Radu Mircea MORARIU-GLIGOR

Iuliana Fabiola MOHOLEA

Florina Maria ŞERDEAN

PROGRAMARE ÎN LIMBAJUL C CU APLICAŢII ÎN INGINERIE MECANICĂ

*

Volumul I

UTPRESS
Cluj - Napoca, 2021
ISBN 978-606-737-550-3

Editura U.T.PRESS
Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: ing. Călin Câmpean

Recenzia: Prof.Dr.Ing. Tiberiu Alexandru Antal
Conf.Dr.Ing. Ovidiu Aurelian Deteșan

ISBN 978-606-737-549-7
ISBN 978-606-737-550-3 Volumul 1
Bun de tipar: 20.12.2021

Cuprins:

Prefaţă

„Toată lumea din această țară ar trebui să învețe să programeze un computer, pentru că te învață să gândești".

Steve Jobs

Într-adevăr, învățarea unui limbaj de programare dezvoltă mult gândirea logică a unei persoane. Oamenii care pot gândi logic sunt capabili să analizeze problemele și să elaboreze soluții. Acest lucru nu este valoros doar atunci când se dezvoltă programe, ci este vital pentru orice situație care necesită o gândire rațională. Iar inginerii au nevoie de această aptitudine intelectuală pe lângă gândirea tehnică.

Cartea se adresează studenților de la specializările facultăților cu profil mecanic şi tuturor celor care doresc să îşi îmbunătăţească cunoştinţele în domeniul programării calculatoarelor. Cartea își propune să reprezinte un instrument util în activitatea de înțelegere și învățare a unor algoritmi de programare, utilizând un limbaj uzual – limbajul C.

Această carte adoptă o abordare hibridă, urmărind înţelegerea şi învăţarea algoritmilor şi în paralel, programarea în limbajul C. Pentru studenți este imposibil să prevadă dacă va trebui să cunoască în viitor conceptele de programare sau dacă este suficient să cunoască utilizarea unor programe dedicate. Prin urmare, considerăm că cea mai potrivită abordare este aceea de a le oferi studenților atât conceptele de programare cât și noţiunile specifice pentru învăţarea unui limbaj de programare. Deoarece limbajul C este foarte răspândit, se deschide astfel calea pentru învăţarea şi a altor limbaje de programare în vederea rezolvării unor probleme tehnice.

Prin structura sa, această carte prezintă în mod gradual atât noțiunile teoretice cât și problemele practice care utilizează aceste noţiuni, pornind de la simplu spre complex. Lucrarea este structurată în opt capitole, urmate de bibliografie și două anexe.

Primul capitol include o prezentare compactă a limbajului de programare C, accentuând elementele necesare scrierii unor programe de complexitate scăzută sau medie în acest limbaj de programare.

Al doilea capitol prezintă elementele care stau la baza reprezentării unui algoritm, atât prin schemă logică, cât și prin limbaj pseudocod.

Al treilea capitol conține o serie de probleme de complexitate redusă, probleme care parcurse succesiv permit asimilarea treptată a cunoștințelor necesare studenților pentru a înțelege cum se abordează o problemă, cum se stabilesc datele de intrare, de ieşire, şi cele intermediare, respectiv cum se construieşte un algoritm şi cum se obţine soluția.

Capitolul patru este dedicat rezolvării problemelor cu funcţii, relaţia de definiţie a funcţiei depinzând de evaluarea unor condiţii. De asemenea, calculul valorilor funcţiei poate fi realizat pentru un interval cunoscut, parcurs cu un pas cunoscut.

Capitolul cinci tratează prelucrarea elementelor tablourilor unidimensionale, fiind prezentate o serie de probleme considerate clasice: sume, produse, numărarea elementelor care îndeplinesc o anumită cerinţă, ordonarea elementelor, căutarea, inserarea sau eliminarea unor elemente, probleme cu cifrele unui număr natural. De asemenea, sunt prezentate probleme cu o complexitate mai ridicată, dar foarte utile, cum ar fi: verificarea CNP-ului, verificarea codului de pe card-ul bancar sau a codului ISBN.

