se deplaseze spre stânga, iar cele mari spre dreapta.

Pentru exemplificare, vom urmări ordonarea unui vector cu mai puține elemente (figura 16).

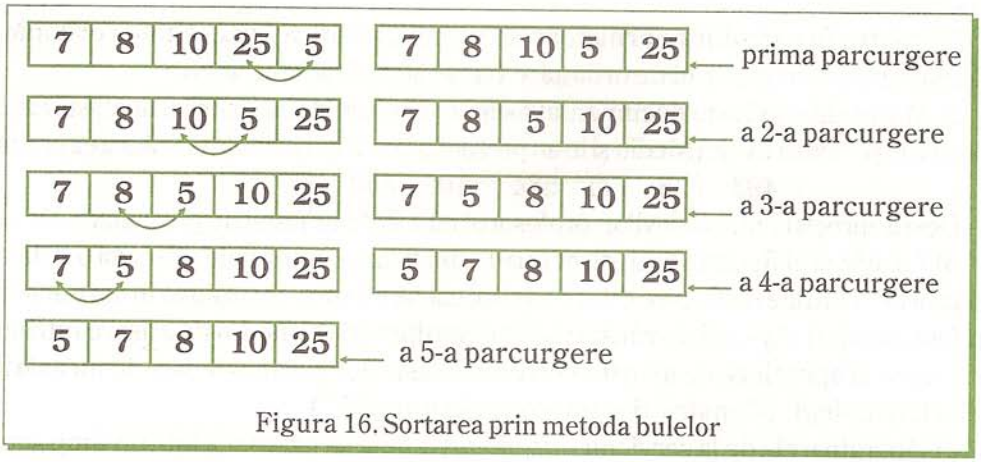


Figura 16. Sortarea prin metoda bulelor

Vectorul a fost parcurs de mai multe ori până când s-a ajuns la configurația finală.

La fiecare parcurgere, s-au comparat rând pe rând elementele învecinate și s-a făcut schimbarea de poziții (interschimbul) doar între valorile care nu respectau relația de ordine: 25 cu 5 la prima parcurgere, 10 cu 5 la a doua, 8 cu 5 la a treia și 7 cu 5 la a patra. Configurația finală se recunoaște prin faptul că nu s-a mai făcut nici un interschimb.

1. *Raționamentul metodei*

Se repetă parcurgerea vectorului cu interschimbul elementelor care nu respectă relația de ordine până când se ajunge la o parcurgere fără interschimb.

Pentru a reține dacă a avut loc sau nu cel puțin un interschimb într-o parcurgere, algoritmul va folosi o zonă de memorie S cu două valori posibile: 1, dacă a avut loc un interschimb, și 0, dacă nu a avut loc nici un interschimb.

2. *Reprezentarea algoritmului*

```
început ordonează-G
  repetă
      S ← 0
      pentru i=1,N-1 execută
          dacă G[i] > G[i+1] atunci
              bloc
                  A ← G[i+1]
                  G[i+1] ← G[i]
                  G[i] ← A
                  S ← 1
              sfârșit bloc
          sfârșit dacă
      sfârșit pentru
  până când S=0
sfârșit ordonează-G
```

3. *Tabelul de variație*

Verificați algoritmul de ordonare completând tabelul de variație:

N	S	I	A	G				
				1	2	3	4	5

Secvența de ordonare poate fi introdusă în algoritmul oricărei probleme.

Rețineți! Sortarea prin metoda bulelor

1. Ordonarea datelor prin interschimb și „mișcarea" lor spre stânga (valorile mici) sau dreapta (valorile mari), asemenea unor bule, poartă numele de Bubble Sort sau Metoda bulelor.

2. Metoda necesită un timp de lucru care depinde de numărul de treceri prin vector și de numărul de interschimburi la fiecare trecere.

3. Există numeroase metode de sortare a căror eficiență se măsoară după timpul de execuție și memoria folosită.

4. Orice programator trebuie să cunoască metodele de sortare și să aleagă folosirea unei metode, în funcție de criteriul de eficiență urmărit.

TEME

1. Rescrieți algoritmul *rucsac* înlocuind prelucrarea nerezolvată (ordonează_G) cu secvența corespunzătoare ordonării prin metoda bulelor.

2. Codificați algoritmul în limbajul de programare studiat și testați programul.

3. Realizați programul pentru ordonarea crescătoare a evenimentelor istorice (problema 1 punctul *a* de la pagina 49).

4. Realizați programul pentru ordonarea descrescătoare a înălțimilor candidaților la clubul sportiv (problema 1 punctul *d* de la pagina 49).

1.3. Sortarea prin selecție

Vă mai amintiți cum vă așeza doamna învățătoare după înălțime? Îl scotea din rând pe cel mai mic: „Dacă ești mic, stai în față!" Primul copil din rând fugea pe locul celui mai mic și acesta se așeza pe prima poziție. Apoi, venea în față cel mai mic dintre copiii rămași. Și tot așa, până la ultimii doi.

Să încercăm și noi cu numere (figura 17)!

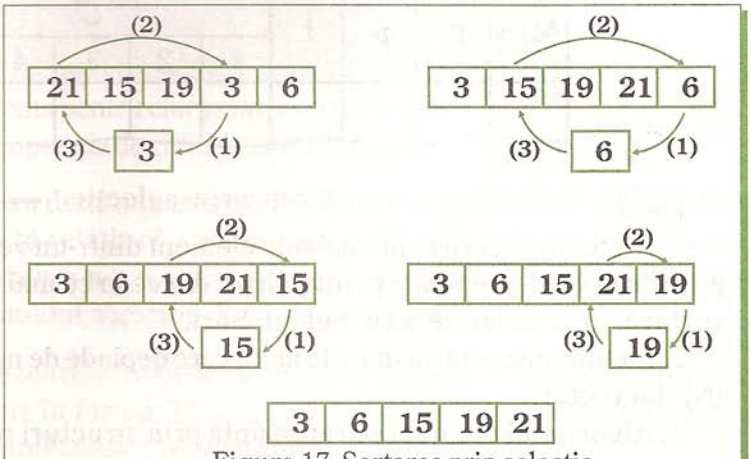


Figura 17. Sortarea prin selecție

1. *Raționamentul metodei*

Pentru calculator, șirul de copii va fi un vector C. Determinăm elementul minim (MIN) și realizăm interschimbul cu cel aflat pe prima poziție (p); pentru interschimb, reținem adresa elementului minim (m). Șirul în care se caută elementul minim devine tot mai scurt, până ajunge la doar două elemente. Pentru reprezentarea algoritmului considerăm că vectorul este citit și are N elemente.

2. *Reprezentarea algoritmului*

```
început ordonare2
  pentru p=1,N-1 execută
      bloc
        MIN ← C[p]
        m ← p
        pentru i=p+1,N execută
            dacă C[i] < MIN atunci
                              bloc
                                MIN ← C[i]
                                m ← i
                              sfârșit bloc
            sfârșit dacă
        sfârșit pentru
        C[m] ← C[p]
        C[p] ← MIN
      sfârșit bloc
  sfârșit pentru
sfârșit ordonare2
```

3. *Tabelul de variație*

Verificați algoritmul de ordonare completând tabelul de variație:

MIN	m	p	i	C				
				1	2	3	4	5

Rețineți! Sortarea prin selecție

1. Metoda alegerii celui mai mic element dintr-un vector și așezarea lui pe prima poziție, repetată pentru șiruri din ce în ce mai scurte, se numește sortare prin selecție sau Select Sort.

2. Metoda necesită un timp de lucru care depinde de numărul de elemente (N) din vector.

3. Algoritmul metodei se reprezintă prin structuri repetitive cu număr cunoscut de pași.

TEME

1. Rescrieți algoritmul *rucsac* înlocuind prelucrarea nerezolvată (ordonează_G) cu secvența corespunzătoare ordonării prin metoda sortare prin selecție.

2. Codificați algoritmul în limbajul de programare studiat și testați programul.

3. Realizați programul pentru ordonarea crescătoare a evenimentelor istorice (problema 1 punctul *a* de la pagina 49).

4. Realizați programul pentru ordonarea descrescătoare a înălțimilor candidaților la clubul sportiv (problema 1 *d* de la pagina 49).

2. ANALIZA EFICIENȚEI UNUI ALGORITM

Suntem în situația de a alege unul dintre mai mulți algoritmi care rezolvă aceeași problemă. Cum alegem cel mai bun algoritm? După ce criteriu?

Estimăm timpul de execuție. Programatorul își pune problema determinării timpului de execuție chiar în faza proiectării algoritmului. Este vorba de un timp teoretic întrucât, practic, timpul de execuție depinde de viteza de lucru a procesorului.

Rețineți!

Timpul teoretic în care un algoritm rezolvă problema depinde de numărul de operații executate.

Pentru a compara între ei doi algoritmi, se determină operațiile de bază, al căror număr de execuții variază de la un algoritm la altul. În general, acestea sunt operații de atribuire și de comparare. Contează și numărul de valori (n) cărora li se aplică aceste operații.

Exemple:

1. În secvența pentru citirea unui vector V cu n elemente, operația de citire se execută de n ori:

```
pentru i = 1, n execută
        citește V [i]
sfârșit pentru
```

2. În secvența pentru determinarea tripletelor a, b, c mai mici decât un n dat, $a < b < c$, care respectă relația $c^2 = a^2 + b^2$, operațiile de atribuire și verificare a relației se execută de $n * n * n = n^3$ ori (pentru o mai bună înțelegere, urmăriți reprezentarea primului algoritm de pe pagina următoare).

Din exemplele prezentate, rezultă că numărul de execuții ale operațiilor de bază se poate exprima în forma $O(n)$, $O(n^2)$, $O(n^3)$, formă numită și ordinul de mărime sau complexitatea algoritmului (tabelul 24).

Se consideră ca fiind cel mai bun algoritmul cu ordinul de mărime cel mai mic.

```
pentru a = 1, n-1 execută
    pentru b = a+1, n execută
        pentru c = b+1, n execută
            bloc
                a ← a*a
                b ← b*b
                c ← c*c
                dacă c= a+b atunci scrie a, b, c
                sfârșit dacă
            sfârșit bloc
        sfârșit pentru
    sfârșit pentru
sfârșit pentru
```

În faza de proiectare, programatorul determină complexitatea algoritmului și caută să reducă ordinul de mărime pentru a rezolva problema într-un timp cât mai scurt.

Tabelul 24

Complexitatea algoritmilor

Complexitate	Tipul de algoritm
$O(n)$	algoritm liniar
$O(n^2)$	algoritm pătratic
$O(n^3)$	algoritm cubic
$O(2^n)$	algoritm exponențial
$O(\log_2 n)$	algoritm logaritmic

În exemplul 2, complexitatea secvenței pentru determinarea tripletelor a, b, c poate fi redusă în mod evident de la $O(n^3)$ la $O(n^2)$:

```
a ← 1
cât timp a ≤ n-2 execută
    bloc
        b ← a+1
        cât timp b ≤ n-1 execută
            bloc
                c ← a*a + b*b
                c ← [√c]
                dacă (c < n) și (c*c=a*a + b*b)
                    atunci
                        scrie a, b, c
                sfârșit dacă
                b ← b+1
            sfârșit bloc
        sfârșit cât timp
        a ← a+1
    sfârșit bloc
sfârșit cât timp
```

Complexitatea unui algoritm poate varia în funcție de relațiile dintre valorile datelor de intrare. Spre exemplu, algoritmul de sortare Bubble Sort are complexitatea O(n) dacă secvența datelor de intrare este gata ordonată (vectorul este parcurs o singură dată; nu e necesar nici un interschimb) și complexitatea $O(n^2)$ în rest.

Cazul care conduce la complexitatea cea mai mică este numit situație favorabilă. Se caută, însă, și cazul care conduce la complexitatea cea mai mare (sau la numărul cel mai mare de operații executate pentru aceeași complexitate) — acesta corespunde situației defavorabile. Restul cazurilor sunt considerate situații intermediare sau medii.

Ca exercițiu practic, analizăm metodele de sortare prezentate punând în evidență numărul de parcurgeri, comparații și atribuiri în funcție de gradul de sortare a datelor de intrare. Precizăm că o astfel de analiză se face pentru un n foarte mare.

Metode de sortare / Situația analizată	Bubble Sort	Select Sort
Situația favorabilă 1, 4, 5, 7, 8, 9	— o singură parcurgere — doar comparații — nici un interschimb O(n)	— parcurgeri repetate — doar comparații — număr minim de atribuiri $O(n^2)$
Situația defavorabilă 9, 8, 7, 5, 4, 1	— parcurgeri repetate — număr maxim de interschimburi $O(n^2)$	— parcurgeri repetate — comparații și interschimburi — număr maxim de atribuiri $O(n^2)$
Situația medie 5, 7, 1, 8, 4, 9	— parcurgeri repetate — numărul de interschimburi depinde de gradul de sortare $O(n^2)$	— parcurgeri repetate — comparații și interschimburi — numărul de atribuiri depinde de gradul de sortare $O(n^2)$

Rețineți! Complexitatea algoritmilor

1. Complexitatea reprezintă un criteriu de apreciere a eficienței unui algoritm.
2. Timpul de execuție al unui algoritm este cu atât mai mic cu cât complexitatea acestuia este mai redusă.
3. Complexitatea algoritmului este aproximată prin numărul operațiilor de bază executate (atribuiri, comparații) și se notează cu O(f(n)), unde f(n) este relația care dă ordinul de mărime.

3. ALGORITMI DE CĂUTARE

Cu ajutorul algoritmilor de căutare pe care-i vom prezenta în continuare, vom realiza căutarea unei valori într-un tablou unidimensional și vom proceda la alegerea celui mai eficient algoritm.

3.1. Căutarea secvențială

Operația de căutare a unei valori (de fapt doar a primei sale apariții) într-un tablou presupune efectuarea operației de parcurgere a acestuia. Pe de altă parte, structura de date tablou permite accesul direct la fiecare componentă a sa într-un timp egal cu O(1). Ca urmare, complexitatea operației de căutare într-un tablou oarecare este determinată de complexitatea operației de parcurgere a acestuia și este, în cazul cel mai nefavorabil, egală cu O(n), unde n este numărul de elemente ale tabloului (figura 18).

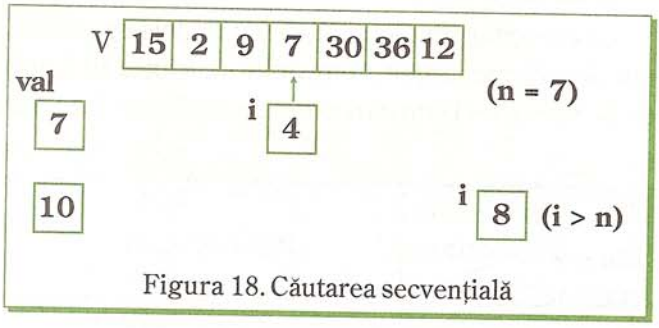


Figura 18. Căutarea secvențială

1. *Analiza problemei*

• *Date de intrare*
— numărul de elemente (n);
— valoarea căutată (val);
— tabloul în care se face căutarea (V).

• *Date de ieșire*
— indexul elementului din tabloul V a cărui valoare este egală cu valoarea căutată val; dacă valoarea nu a fost găsită, atunci se afișează un mesaj corespunzător.

2. *Reprezentarea algoritmului*

```
început căutare_secvențială
// se consideră introduse valorile în vectorul V cu n elemente
scrie "Ce valoare căutăm?"
citește val
i ← 1
cât timp (i <= n) și (V[i] <> val) execută i = i + 1
sfârșit cât timp
dacă i <= n atunci
        scrie "valoarea" val "a fost gasita pe pozitia" i
    altfel
        scrie "valoarea" val "nu exista in tablou"
sfârșit dacă
sfârșit căutare-secvențială
```

3.2. Căutarea într-un tablou ordonat

Dacă valorile tabloului sunt ordonate, atunci operația de căutare poate fi simplificată. În continuare, presupunem tabloul ordonat crescător și analizăm situațiile apărute în raport cu această presupunere. În cazul în care tabloul este ordonat descrescător, analiza se desfășoară analog. Convenim, de asemenea, să apreciem numai complexitatea căutării în cazul cel mai nefavorabil.

1. *Analiza problemei*

• *Date de intrare*
— numărul de elemente (n);
— valoarea căutată (val);
— tabloul în care se face căutarea (V).

• *Date de ieșire*
— indexul elementului din tabloul V a cărui valoare este egală cu valoarea căutată val; dacă valoarea nu a fost găsită, atunci se afișează un mesaj corespunzător.

2. *Raționamentul de rezolvare*

Datorită relației de ordine existente pe mulțimea elementelor tabloului, e suficient să vizităm elementele mai mici sau cel mult egale cu valoarea căutată. Atunci când valoarea elementului curent este egală cu valoarea val, căutarea se încheie cu succes. Dacă am parcurs tabloul și toate elementele sale au fost strict mai mici decât valoarea căutată sau dacă, deși nu am vizitat toate elementele, valoarea curentă este strict mai mare decât val, atunci căutarea se încheie fără succes (figura 19).