În capitolul şase al lucrării sunt prezentate probleme legate de prelucrarea elementelor tablourilor bidimensionale (matrice): sume, produse, numărarea unor elemente care îndeplinesc o anumită cerinţă, determinarea elementului maxim sau minim, eliminarea sau inserarea unor linii sau coloane. De asemenea, sunt tratate matricele pătratice, precum şi generarea unor matrice particulare, ale căror elemente se obţin pe baza unor anumite reguli.

Capitolul şapte tratează probleme legate de şiruri de numere reale, serii de numere şi dezvoltări în serie de puteri.

Ultimul capitol conţine o colecţie de probleme tehnice din geometrie, mecanică tehnică (statică, cinematică şi dinamică), organe de maşini sau rezistenţa materialelor. Aceste probleme ilustrează în mod foarte sugestiv modul în care o serie de probleme tehnice pot fi rezolvate prin scrierea unor programe într-un anumit limbaj de programare.

Considerăm că programarea se învață prin exemple. De aceea, această lucrare abundă în exemple sugestive și probleme rezolvate pas cu pas, atât probleme clasice de programare cât și probleme de natură tehnică.

Autorii

Capitolul 1. Limbajul de programare C

1.1. Introducere

1.1.1. Etapele de rezolvare a unei probleme tehnice cu ajutorul calculatorului (folosind un limbaj de programare)

A. Stabilirea datelor iniţiale (de pornire) şi a modelului matematic pentru problema de rezolvat;
B. Întocmirea unui algoritm pentru rezolvarea problemei (utilizând limbajul pseudo-cod sau schema logică);
C. Scrierea fişierului sursă utilizând un limbaj de programare (în acest caz limbajul **C**), cu ajutorul unui **editor de texte**. Fişierul rezultat va avea extensia **.C** sau **.CPP**;
D. Compilarea fişierului sursă şi obţinerea fişierului **obiect** (fişier cu extensia **.OBJ**). Această operaţie este realizată de către **compilator** care transcrie fişierul sursă în cod obiect;
E. Link-editarea (ediţia de legături). În această etapă se leagă fişierul sursă de bibliotecile de funcţii, făcându-se legătura între modulele obiect. În urma acestei etape rezultă un fişier executabil (extensia **.EXE**);
F. Lansarea în execuţie a fişierului executabil (rularea programului).

Un **mediu de dezvoltare** (engl. *development environment*, sau *integrated development environment* – „mediu integrat de dezvoltare") este un program care grupează mai multe programe independente sub o interfaţă unică, care ajută programatorul în scrierea altor programe. Un mediu de dezvoltare combină toţi paşii necesari creării unui program (ex.: editarea codului sursă, compilarea, depanarea, testarea, generarea de documentaţie) într-un singur soft, care de regulă, oferă o interfaţă grafică, prietenoasă cu utilizatorul.

Principalele componente ale unui mediu de dezvoltare sunt: editorul de text sursă, compilatorul, link-editorul şi depanatorul. Mediile de dezvoltare apelează compilatoare sau interpretoare, care pot veni în acelaşi pachet cu mediul însuşi sau pot fi instalate separat de către programator. Printre facilităţile prezente în mediile de dezvoltare mai sofisticate se numără: exploratoare de cod sursă, sisteme de control al versiunilor, instrumente de generare a interfeţelor grafice, sau unelte de ingineria programării. De obicei, un mediu de dezvoltare este specific unui anumit limbaj de programare, însă există şi medii de dezvoltare care pot lucra cu mai multe limbaje, de ex. **Code::Blocks** sau **Microsoft Visual Studio**.