3. *Reprezentarea algoritmului*

```
început căutare_ordonată
// se consideră introduse valorile în vectorul V cu n elemente
scrie "Ce valoare căutăm?"
citește val
i ← 1
// inițiem parcurgerea tabloului

  cât timp (i <= n) și (V[i] < val) execută
        i ← i+1
  sfârșit cât timp
   // verificăm modul în care s-a terminat parcurgerea
  dacă i <= n
        atunci
          dacă V[i]= val
                atunci                    // am găsit valoarea pe poziția i
                  scrie "Valoarea" val "a fost găsită pe poziția" i
                altfel                    // componenta curentă depășește val
                  scrie " Valoarea" val "nu a fost găsită în tablou."
          sfârșit dacă
        altfel          // pentru i > n
          scrie " Valoarea" val "nu a fost găsită în tablou."
  sfârșit dacă
sfârșit căutare_ordonată
```

4. *Analiza eficienței*

Complexitatea algoritmului rămâne tot de ordinul O(n) deoarece, dacă valoarea căutată nu se află în tablou sau este egală cu cea mai mare valoare din tablou, atunci trebuie vizitate toate elementele.

Dacă valoarea căutată nu se află în tablou și este mai mică decât, de exemplu, a treia componentă, atunci căutarea se încheie rapid, după vizitarea primelor trei elemente ale tabloului.

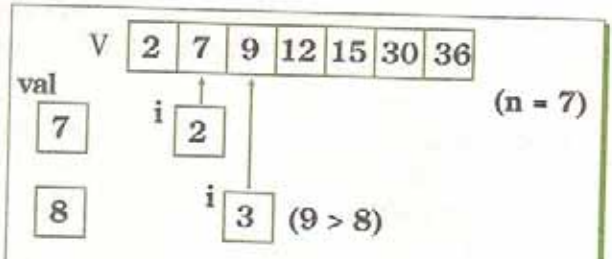


Figura 19. Căutarea într-un tablou ordonat

3.3. Căutarea cu metoda componentei marcaj

În cazul cel mai nefavorabil, căutarea unei valori într-un tablou (ordonat sau neordonat) cu n elemente necesită efectuarea a 2*n comparații (vezi ciclul cât timp de la pagina 56). Putem reduce complexitatea algoritmului de căutare astfel încât să efectuăm numai n comparații, chiar în cazul cel mai nefavorabil, folosind metoda componentei marcaj. Această metodă constă în memorarea valorii căutate în tablou pe poziția n + 1. Ca urmare, testul i <= n din structura cât timp nu mai are rost. Distincția dintre cele două situații — *valoarea se află în* tablou, respectiv, *valoarea nu se află în tablou* — se face prin testarea poziției pe care s-a găsit egalitatea. Dacă aceasta este n + 1, atunci evident valoarea nu se află inițial în tablou; dacă aceasta este cel mult n, atunci valoarea se află inițial în tablou și componenta marcaj reprezintă numai o repetare a acesteia.

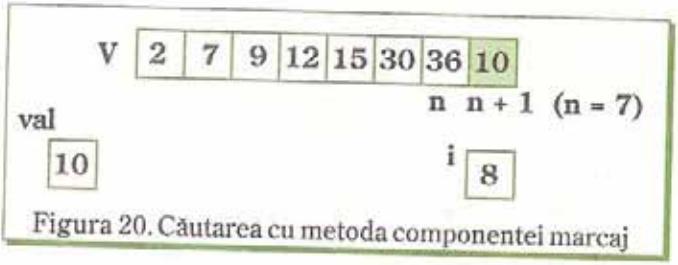


Figura 20. Căutarea cu metoda componentei marcaj

1. *Analiza problemei*

• *Date de intrare*
— numărul de elemente (n);
— valoarea căutată (val);
— tabloul în care se face căutarea (V).

• *Date de ieșire*
— indexul elementului din tabloul V a cărui valoare este egală cu valoarea căutată val; dacă valoarea nu a fost găsită, atunci se afișează un mesaj corespunzător.

2. *Reprezentarea algoritmului*

```
început căutare_componentă marcaj
// se consideră introduse cele n valori în vectorul V cu n + 1 elemente
scrie "Ce valoare căutăm?"
citește val
  // creăm componenta marcaj
  V[n + 1] ← val
  // inițiem parcurgerea tabloului
  i ← 1
  cât timp V[i] < val execută
        i ← i + 1
  sfârșit cât timp
  // verificăm la ce indice s-a încheiat parcurgerea
  dacă i <= n                    // nu s-a atins componenta marcaj
        atunci
          dacă V[i] = val        // am găsit valoarea
                atunci
                  scrie "Valoarea" val "a fost găsită pe poziția" i

                altfel                    // componenta curentă a depășit valoarea val
                  scrie "Valoarea" val "nu a fost găsită în tablou."
          sfârșit dacă
        altfel                    // i = n + 1, deci val nu apare decât ca marcaj
          scrie "Valoarea" val "nu a fost găsită în tablou."
  sfârșit dacă
sfârșit căutare_componentă marcaj
```

3. *Analiza eficienței*

Complexitatea algoritmului rămâne tot de ordinul O(n), dar pentru că tabloul este ordonat, se efectuează cel mult n + 1 comparații și asta numai în cazul în care val este strict mai mare decât cea mai mare valoare memorată în tablou (adică V[n+1] > V[n]). În celelalte cazuri (există o componentă i, $1 \le i \le n$, astfel încât V[i]= val sau există o componentă i, $1 \le i \le n$, astfel încât V[i] > val), numărul de comparații scade la i, indiferent dacă găsim sau nu valoarea căutată. Prin urmare, în cazul cel mai nefavorabil, complexitatea operației de căutare într-un tablou total ordonat rămâne de ordinul O(n), dar se fac doar n + 1 comparații în loc de 2*n.

3.4. Căutarea prin metoda Divide et Impera — căutarea binară

O metodă care reduce mult mai mult complexitatea operației de căutare a unei valori val într-un tablou V ordonat (crescător) este metoda „divide et impera", prin care domeniul de valori în care se caută soluția unei probleme este redus pas cu pas (*divide*) până la găsirea soluției (*impera*).

Inițial, se compară valoarea val cu valoarea unui element reper, anume cel din mijlocul tabloului:

— dacă valoarea căutată este mai mare decât a elementului din mijloc, e clar că îl vom căuta printre elementele aflate la dreapta acestuia; altfel, vom căuta la stânga lui;

— căutarea la dreapta sau la stânga o vom face la fel, alegând ca reper elementul din mijloc.

Exemplu:

Se consideră tabloul V cu valori ordonate crescător.

V	5	5	5	6	7	7	(8)	8	8	9	10	10	25	25
	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valoarea căutată este 10 (5 < 10 < 25).

Începem căutarea:

— localizăm elementul reper pe poziția 7;

— comparăm valoarea căutată cu valoarea din mijloc (10 > 8); rezultă că elementul căutat s-ar putea afla la dreapta;

continuăm căutarea:

8	8	9	(10)	10	25	25
8	9	10	11	12	13	14

— localizăm elementul reper pe poziția 11;

— comparăm valoarea căutată cu cea a elementului reper (10 = 10); rezultă că elementul căutat a fost găsit pe poziția 11 din doar două căutări.

1. *Raționamentul metodei*

Pentru scrierea algoritmului de căutare, folosim următoarele notații:

• p și u pentru adresa primului, respectiv, ultimului element din secvența de valori în care se face, la un moment dat, căutarea.

În exemplul nostru, la prima căutare, p = 1 și u = 14; la a doua căutare, p = 8 și u = 14;

• m pentru adresa elementului din mijlocul secvenței în care se face căutarea:

$$m = (p + u) \text{ div } 2$$

În exemplul nostru, la prima căutare, m = (1 + 14) div 2 = 7, iar la a doua căutare, m = (8 + 14) div 2 = 11.

Inițial, p și u au valorile 1, respectiv 14. După ce se compară valoarea căutată cu valoarea elementului din mijloc (m), dacă valoarea căutată este la dreapta lui m, atunci p devine m + 1; dacă valoarea căutată este la stânga lui m, atunci u devine m – 1.

Secvența de valori în care se face căutarea este delimitată de adresele p și u. Căutarea are sens doar dacă p < u. Dacă se ajunge la situația p > u, valoarea căutată nu există în secvență și se afișează mesajul corespunzător.

Raționamentul descris se aplică doar dacă valoarea căutată se poate afla printre elementele vectorului. De aceea, întâi se compară valoarea căutată cu primul, respectiv ultimul element din vector.

Dacă valoarea căutată se poate afla în vector, se intră în secvența căutărilor prin înjumătățire (*căutare binară*).

2. *Reprezentarea algoritmului*

```
început căutare_binară
citește val //valoare căutată
dacă (val >= V[1]) și (val <= V[N])
   atunci
        p ← 1
        u ← N
        găsit ← 0
        repetă
           m ← (p+u) div 2
           dacă val = V[m]
                atunci
                     găsit ← 1
                altfel
                     dacă val > V[m]
                        atunci
                             p ← m+1
                        altfel
                             u ← m-1
                     sfârșit dacă
           sfârșit dacă
        până când (găsit = 1) sau (p > u)
        dacă găsit = 0
           atunci scrie "valoare inexistentă"
           altfel scrie "valoarea a fost găsită pe poziția" p
        sfârșit dacă
   altfel
        scrie "valoare inexistentă"
sfârșit dacă
sfârșit cautare_binară
```

3. *Tabelul de variație*

Verificați algoritmul completând tabelul de variație pentru vectorul următor:

V	3	7	9	15	21	24	28	30	32	36	43	51	62	90

val	p	u	m	V[m]	Rezultatul căutării

Metoda căutării rapide prin înjumătățiri succesive se numește căutare binară și folosește tehnica cuceritorului: Divide et Impera (*Dezbină și stăpânește*).

Reţineţi! Divide et Impera

1. Divide et Impera este o tehnică de rezolvare a problemelor prin descompunere în probleme din ce în ce mai simple, până la o problemă cu soluţie imediată — soluţie parţială. Soluţia problemei iniţiale se formează prin compunerea soluţiilor parţiale.

2. Căutarea binară foloseşte metoda Divide et Impera. Problema iniţială — căutarea unei valori într-un tablou ordonat — este descompusă în probleme de acelaşi fel, dar din ce în ce mai simple, prin reducerea, până la un singur element, a intervalului de valori în care se face căutarea.

4. *Analiza eficienţei*

Complexitatea algoritmului este dată de numărul k de înjumătăţiri ale tabloului V necesare pentru a semnala existenţa/absenţa valorii căutate val. Pornim cu un tablou de n elemente — după prima înjumătăţire, numărul elementelor pe care are sens să le vizităm devine n/2, după a doua înjumătăţire acest număr devine n/4. Se demonstrează prin inducţie matematică completă că, după i segmentări, numărul elementelor rămase de vizitat devine $n/2^i$. În cazul cel mai nefavorabil, înjumătăţirea poate continua până când rămânem cu o singură componentă.

Prin urmare,

$1 = n/2^k \Rightarrow 2^k = n \Rightarrow$ numărul de segmentări necesare este $k = \log_2 n$.

La fiecare înjumătăţire, se execută operaţii de comparare şi atribuiri care nu afetează semnificativ ordinul de complexitate.

Prin metoda căutării binare, numărul de operaţii efectuate scade logaritmic.

TEME

1. Organizatorii unui eveniment cu premii au distribuit participanţilor (cel mult 100) câte un tichet pe care se află înscris un număr nu mai mare de 999. Fiecare participant este identificat prin numărul său de ordine. La sfârşit, posesorul tichetului cu numărul nr este declarat câştigător. Se poate întâmpla să nu existe nici un câştigător.

a) Descrieţi algoritmul necesar rezolvării acestei probleme.

b) Scrieţi programul corespunzător algoritmului propus.

c) Testaţi programul pentru următoarele date de test:

Date de intrare	Date de ieşire
n = 5 77, 56, 12, 6, 80 nr = 6	Câştigătorul are numărul 4.
nr = 9	Nu există câştigător.

2. După desfășurarea Olimpiadei de informatică a fost afișat clasamentul participanților, în ordinea descrescătoare a punctajului. Realizați un program care să afișeze pe ce poziție se află, în clasament, concurentul care a obținut punctajul p. În cazul în care nu s-a obținut un astfel de punctaj, se va afișa mesajul „punctaj inexistent".

3. Meteorologii de la stația meteo locală au înregistrat temperatura atmosferică timp de n zile (n <= 30). Determinați prima zi în care a fost înregistrată temperatura t. Dacă această temperatură nu a fost înregistrată în nici o zi, se va afișa un mesaj corespunzător.

4. Modificați algoritmul de căutare într-un vector ordonat astfel încât să obțineți un algoritm mai eficient pentru cazul în care valoarea căutată este mai mare decât oricare dintre valorile tabloului.

5. Justificați folosirea metodei *Divide et Impera* la sortarea prin selecție.

6. Modificați algoritmul de căutare binară pentru cazul în care vectorul este ordonat descrescător.

7. Demonstrați, prin inducție matematică completă, că după a i-a înjumătățire, dimensiunea tabloului de căutare este $n/2^i$.

Să presupunem că vrem să aplicăm acest algoritm unui tablou neordonat. Cum procedăm? Se justifică acest demers?

8. Un expert bancar a realizat *Topul 50* al celor mai bogați oameni de afaceri. Pentru păstrarea confidențialității, cei 100 de participanți la selecția pentru top au fost codificați cu numere de la 1 la 100.

a) Realizați un program care să determine rapid dacă participantul cu codul c este în top și pe ce poziție.

b) Modificați programul astfel încât, pentru fiecare set de date de test, să se afișeze numărul de înjumătățiri după care s-a ajuns la un rezultat.

MINIPROIECT: *Căutarea valorilor semnificative*

1. Realizați o listă cu date reale dintr-un domeniu de interes școlar (evenimente istorice, cote montane, populația unor orașe/țări, măsurători experimentale etc.).

2. Formulați împrejurările în care este necesară căutarea unei valori în lista realizată.

3. Specificați metoda de căutare propusă (în funcție de numărul de valori din listă, frecvența căutării, alte situații).

4. Justificați metoda prin analiza comparativă a eficienței metodelor de căutare.

5. Formulați câteva concluzii referitoare la utilizarea metodelor de căutare (când, de ce, cum alegem metoda).

6. Realizați programul complet care să rezolve cerința căutării unei valori în lista specifică domeniului ales.

4. ALGORITMUL DE INTERCLASARE

STUDIU DE CAZ: *DISPOZITIV MOBIL*

Firma Practic Soft are de programat un dispozitiv mobil care să ajute la rezolvarea următoarei probleme:

Într-o fabrică de încălțăminte există două secții de producție S1 și S2 și un singur compartiment (C) pentru controlul calității. În fiecare secție, pantofii sunt așezați în cutii. Dispozitivul mobil preia toate cutiile dintr-o secție și le așază pe banda controlorului de calitate. Apoi, sunt aduse toate cutiile din cealaltă secție — ordinea secțiilor nu contează (figura 21).

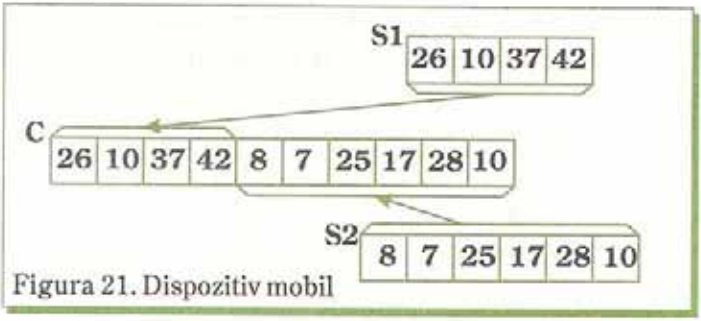


Figura 21. Dispozitiv mobil

Pentru început, controlorul de calitate are de verificat toate cutiile din fiecare măsură. Rezultatul verificării este înscris în raportul de calitate. Pentru a fi mai ușor de urmărit, raportul de calitate trebuie întocmit în ordinea crescătoare a măsurilor de pantofi (figura 22). Directorul fabricii de pantofi solicită firmei Practic Soft o soluție care să reducă timpul de lucru al controlorului de calitate.


RAPORT DE CALITATE

Măsură	Calificativ
7	FB
8	B
10	Respins
.....	
42	FB

Figura 22. Raport de calitate – model

1. *Analiza problemei*

➢ Dispozitivul mobil preia toate cutiile de pantofi dintr-o secție într-o ordine oarecare.

➢ Toți pantofii produși în cele două secții ajung pe banda controlorului de calitate.

➢ Ordinea în care sunt verificați pantofii diferă de ordinea în care se face înregistrarea în raportul de calitate.

Concluzie: Timpul de lucru al controlorului de calitate depinde de numărul de cutii verificate și de ordinea în care sunt verificate acestea.

2. *Soluția problemei*

➢ Fiecare secție va trimite cutiile în ordinea crescătoare a măsurilor de pantofi, câte o cutie pentru fiecare măsură.

➢ Dispozitivul va fi programat astfel încât să preia toate cutiile de pantofi din

fiecare măsură și să le așeze pe masa controlorului de calitate, în ordinea crescătoare a măsurilor.