1.1.2. Tipuri de limbaje de programare. Clasificare.

Limbajul de programare reprezintă *un sistem de convenţii adoptate pentru realizarea unei comunicări între programator şi calculator.*

Limbajele de programare sunt foarte asemănătoare limbajelor naturale, astfel ele sunt compuse din: cuvinte rezervate, punctuaţie, propoziţii şi fraze, reguli sintactice, etc. Aşa cum pentru învăţarea unei limbi străine este necesar să se înveţe cuvintele acesteia şi regulile prin intermediul cărora acestea pot fi manevrate, pentru învăţarea unui limbaj de programare trebuie studiate cuvintele şi semnele care îl compun, precum şi ansamblul de reguli pentru manevrarea acestora.

După modul în care este conceput ansamblul de reguli de comunicare, limbajele de programare se clasifică astfel:
A. Limbaje de nivel scăzut – nivel înalt: Nivelul unui limbaj este dat de poziţia pe care acesta îl ocupă pe o scară de la nivelul recunoscut de microprocesor (limbaj maşină) şi până la limbajul natural al programatorului (limba engleză, etc). Un limbaj de nivel scăzut este un limbaj foarte apropiat de limbajul maşină şi care lucrează cu elemente de nivel hardware, cum ar fi: regiştrii, microprocesor, locaţii de memorie, porturi de intrare/ieşire etc. Un limbaj de nivel înalt utilizează concepte apropiate de limbajul natural, concepte de nivel logic, cum ar fi: colecţii de date, nume de operaţii, variabile, constante etc.

O deosebire extrem de importantă între cele două tipuri de limbaje de programare o constituie portabilitatea acestora, adică posibilitatea transferării programelor pe un alt tip de platformă (procesor + sistem de operare) decât cea pe care au fost realizate.

Limbajele de nivel scăzut sunt neportabile, deoarece sunt legate de tipul procesorului utilizat de maşină. În ceea ce priveşte limbajele de nivel înalt, acestea permit transferul programelor deoarece între program şi calculator se interpune compilatorul care rezolvă transformarea fişierului sursă în fişier executabil, ţinând cont de caracteristicile maşinii de calcul.

B. Limbaje procedurale – neprocedurale: Cele două tipuri de limbaje se deosebesc prin nivelul de organizare (structurare) a programelor. În cazul limbajele neprocedurale programele sunt gândite la nivel de instrucţiune pe când la cele procedurale programatorul concepe programele la nivel de bloc de instrucţiuni. În limbajele procedurale programele sunt scrise instrucţiune cu instrucţiune, însă ele sunt organizate logic în blocuri (grupuri de instrucţiuni) care realizează acţiuni specifice.

C. Limbaje orientate sau de uz general: Din acest punct de vedere, limbajele pot fi orientate pe o anumită problemă sau pot fi concepute pentru soluţionarea oricărui tip de problemă.

Un exemplu de limbaj de nivel scăzut este limbajul de asamblare. Limbaje de nivel înalt sunt: **BASIC** (**B**eginner's **A**llpurpose **S**ymbolic **I**nstruction **C**ode - Cod de instrucţiuni simbolice, de uz general, destinat începătorilor), **FORTRAN** (**FOR**mula **TRAN**slation), **PASCAL** (numele acestui limbaj provine de la matematicianul şi filosoful BLAISE PASCAL, în semn de recunoaştere a contribuţiilor sale în conceperea maşinilor de calcul), **ADA** (denumit după **Augusta Ada Byron**, contesa de Lovelace, asistenta lui Charles Babbage, acesta fiind considerat primul programator din lume), precum şi limbajul **C**.

1.1.3. Limbajul C. Scurt istoric.

La un anumit moment s-a pus problema conceperii unui sistem de operare universal, care să poată funcţiona, teoretic, pe orice tip de platformă – sistemul **Multics** / **UNIX**. Pentru aceasta era nevoie de un limbaj care să exploateze toate posibilităţile unei maşini de calcul, dar care nu putea fi limbajul de asamblare deoarece acesta este specific maşinii de calcul, astfel că o nouă implementare presupunea rescrierea integrală a sistemului.