➢ Dispozitivul va fi prevăzut cu trei brațe mobile, câte unul pentru fiecare dintre cele trei benzi: brațul i pentru banda S 1, brațul j pentru banda S 2, brațul k pentru banda C (figura 23).

➢ Fiecare cutie are o poziție sau adresă corespunzătoare benzii de pe care va fi preluată (S1 sau S2) sau benzii pe care va fi așezată (C).

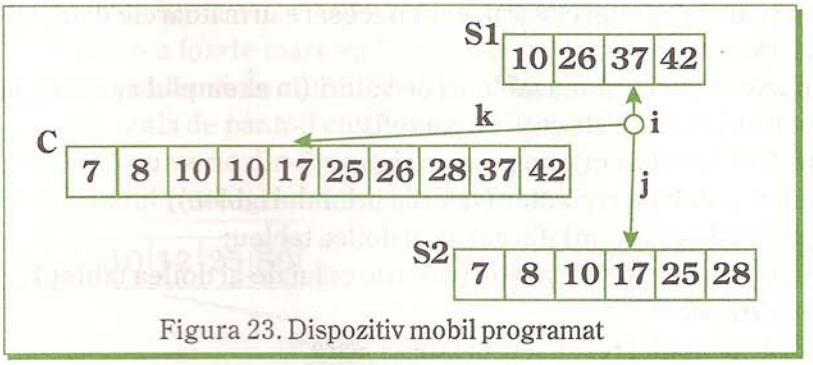


Figura 23. Dispozitiv mobil programat

3. *Raționamentul pas cu pas*

Pasul 1. Poziționarea celor trei brațe la prima adresă a fiecărei benzi este:

$$i \leftarrow 1$$
$$j \leftarrow 1$$
$$k \leftarrow 1$$

Pasul 2. Se compară cutiile aflate în dreptul celor două brațe, deci adresele i și j.

➢ Dacă măsurile coincid, atunci:

— una dintre cutii (o alegem pe cea de la adresa i) este așezată pe banda C la adresa brațului k;

— cele două brațe i și k ale dispozitivului mobil avansează spre dreapta.

$$i \leftarrow i + 1$$
$$k \leftarrow k + 1$$

➢ Dacă măsurile nu coincid, atunci:

— cutia cu numărul cel mai mic este așezată pe banda C la adresa brațului k;

— brațul k și brațul corespunzător benzii de pe care a fost preluată cutia se deplasează spre dreapta.

Se repetă pasul 2 cât timp există cutii atât pe banda S1, cât și pe banda S2.

Pasul 3. Dacă mai sunt cutii pe una dintre benzi, atunci:

— fiecare cutie este așezată în ordine pe banda C;

— brațul mobil k și brațul benzii pe care se mai află cutii se deplasează spre dreapta.

Se repetă pasul 3 cât timp mai sunt cutii pe bandă.

Programarea dispozitivului mobil folosește raționamentul numit *interclasarea a două șiruri de valori.*

Interclasarea este raționamentul prin care, pornind de la două tablouri de date ordonate după același criteriu, se obține un tablou nou ce conține fiecare dintre valorile celor două tablouri, ordonate după același criteriu.

4. *Organizarea datelor*

Pentru realizarea interclasării, sunt necesare următoarele date:

• *Date de intrare:*

— capacitatea celor două tablouri de valori (în exemplul analizat, pe fiecare bandă pot fi cel mult 100 de cutii de pantofi);

— numărul de valori (n) din primul tablou;

— n valori ordonate crescător (valorile primului tablou);

— numărul de valori (m) din cel de-al doilea tablou;

— m valori ordonate crescător (valorile celui de-al doilea tablou).

• *Date de ieșire:*

— valorile tabloului rezultat prin interclasare.

5. *Reprezentarea algoritmului*

```
început interclasare
S1[100] S2[100] C[200]  // se consideră citite datele de intrare
i ← 1   j ← 1   k ← 1
    cât timp (i <= n) și (j <= m) execută
        dacă S1[i] = S2[j] atunci
                                C[k] ← S1[i]
                                i ← i + 1
                            altfel
                                dacă S1[i] < S2[j] atunci
                                    C[k] ← S1[i]
                                    i ← i + 1
                                altfel
                                    C[k] ← S2[j]
                                    j ← j + 1
                                sfârșit dacă
        sfârșit dacă
                k ← k+1
    sfârșit cât timp
    cât timp i <= n execută  C[k] ← S1[i]
                             i ← i + 1   k ← k + 1
    sfârșit cât timp
    cât timp j <=m execută   C[k] ← S2[j]
                             j ← j + 1   k ← k + 1
    sfârșit cât timp
sfârșit interclasare
```

Observație: Ultimele două structuri cât timp nu se execută în cazul în care ultimele elemente din cele două tablouri sunt egale, adică pentru S1[n] = S2[m].

Pornind de la această observație, vom „inventa" o valoare foarte mare (mai mare decât ultimul element al fiecărui tablou — o valoare gen *infinit*), pe care o vom plasa după ultimul element al fiecărui tablou. Cu acest artificiu, ne-am asigurat că șirurile se termină în același timp.

Exemplu:

Valoarea foarte mare va fi aleasă în funcție de semnificația datelor din problemă. În studiul de caz, valoarea 50 este *foarte mare* — nu există o măsură reală de pantofi cu această valoare.

Din exemplul numeric din figura 24, observăm că valoarea foarte mare nu afectează rezultatul (este ignorată).

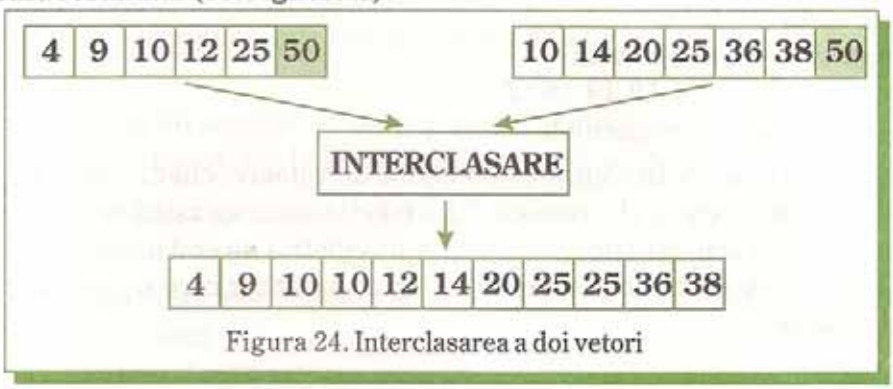


Figura 24. Interclasarea a doi vetori

TEME

1. Aplicați algoritmul de interclasare pentru exemplul numeric al problemei dispozitivului mobil.
2. În ce situație ultimele două structuri cât timp nu se execută niciodată? Rescrieți algoritmul folosind observația rezultată din această situație.
3. Construiți un exemplu numeric care să conducă la execuția ultimei structuri cât timp (după j).
4. Cum se modifică algoritmul de interclasare dacă șirurile sunt ordonate descrescător?
5. Realizați programul complet pentru introducerea șirurilor S1 și S2, obținerea și afișarea șirului C.
6. Modificați algoritmul prezentat pentru interclasarea a două șiruri S1 și S2 cu valori distincte (nici o valoare din S1 nu se găsește printre valorile lui S2).
7. Câte elemente are șirul rezultat din interclasarea a două șiruri cu valori distincte? Dar în cazul general (când se admit valori identice în cele două șiruri)?
8. Generalizați problema interclasării pentru mai multe șiruri de valori.

Probleme propuse

1. Se cunosc laturile a n cuburi. Să se construiască un turn stabil cu înălțimea mai mică sau egală cu h. Fiecare cub este identificat prin numărul său de ordine.

Exemplu:

Cuburile au laturile 4 3 7 9 5 6 14

Înălțimea h este 30 turnul va fi format din cuburile 7 4 3

Înălțimea h este 40 turnul va fi format din cuburile 7 4 3 6

2. Se cunoaște ziua în care este programat fiecare eveniment dintr-o listă cu n evenimente (n <= 15). Știind că un eveniment durează cel mult două zile, realizați un program care să planifice ordinea de desfășurare a evenimentelor. Fiecare eveniment va fi identificat prin numărul său de ordine în lista inițială.

Exemplu:

Pentru 5 evenimente avem:

— lista inițială 14 10 19 18 25

— planificarea evenimentelor 2 1 4 5

3. Pentru a păstra în siguranță obiectele de valoare, clienții unui hotel pot folosi seifurile cu cheie electronică. Cum funcționează un astfel de seif? Clienții primesc o cartelă magnetică pe care este înregistrat un cod format din n cifre. Parola sau cheia electronică este cel mai mic număr par care se poate forma cu cifrele codului.