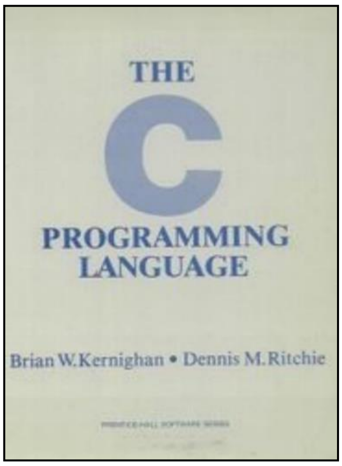
Limbajul **C** a apărut în anul 1972, autorii acestuia fiind Brian W. Kerningham şi Dennis M. Ritchie de la Bell Laboratories. Limbajul **C** a fost proiectat pentru a asigura implementarea portabilă a sistemului de operare **UNIX**. Ca o consecinţă a acestui fapt programele scrise în limbajul **C** au o portabilitate foarte bună.

Multe din cele mai importante idei din **C** îşi au originea în limbajul **BCPL**, dezvoltat de Martin Richards de la Universitatea din Cambridge, în anul 1960. Din limbajul **BCPL** a derivat un nou limbaj, limbajul **B**, scris de Ken Thompson în 1970. Limbajului **B** i s-au adus o serie de îmbunătăţiri, apărând chiar şi o formă nouă, numită **NB** (New B). Din acest limbaj a derivat limbajul **C**.

Limbajul **C**, deşi este un limbaj de nivel înalt, păstrează şi concepte de nivel scăzut (registru, adresă, locaţie de memorie etc). Peste 90% din sursele primului sistem de operare **UNIX** au fost scrise utilizând limbajul **C**, pentru restul surselor utilizându-se limbajul de asamblare. S-a dorit astfel transformarea acestui sistem de operare într-unul universal, motiv pentru care sistemul a fost distribuit împreună cu codul sursă şi cu descrierea limbajului. Acest fapt a incitat o serie de programatori să-l dezvolte, să creeze noi module şi să-l implementeze pe alte maşini prin rescrierea modulelor elaborate în limbajul de asamblare. Astfel, limbajul **C** a devenit un limbaj de referinţă.

Principalele caracteristici ale limbajului **C** sunt:

- limbaj procedural de nivel înalt;
- posedă concepte de nivel scăzut, ceea ce permite exploatarea portabilă a caracteristicilor interne ale unei maşini;
- posedă funcţii de conversie a datelor foarte evoluate;
- permite definirea de noi tipuri de date de către utilizator;
- gestionarea elaborată a datelor de tip dinamic;
- posibilitatea definirii de noi funcţii;
- adresări indirecte ale datelor şi variabilelor (pointeri);
- recursivitate;
- set complet de funcţii matematice;
- posibilitatea de apel ale unor funcţii sistem ale sistemului de operare (Windows sau DOS);
- concizie deosebită a limbajului.



În anul 1978 a apărut lucrarea *"The C Programming Language"* scrisă de Brian W. Kernighan și Dennis M. Ritchie, iar în anul 1988 această lucrare a fost reeditată. Ediţia originală a servit timp de mulţi ani ca manual de referinţă pentru cei care doreau să înveţe limbajul **C**.

În anul 1983 în cadrul ANSI (American National Standard Institute) s-a creat un comitet de lucru (X3J11) care a realizat limbajul **ANSI C**. Acesta înlătură o serie de inconsistenţe şi scăpări ale versiunii originale. Versiunea finală a limbajului **C** a apărut în anul 1990 la care au fost aduse o serie de completări în anul 1995.

Limbajul **C** oferă o serie de avantaje: este unul dintre limbajele de uz general folosite intens la nivel mondial, permite programarea structurată şi asigură o bună portabilitate, permite operaţii pe biţi ceea ce face să fie foarte utilizat în programarea la nivel de sistem, nu în ultimul rând are o flexibilitate ridicată.