Exemplu:

Codul este 1235667 parola va fi 1235676

Cerințe:

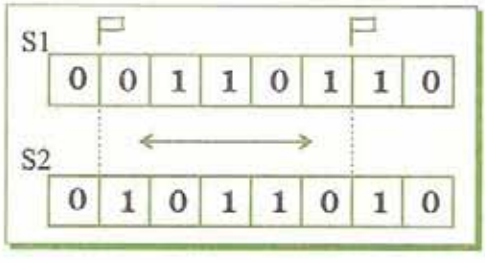
a) Realizați algoritmul prin care să fie obținută cheia electronică pentru un cod.

b) Întrucât fiecărui cod trebuie să-i corespundă o cheie, specificați condițiile pe care trebuie să le îndeplinească un cod valid (corect).

c) Realizați programul care să verifice validitatea codului și să obțină cheia corespunzătoare.

4. N copii, fete și băieți, sunt așezați în șir, unul după altul. Se plasează două steguțe: unul la stânga copilului de la adresa i, celălalt la dreapta copilului de la adresa j. Copiii egal depărtați de cele două steguțe, care sunt capetele intervalului (i, j), își schimbă locurile între ei.

Cunoscând două șiruri de valori 1 și 0 (1 – fete, 0 – băieți) de aceeași lungime n, stabiliți pozițiile în care vor fi plasate steguțele în primul șir astfel încât, după schimbarea locurilor, să se obțină al doilea șir.



5. Profesorul diriginte dorește o listă cu n fete și n băieți din care să formeze perechi pentru serviciul pe clasă (n număr natural par).

Ordinea elevilor în listă va fi aleasă astfel încât perechile să se formeze din primul și ultimul elev, al doilea și penultimul și așa mai departe. Nu va fi nici o pereche mixtă (fată–băiat).

Cerințe:

a) Cunoscând numele elevilor (prima literă) și tipul lor (fată sau băiat), să se formeze lista cerută de profesorul diriginte.

Exemplu:

$n = 4$	lista va fi
C M A D L O H P	C A L P D M O H
f f b b f b f b	

b) Să se formeze perechile elevilor de serviciu.

Exemplu:

Pentru exemplul considerat la punctul *a*, perechile vor fi:
C—H A—O L—M P—D (există mai multe soluții).

6. Se introduc n + m numere naturale.

Cerințe:

a) Să se formeze mulțimea A din primele n valori și mulțimea B din următoarele m valori.

b) Să se verifice dacă o valoare oarecare x, introdusă de la tastatură, aparține mulțimii A.

c) Să se formeze mulțimea R ca reuniune dintre mulțimile A și B.

7. Doi elevi lucrează împreună la un proiect pentru ora de istorie. Fiecare s-a documentat și a întocmit o listă cu cele mai importante evenimente istorice. În proiect, trebuie prezentat o singură dată fiecare eveniment istoric, în ordine cronologică.

Cerințe:

a) Întocmiți cele două liste cu câte cel mult 10 evenimente istorice.

b) Verificați dacă cele două liste pot fi interclasate.

c) Obțineți lista finală prin interclasarea celor două liste.

d) Să se determine dacă un an oarecare, *A*, aparține listei de evenimente. Justificați metoda de căutare folosită.

e) Realizați un program care să rezolve cerințele de la punctele *a*, *b*, *c* și *d*.

8. Se cunosc rezultatele la Olimpiada de informatică pentru trei școli. Numărul de participanți din fiecare școala este n_1, n_2, respectiv n_3.

Cerințe:

a) Propuneți un algoritm pentru obținerea clasamentului fiecărei școli.

b) Propuneți un algoritm prin care să se obțină clasamentul pentru toate școlile.

c) Analizați eficiența algoritmului propus pentru următoarele situații:

- $n_1 = n_2 = n_3$;
- $n_1 > n_2 > n_3$.

9. Se cunosc două șiruri de valori reprezentând coordonatele a n_1, respectiv n_2 puncte egal depărtate de axele unui sistem *xOy*.

Cerințe:

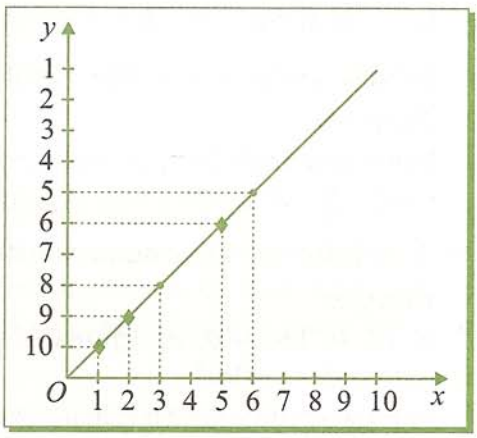
a) Să se determine dacă un punct oarecare *X* se află printre cele n_1 sau n_2 puncte cunoscute.

b) Să se aranjeze toate punctele pe semidreapta cu originea în punctul *O* căreia îi aparțin.

Exemplu:

$n_1 = 3$ punctele marcate prin ♦

$n_2 = 2$ punctele marcate prin •



10. Doi copii primesc câte un alfabetar A1, respectiv A2 (set de cartonașe cu literele alfabetului). După ce au așezat toate cartonașele în ordine alfabetică, copiii au constatat că nici unul dintre ei nu are alfabetul complet.

Cerințe:

a) Să se determine dacă o literă oarecare L aparține alfabetarului A1 sau A2.

b) Să se construiască un alfabetar nou A12 din literele alfabetarelor A1 și A2.

c) Să se verifice dacă alfabetarul A12 este complet.

APLICAȚII INTERDISCIPLINARE SPECIFICE PROFILULUI

1. VARIABILE ALEATOARE. VALORI MEDII

Fie X o variabilă aleatoare (legată de o experiență concretă: aruncarea unei monede, aruncarea unui zar, extragerea unei bile de o anumită culoare dintr-o urnă care conține bile de diferite culori etc.) și fie $x_1, x_2, \ldots, x_n$ valorile pe care le poate lua această variabilă, cu probabilitățile $p_1, p_2, \ldots, p_n$.

Exemplu:

Dacă X este variabila aleatoare legată de aruncarea unui zar cu 6 fețe, atunci valorile pe care le poate lua X sunt 1, 2, 3, 4, 5, 6, iar probabilitățile cu care X poate lua aceste valori sunt toate egale între ele și egale cu $\frac{1}{6}$.

Obținem astfel distribuția variabilei aleatoare X cu $x_i = i$ și $p_i = 6, \forall\, i = \overline{1, n}$.

Se numește *valoare medie* a unei variabile aleatoare X numărul real:

$$M(X) = x_1p_1 + x_2p_2 + \ldots + x_np_n.$$

Dacă numărul cazurilor observate este foarte mare, valoarea medie a unei variabile aleatoare X este aproximativ egală cu media aritmetică a valorilor observate. În exemplul de mai sus,

$$M(X) = \frac{1}{6}(1+2+\ldots+6) = \frac{1}{6}\cdot\frac{6\cdot 7}{2} = \frac{7}{2},$$

deci, numărul mediu de puncte pe care îl obținem la aruncarea unui zar cu 6 fețe este aproximativ egal cu 3.

Aplicație: *În lift*

Să presupunem că urcăm în liftul unui bloc cu 9 etaje împreună cu o altă persoană. Nu știm la ce etaj vrea să coboare această persoană, dar putem descrie acest lucru printr-o variabilă aleatoare (care ia valorile 1, 2, ... , 9 — fiecare cu probabilitatea 1/9). Calculați valoarea medie a acestei variabile aleatoare. Generalizare pentru n etaje și k persoane.

1. *Analiza problemei*

Datele problemei:

- *Date de intrare*
  - — numărul de etaje (n), valoare întreagă;
  - — numărul de persoane (k), valoare întreagă.
- *Date de ieșire*
  - — valoarea medie (m).

2. Raționamentul de rezolvare

Variabilele aleatoare X_1, X_2, ... , X_k asociate comportamentului celor k persoane sunt independente. Prin urmare, variabila aleatoare care descrie comportamentul lor este $X = X_1 + X_2 + ... + X_k$, iar valoarea ei medie este:

$$M(X) = M(X_1) + M(X_2) + ... + M(X_k).$$

Cele k variabile iau aceleași valori $(1, 2, ..., n)$ cu aceleași probabilități $\frac{1}{n}$, deci:

$$M(X_1) = M(X_2) = ... = M(X_k) = \frac{1}{n} * \frac{n*(n+1)}{2} = \frac{n+1}{2}.$$

De unde rezultă că:

$$m = M(X) = k * M(X_1) = k * \frac{n+1}{2}.$$

TEMĂ

Realizați algoritmul și programul corespunzător raționamentului propus.