Limbajul **C** s-a impus în principal datorită existenţei unui standard care conţine toate facilităţile necesare unui limbaj pentru a putea fi utilizat într-o mare diversitate de aplicaţii, fără a fi nevoie de abateri sau extinderi faţă de standard. Limbajul **C** oferă posibilitatea ca un program să poată fi format din mai multe fişiere sursă, posibilitatea compilării independente ale acestora, precum şi posibilitatea referirii dintr-un fişier în altul. De asemenea, există un număr mare de funcţii uzuale care fac parte din standardul limbajului şi care asigură portabilitatea ridicată a programelor scrise în **C**.

Un fapt extrem de important, foarte apreciat de programatori, reprezintă posibilitatea controlului total asupra operaţiilor realizate de procesor şi asupra funcţiilor sistemului de operare gazdă, în mod asemănător limbajelor de asamblare. Din această cauză, majoritatea programelor de sistem, a utilitarelor şi a aplicaţiilor au fost scrise în ultimii ani în limbajul **C**.

Limbajul **C** oferă posibilitatea scrierii unor programe compacte. Reducerea dimensiunilor programelor s-a realizat prin reducerea numărului de cuvinte cheie, prin existenţa unui număr mare de operatori exprimaţi prin unul sau prin două caractere speciale precum şi prin posibilitatea de a combina mai mulţi operatori şi expresii într-o singură instrucţiune. Utilizarea adreselor de memorie (prin intermediul pointerilor) este asemănătoare limbajelor de asamblare şi permite operaţii imposibile altor limbaje. Programarea în limbajul **C** permite o mare diversitate de construcţii corecte din punct de vedere sintactic dar care pot genera o serie de erori greu de identificat la execuţie. Limbajul **C** utilizează elementele fundamentale ale controlului fluxului de execuţie al programului, cum ar fi: grupare de instrucţiuni; luare de decizii ("**if**"); bucle cu test de terminare la început ("**while**", "**for**") sau la sfârşit ("**do...while**"), selectare a unui caz dintr-o mulţime de cazuri posibile ("**switch**").

Orice funcţie poate fi apelată recursiv şi variabilele sale sunt tipic "automate" sau nou create la fiecare apelare. Modulele unui program **C** pot fi compilate separat, variabilele pot fi interne unei funcţii sau externe, recunoscute numai într-un singur fişier sursă, sau complet globale.

Limbajul **C** prezintă şi o serie de neajunsuri însă acestea nu reprezintă un dezavantaj aşa de important încât să împiedice acceptarea în practică a acestui limbaj. Faptul că permite scrierea de noi translatoare (compilatoare şi interpretoare) pentru tipuri noi de platforme, precum şi faptul că un cod scris în limbajul **C** este mult mai compact decât dacă ar fi scris în alte limbaje de programare au făcut ca limbajul **C** să fie preferat şi să fie acceptat de tot mai mulţi programatori. O consecinţă importantă este aceea că au fost implementate în **C** o serie de compilatoare, biblioteci şi interpretoare de nivel înalt. Sintaxa limbajului **C** a stat la baza altor limbaje create ulterior şi care sunt foarte populare astăzi: Java, JavaScript, C#, Visual C++, Borland C++ Builder etc.

1.1.4. Structura generală a unui program C

Un program scris în limbajul **C** are următoarea structură:

// linii de comentariu	
Directive preprocesor	← constau în includeri de fişiere antet şi definiţii
Declaraţii de variabile globale	
Declaraţii / definiţii de funcţii	← declaraţii / definiţii de funcţii utilizator
main()	← antetul funcţiei principale
{	← marchează începutul funcţiei principale
Declaraţii de variabile	← declaraţii de variabile locale
Funcţii	← apeluri de funcţii de bibliotecă sau funcţii utilizator
Instrucţiuni	← instrucţiuni ale limbajului C;
}	← marchează sfârşitul funcţiei principale
Definiţii de funcţii	← definiţii de funcţii utilizator

Limbajul **C** este compus din mai multe funcţii, spre deosebire de alte limbaje de programare care utilizează şi proceduri (de exemplu limbajul Pascal). Funcţiile sunt unităţi de program care efectuează anumite acţiuni, adică manipulează / modifică date. Deşi există variante diferite de compilatoare **C**, în urma standardizării biblioteca **C** este aceeaşi şi se numeşte **ISO C library**. Interfaţa de programare (**API**) a bibliotecii standard **C** este declarată într-un număr de fişiere antet.