2. SERII DE VALORI

Aplicație: *La cabinetul medical*

La cabinetul medical al școlii, au fost măsurați n elevi. Înălțimea elevilor (exprimată în centimetri) a fost înregistrată într-un tabel (vezi tabelul 26 pentru n = 10).

Tabelul 26

Elevul	1	2	3	4	5	6	7	8	9	10
Înălțimea (cm)	162	159	173	186	162	173	162	186	162	170

Cerințe:

a) Să se determine înălțimea minimă, maximă și, respectiv, medie a grupului.

b) Să se determine numărul de elevi cu aceeași înălțime — frecvența absolută (vezi tabelul 27).

c) Să se determine, pentru fiecare înălțime în parte, frecvența înălțimii din grup, raportată la numărul de elevi — frecvența relativă (vezi tabelul 27).

d) Să se determine numărul de elevi, pe intervale de înălțimi — frecvența absolută pe clase de valori (vezi tabelul 28).

Tabelul 27

Înălțimea (cm)	Frecvența absolută (f)	Frecvența relativă (fr)	Frecvența absolută cumulată crescător (fc)	Frecvența absolută cumulată descrescător (fd)
159	1	1/10	1	10
162	4	4/10	5	9
170	1	1/10	6	5
173	2	2/10	8	4
186	2	2/10	10	2

e) Să se determine numărul de elevi cu înălțimea mai mică decât o înălțime h specificată (vezi tabelul 27).

f) Să se determine numărul de elevi cu înălțimea mai mare decât o înălțime h specificată (vezi tabelul 27).

Tabelul 28

Grupe de înălțimi	Frecvența absolută pe clase de valori (fv)
150-160	1
160-170	4
170-180	3
180-190	2

TEME

1. Realizați câte un program care să rezolve, pe rând, fiecare dintre cerințele *a-f.*

2. Pentru fiecare cerință, justificați soluția de memorare și organizare a datelor.

3. a) Formulați exemple de probleme a căror rezolvare să conducă la determinarea unor mărimi specifice seriilor de valori.

b) Construiți date de test corespunzătoare exemplului formulat.

c) Realizați și testați programul pentru rezolvarea problemei din exemplul formulat.

3. DETERMINAREA VALORII UNUI POLINOM

Fie polinomul $P(X) = 3 * X^6 + X^5 - 17 * X^4 + 5 * X^2 - X + 0{,}75$.

Să se calculeze valoarea polinomului într-un punct oarecare pe care îl vom nota cu a (de exemplu, pentru $a = 3{,}14$).

Generalizare pentru un polinom de grad n, $n \geq 2$, *cu coeficienți reali.*

1. *Analiza problemei*

Datele problemei:

• *Date de intrare*

— punctul real (a) în care se calculează valoarea polinomului;

— gradul polinomului (n);

— vector (p) de dimensiune n + 1 care conține coeficienții polinomului.

• *Date de ieșire*

— valoarea polinomului (v).

2. *Raționamentul de rezolvare*

Pentru a calcula valoarea polinomului

$$P(X) = \alpha_n * X^n + \alpha_{n-1} * X^{n-1} + \ldots + \alpha_1 * X + \alpha_0$$

într-un punct *a*, putem folosi două metode:

➢ metoda clasică — presupune însumarea monoamelor $\alpha_{i-1} * a^{i-1}$, unde *a* este punctul dat, iar $1 \leq i \leq n$;

➢ schema lui Horner — presupune rescrierea polinomului sub forma

$$P(X) = (((\alpha_n * X + \alpha_{n-1}) * X + \alpha_{n-2}) * X + \ldots + \alpha_1) * X + \alpha_0$$

și calcularea valorii în punctul *a* dat.

3. *Reprezentarea algoritmului*

```
începeput Horner;
// se lucrează cu variabilele n, a, i, vectorul p[21] cu valori reale
scrie ("Introduceți n <= 20, gradul polinomului"); citește (n);
scrie ("Introduceți a = punctul în care se calculează"); citește (a);
scrie ("Introduceți coeficienții polinomului, începând cu cel mai semnificativ");
citește (p[n + 1]);
v = p[n + 1];
pentru i = n la 1 execută
        citește (p[i]);
            v = v * a + p[i];
 sfârșit pentru
scrie ("Valoarea polinomului este v = ", v);
sfârșit Horner
```

TEME

1. Codificați algoritmul propus în limbajul de programare studiat.
2. Rescrieți programul astfel încât pentru citirea coeficienților să se folosească o singură variabilă.
3. Justificați introducerea coeficienților începând cu cel mai semnificativ: p[n + 1].
4. Construiți algoritmul și programul corespunzător pentru calcularea valorii P(a) prin însumarea termenilor.
5. Calculați și comparați complexitatea celor doi algoritmi.

4. CALCULE COMBINATORIALE

4.1. Diagonale

Se calculează numărul diagonalelor unui poligon convex cu **n** *vârfuri.*

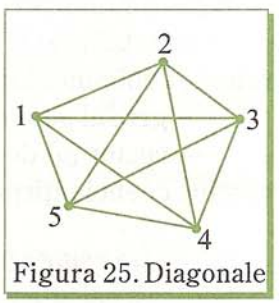


Figura 25. Diagonale

1. *Analiza problemei*

Datele problemei:

- *Date de intrare*
  — numărul de vârfuri ale poligonului (n).
- *Date de ieșire*
  — numărul diagonalelor poligonului (d).

2. *Raționamentul de rezolvare*

Fixăm un vârf al poligonului. Din el pot pleca numai n – 3 diagonale deoarece trebuie să excludem cele două vârfuri vecine și pe el însuși (figura 25). Cum vârful este oarecare, rezultă că am avea n * (n – 3) diagonale. De fapt, fiecare diagonală este numărată de două ori, adică pentru fiecare dintre extremitățile sale. Rezultă că numărul diagonalelor, d, este dat de relația:

$$d = (n * (n - 3) / 2).$$

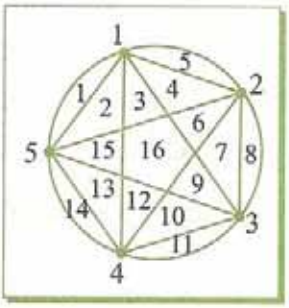
TEME

1. Reprezentați în pseudocod algoritmul de rezolvare a problemei.

2. Codificați algoritmul în limbajul de programare studiat.

3. Dezvoltați algoritmul astfel încât, printr-o singură execuție de program, să poată fi determinat numărul diagonalelor pentru mai multe poligoane.

4. Se cunosc n puncte amplasate pe circumferința unui cerc. Punctele au următoarea proprietate: nu există trei corzi cu capetele în aceste puncte care să fie concurente într-un punct interior cercului. Între oricare astfel de puncte se trasează corzile corespunzătoare. Să se determine numărul de zone (z) mărginite de corzi sau de circumferința cercului.



Indicație: $z = n + 1 + n * (n - 3)/2 + C_n^4$.

4.2. Loto (6 din 49)

Un jucător la «6 din 49» și-a propus să determine toate variantele posibile de joc. Întrucât numărul variantelor este foarte mare, jucătorul vrea să rezolve problema folosind calculatorul. Generalizare pentru un joc «y din x».

1. *Raționamentul de rezolvare*

Numărul total de variante este chiar numărul de submulțimi de *y* elemente care se pot forma dintr-o mulțime cu *x* elemente. Acest număr corespunde *combinărilor de x luate câte y*, (C_x^y). Pentru cazul particular *«6 din 49»*, avem $x = 49$ și $y = 6$.

Matematic avem: $C_x^y = \dfrac{x!}{y!(x-y)!}$ (1).

Calculul factorialelor (49!) conduce la lucru cu numere uriașe (*huge*) care depășesc capacitatea de memorare pentru un întreg foarte mare (2 147 483 647).

De aceea, vom folosi altă metodă de calcul:

$$C_x^k = (x - k + 1)/k * C_x^{k-1} \quad (2)$$

Relația (2) este o relație recurentă într-un pas (valoarea calculată la pasul k depinde de valoarea calculată la pasul $k-1$); $C_x^0 = 1$.

2. *Reprezentarea algoritmului*

```
începput combinari
// se lucrează cu variabilele x, y, k, c
citește x, y
c ← 1
pentru k = 1 la y execută
        c ← (x - k + 1) * c div k
sfârșit pentru
scrie c
sfârșit combinari
```

TEME

1. Codificați algoritmul în limbajul de programare studiat.

2. Modificați algoritmul astfel încât valorile calculate la fiecare pas (iterație) să fie memorate și afișate la sfârșitul prelucrării.

Indicație: Se va folosi un vector C; la fiecare iterație, se calculează

$$C[k] = (x - k + 1) * C[k - 1] \text{ div } k$$

5. DETERMINAREA UNOR MĂRIMI FIZICE DINTR-UN CIRCUIT ELECTRIC

Se dau rezistențele R_1 = 3 W, R_2 = 4 W, R_3 = 5 W legate în serie. Se calculează rezistența echivalentă a acestei grupări. Generalizare pentru n rezistențe legate în serie.

1. *Analiza problemei*

Datele problemei:

- *Date de intrare*
  - — numărul de rezistențe (n);
  - — valorile celor n rezistențe.
- *Date de ieșire*
  - — rezistența echivalentă a circuitului (R_s).

2. *Raționamentul de rezolvare*

Pentru a calcula rezistența echivalentă a circuitului serie se folosește formula:

$$R_s = \sum_{i=1}^{n} R_i .$$

Se introduc de la tastatură, numărul de rezistențe și valorile acestora. Valoarea fiecărei rezistențe este însumată în variabila R_s.

TEME

1. Realizați și testați programul pentru calculul rezistenței echivalente.

2. Se cunosc valorile a n rezistențe și se dorește legarea în serie a primelor p rezistențe. Determinați rezistența echivalentă pentru o grupare de p rezistențe, unde $p \in \{2, \ldots, n\}$ este o valoare introdusă de la tastatură ($p = 0$ semnifică sfârșitul prelucrării).

3. Justificați soluția de memorare a datelor, pentru rezolvarea cerinței precedente.

4. Rezolvați cerințele 1, 2 și 3 pentru o grupare de rezistențe legate în paralel.

Indicație: $\frac{1}{R_p} = \sum_{i=1}^{n} \frac{1}{R_i}$.

6. APLICAȚII DIN GENETICĂ

6.1. Legea Hardy-Weinberg

Pentru o populație în care nu se produc mutații și în care nu are loc selecția, proporția diferitelor gene rămâne constantă de la o generație la alta (conform Legii Hardy-Weinberg). Ne propunem să determinăm frecvența genotipurilor **AA**, **Aa** *și* **aa** *într-o generație nouă (unde* **A** *și* **a** *reprezintă formele alternative — **alelele** — unei gene) și să verificăm astfel legea. Cunoaștem* **p** *= frecvența alelei* **A** *și* **q** *= frecvența alelei* **a** *(de exemplu:* **p** *= 0,75 și* **q** *= 0,25).*

1. *Analiza problemei*

Datele problemei:

- *Date de intrare*
  - — frecvența alelei A (p);
  - — frecvența alelei a (q).
- *Date de ieșire*
  - — frecvențele genelor și un mesaj corespunzător pentru verificarea legii Hardy-Weinberg.

2. *Raționamentul de rezolvare*

Se introduc cele două probabilități p și q.

Se verifică p + q = 1.

Se calculează frecvența genei *A*: Sp = p * p + p * q.

Se calculează frecvența genei *a*: Sq = q * q + q * p.

Se verifică dacă legea este respectată S = Sp + Sq = 1.

Se afișează frecvențele Sp, Sq și un mesaj corespunzător: *„Legea este/nu este respectată"*.

TEME

1. Descrieți un algoritm de rezolvare a problemei propuse.
2. Codificați algoritmul în limbajul de programare studiat.
3. Testați programul pentru mai multe cazuri (date de test).
4. Justificați, prin demonstrație matematică, corectitudinea algoritmului.

Indicație: Dacă (a + b) = 1, atunci $(a + b)^2 = 1$.

6.2. Roiul de albine — arborele de familie

Albinele trăiesc în familii mari numite roiuri. În fiecare roi, există o regină, albine lucrătoare și trântori. Trântorii se nasc din ouăle nefecundate ale reginei. Se poate spune că ei nu au tată. În figura 26 este prezentată evoluția arborelui de familie pentru o albină trântor.

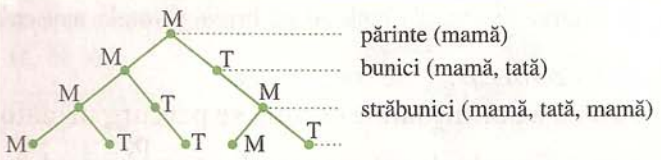


Figura 26. Roiul de albine — arborele de familie pentru o albină trântor

Cerință:

Să se determine numărul de ascendenți ai albinei trântor (t) pentru fiecare dintre cele k generații din familia sa.

1. *Raționamentul de rezolvare*

Ținând seama de legea naturală după care se înmulțesc albinele trântor, observăm că numărul ascendenților se înscrie în următorul șir de valori:

1, 1, 2, 3, 5, 8, ...

Prima valoare 1 din șir reprezintă chiar albina trântor t.

Numărul ascendenților dintr-o generație k se obține ca sumă a numărului de ascendenți din generațiile $k-1$ și $k-2$. Acest șir poartă numele de *șirul lui Fibonacci.*

$$F(k)=\begin{cases}1, & \text{pentru } k=1,2\\ F(k-1)+F(k-2), & \text{pentru } k>=3\end{cases}.$$

TEME

1. Construiți algoritmul de rezolvare astfel încât numărul ascendenților fiecărei generații de la 1 la k să fie păstrat (se va lucra cu un vector F).

2. Codificați algoritmul în limbajul de programare studiat.

3. Realizați un program care să determine ascendenții pentru o albină lucrătoare.

4. Realizați un program care să determine dacă un număr x corespunde numărului de ascendenți din familia albinei trântor sau din familia albinei lucrătoare.

5. Prezentați și alte exemple (aplicații) ale *șirului lui Fibonacci.*

7. APLICAȚII DIN CHIMIA ORGANICĂ

Formula moleculară a unei substanțe

Pentru o substanță organică formată din s elemente E^1, E^2,...E^s se cunosc:

— *procentul în care se găsește fiecare element p_1, p_2,..., p_s [%] ;*

— *masa moleculară corespunzătoare fiecărui element A_1, A_2,..., A_s [g/mol];*

— *masa moleculară a substanței M [g/mol];*

— *valența atomică a fiecărui element v_1, v_2,..., v_s.*

Se determină formula moleculară brută, formula moleculară și nesaturarea echivalentă.

Raționamentul de rezolvare

Pentru a obține datele cerute, se parcurg următorii pașii:

Pas 1. Se calculează rapoartele: $r_i=\frac{p_i}{A_i}$, $i=\overline{1,s}$.

Pas 2. Se determină valoarea minimă a rapoartelor r_i, notată cu min.

Pas 3. Se recalculează rapoartele: $r_i = \frac{p_i}{min}$, $i = \overline{1,s}$.

Pas 4. Se scrie formula moleculară brută a substanței: $(E^1_{r1}, E^2_{r2}, ..., E^s_{rs})_n$.

Pas 5. Se determină coeficientul din formula moleculară brută, utilizând formula de mai jos prin:

$$n = rot\left(\frac{M}{\sum_{i=1}^{s} A_i r_i}\right),$$

unde rot(m) reprezintă rotunjirea numărului real pozitiv m la cel mai apropiat întreg.

Exemplu: Dacă m = 2,48, atunci m = 2, iar dacă m = 3,72, atunci m = 4.

Pas 6. Se recalculează rapoartele: $r_i = r_i * n$, $i = \overline{1,s}$

Pas 7. Se scrie formula moleculară $E^1_{r1}, E^2_{r2}, ..., E^s_{rs}$.

Pas 8. Se calculează nesaturarea echivalentă:

$$NE = \frac{2 + \sum_{i=1}^{s} r_i(v_i - 2)}{2}$$

TEMĂ

Realizați algoritmul și programul corespunzător raționamentului propus.

Din fișierul formula.in se citesc de pe prima linie două numere naturale s și M, care reprezintă *numărul de elemente ale unei substanțe*, respectiv, *masa moleculară*. De pe următoarele s linii se citesc informațiile corespunzătoare fiecărui element, separate prin câte un spațiu: *simbolul, procentul, masa atomică* și *valența*.

În fișierul formula.out se vor afișa pe linii distincte: *formula moleculară brută, formula moleculară* și *nesaturarea echivalentă*.

Exemplu:

formula.in
3 180
C 40 12 4
H 6.67 1 1
O 53.33 16 2

formula.out
Formula moleculară brută:
$(C H_2 O)_n$
Formula moleculară:
$C_6 H_{12} O_6$
NE = 1

Se obține formula *glucozei*.

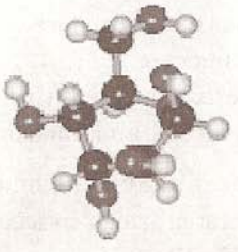


Figura 27. Glucoză $C_6H_{12}O_6$

CUPRINS

Profil real — specializările: matematică-informatică;
ştiinţe ale naturii.

ISBN: 973-653-746-3

Preţ: 2,58 lei