O bibliotecă de funcţii grupează o serie de funcţii după acţiunile pe care le realizează acestea.

Funcţiile pot fi funcţii de bibliotecă sau funcţii definite de utilizator, fiecare funcţie având un nume, eventual o listă de argumente şi poate returna cel mult o valoare. Numele unei funcţii este însoţit întotdeauna de paranteze rotunde între care se pot trece argumentele funcţiei separate prin virgulă. Funcţia definită în program poate fi apelată printr-o instrucţiune de apel sau ca operand într-o expresie, precizând la apel: numele funcţiei și parametrii efectivi ai funcţiei (plasaţi între paranteze rotunde). Într-un program **C**, pe lângă funcţiile de bibliotecă ale limbajului, mai pot fi utilizate şi funcţii definite de programator.

Orice program scris în limbajul **C** are o cel mult o funcţie **main** – punctul de intrare în program – în cazul în care funcţia **main** lipseşte atunci este o funcţie de bibliotecă.

Forma generală a funcţiei **main** este:

```
tip main(lista_agumente)
{
    Corpul funcţiei main
}
```

Execuţia oricărui program **C** constă în execuţia secvenţială a instrucţiunilor din corpul funcţiei **main()**. În corpul funcţiei **main()** pot fi apelate celelalte funcţii definite în program (de bibliotecă sau utilizator).

Preprocesorul este un editor de texte neinteractiv fiind o componentă de limbaj independentă de compilator. Preprocesarea are loc înaintea compilării şi produce în textul sursă modificări din categoria: substituţie, compilare condiţionată sau includerea de fişiere sursă. Liniile care încep cu **#** sunt instrucţiuni adresate preprocesorului fiind independente de instrucţiunile limbajului **C**. Cele mai uzuale directive preprocesor sunt: **#include** şi **#define**.

Dacă în cadrul unui program utilizăm o funcţie din biblioteca de funcţii a limbajului **C**, atunci fişierul antet care conţine definiţia funcţiei trebuie inclus în program, înaintea utilizării funcţiei.

Directiva **#include** permite includerea de fişiere antet în cadrul programului sursă. Fişierele antet sunt fişiere text care conţin declaraţii, definiţii şi alte informaţii referitoare la funcţiile de bibliotecă ale limbajului **C**. Fişierele antet au extensia **.h** (header) şi sunt stocate într-un director ce poartă numele **INCLUDE**.

Funcţiile de bibliotecă sunt module obiect, legătura dintre acestea şi programul sursă realizându-se în etapa de link-editare. Pentru ca link-editorul să poată realiza legătura trebuie să primească informaţii referitoare la fişierele antet. Cea mai importantă informaţie se referă la locul în care se află fişierele antet. Dacă numele fişierului antet este cuprins între paranteze unghiulare **<>** fişierul este căutat în subdirectorul **INCLUDE** din locaţia mediului de programare. Dacă numele fişierului este cuprins între ghilimele **“ ”** acesta va fi căutat în directorul curent.

Directiva **#define** permite definirea constantelor simbolice.

1.1.5. Declaraţii de variabile

Limbajul **C** nu are variabile declarate implicit, motiv pentru care trebuie declarate toate variabilele care apar în program. Declararea variabilelor trebuie realizată înainte de utilizarea acestora. La declararea unei variabile i se alocă un spaţiu în memoria RAM, aceasta fiind formată din mai multe locaţii de memorie. Fiecare locaţie de memorie are o adresă distinctă, dimensiunea unei locaţii fiind de 8 biţi, adică un octet.

Pentru ca datele să poată fi interpretate corect de către calculator, trebuie să precizăm modul în care acestea sunt codificate în formă binară (succesiuni de cifre **0** şi **1**).

Datele sunt caracterizate prin următoarele elemente:

A. Numele datei (**identificator**): se utilizează pentru a identifica data şi pentru a o distinge de celelalte date;

La alegerea numelui unei date trebuie să se ţină cont de următoarele aspecte:

- numele trebuie să fie cât mai scurt;
- numele trebuie să înceapă cu o literă sau cu un caracter de subliniere _;
- nu este admisă prezenţa spaţiului în numele unei variabile;
- limbajul **C** face distincţia între litere mari şi mici (este case-sensitive).

Limbajul **C** are 32 de cuvinte cheie care nu pot fi utilizate ca şi nume de date: ***auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while***.

B. Valoarea: reprezintă conţinutul datei. Datele se deosebesc în: **date variabile** (numite pe scurt *variabile*) a căror valoare se poate schimba pe parcursul rulării programului şi **date constante** (numite pe scurt *constante*) a căror valoare rămâne neschimbată pe parcursul rulării programului;

C. Tipul: specifică modul de reprezentare a datei în memoria calculatorului şi defineşte precizia asigurată de această reprezentare.

După tip, datele se clasifică în:

- date numerice:
  - întregi
    - cu semn;
    - fără semn;
  - reale:
    - în simplă precizie
    - în dublă precizie;
    - în dublă precizie extinsă;
- date de tip caracter (alfanumerice).

Tipurile de date predefinite ale limbajului **C** sunt specificate cu ajutorul unor cuvinte cheie, astfel:

char – defineşte datele de tip caracter (litere, cifre sau simboluri corespunzătoare codurilor ASCII);
int – defineşte datele de tip întreg;
float – defineşte date de tip real, în simplă precizie;
double – defineşte date de tip real, în dublă precizie;
short – se utilizează pentru a specifica că data este de tip scurt;
long – se utilizează pentru a specifica că data este de tip lung.

Aceste cuvinte cheie pot fi combinate pentru a obţine alte tipuri de date (adică predefinite).

D. Domeniul de vizibilitate (scope): reprezintă mulţimea instrucţiunilor (linii de cod) în care poate fi utilizată data. Din acest punct de vedere datele pot fi: **locale**, pot fi declarate în funcţii în blocuri de instrucţiuni sau ca parametri formali, respectiv **globale**, sunt declarate în afara oricăror funcţii.

În tabelul următor sunt reprezentate toate tipurile de date ale limbajului **C**:

Specificarea tipului	Dimensiune [octeţi]	Interval de valori
signed char	1	-128 ~ 127
unsigned char	1	0 ~ 255
int, short	2	-32768 ~ 32767
unsigned	2	0 ~ 65535
long (int)	4	-2147483648 ~ 2147483647
unsigned long (int)	4	0 ~ 4294967295
float	4	$\pm[3,4*10^{-38}$ ~\` $3,4*10^{38}]$
double	8	$\pm[1,7*10^{-308}$ ~ $1,7*10^{308}]$
long double	10	$\pm[3,4*10^{-4932}$ ~ $1,1*10^{4932}]$

Declararea variabilelor simple se realizează pentru stabilirea legăturii dintre numele variabilei şi tipul valorilor pe care le poate avea aceasta. Forma generală a unei instrucţiuni de declarare de variabile simple este:

tip listă_nume_variabile **;**

unde: **tip** reprezintă unul din tipurile de date ale limbajului **C** sau un tip de date definit de programator, respectiv **listă_nume_variabile** conţine numele variabilei sau a variabilelor declarate separate prin caracterul **" , "**.

În cadrul instrucţiunii de declarare a variabilei se poate face şi iniţializarea acesteia cu o anumită valoare, astfel că având toate elementele precizate (nume, tip, valoare) variabila este <u>definită</u>.

Declararea tablourilor

Tabloul reprezintă o mulţime de date de acelaşi tip, grupate sub un nume comun, referirea la elementele tabloului realizându-se prin intermediul indicilor. Un tablou se caracterizează prin **nume, tip şi dimensiune**. Tablourile pot fi cu o dimensiune (vectori), cu două dimensiuni (matrice) sau cu mai multe dimensiuni. Numărul de dimensiuni este dat de numărul de indici necesari pentru definirea unui element al tabloului.

Elementele tablourilor se mai numesc şi variabile cu indici, valorile indicilor începând de la **0**.

Declaraţia unui tablou are următoarea formă:

tip nume_tablou **[lim_1][lim_2]…[lim_n] ;**

unde: **tip** reprezintă unul din tipurile de date ale limbajului **C** sau un tip de date definit de programator;
nume_tablou reprezintă identificatorul tabloului;
lim_1, lim_2, … , lim_n reprezintă limitele superioare ale celor **n** indici ai tabloului;

Declararea unui tablou are ca efect rezervarea unei zone de memorie în care sunt păstrate, unul după altul, elementele tabloului.

1.1.6. Constante

Pe lângă variabile, în programe mai pot interveni şi constante, adică date care nu-şi modifică valoarea pe parcursul rulării programului, sau de la o rulare la alta. Constantele au un tip şi o valoare şi pot fi **literale** sau **simbolice**.

Constantele literale sunt de patru tipuri: *întregi*, *flotante*, *caracter* sau *şiruri de caractere*.

Constantele întregi corespund datelor de tip întreg şi pot fi: **zecimale** adică formate din cifre de la 0 la 9 (corespund numerelor scrise în baza de numeraţie 10); **octale** acestea sunt precedate de un zero nesemnificativ şi sunt urmate de cifre octale de la 0 la 7 (baza de numeraţie 8); **hexazecimale**, acestea sunt precedate de **0x** sau **0X** urmate de o serie de cifre hexazecimale, adică cifre de la 0 la 9 şi litere de la A la F (baza de numeraţie 16).

Tipul constantelor întregi este dat de valoarea acestora, astfel:
- o constantă zecimală a cărei valoare este mai mică decât 32767 (= valoarea celui mai mare întreg cu semn care poate fi reprezentat pe doi octeţi) este considerată de tip **int**, respectiv cea a cărui valoare este mai mare decât 32767 este considerată de tip **long**;
- constantele întregi octale sau hexazecimale a căror valoare este mai mică decât 65535 (= valoarea celui mai mare întreg fără semn care poate fi reprezentat pe 2 octeţi) se consideră de tipul **unsigned**, iar dacă valoarea acestora depăşeşte 65535 atunci constantele sunt de tip **unsigned long**.

Dacă se doreşte precizarea în mod explicit a tipului unei constante întregi, atunci în urma valorii se vor scrie una din literele **l** sau **L** pentru tipul **long**, respectiv **u** sau **U** pentru tipul **unsigned**.

Constante flotante: reprezintă numere raţionale şi se pot scrie în două moduri:

A. **parte întreagă . parte fracţionară**, de exemplu: 3.256, .34, 2.;

Observaţii: Dacă partea întreagă lipseşte se poate evita scrierea cifrei 0 dinaintea punctului, însă dacă partea fracţionară lipseşte, punctul zecimal trebuie pus în mod obligatoriu.

B. **parte întreagă . parte fracţionară e (E) ± ct. zecimală** (format exponenţial), de exemplu: 1.23e2 =$1.23*10^2$ = 123

Observaţii: În cazul în care constanta flotantă se scrie cu exponent, dacă partea fracţionară lipseşte nu este obligatorie utilizarea punctului zecimal (**.**).

Constantele flotante aparţin tipului **double**, dar dacă la sfârşitul valorii se scrie litera **f** sau **F**, constanta va aparţine tipului **float**, iar dacă se scrie litera **l** sau **L** aceasta va aparţine tipului **long double**.

Constante caracter: se formează prin includerea între caractere apostrof a unor caractere imprimabile (de exemplu: **'x'**). Constantele caracter au valoarea egală cu valoarea codului ASCII al caracterului pe care îl reprezintă.
Există două feluri de caractere: imprimabile [32-126] şi neimprimabile (secvenţe **ESCAPE**).[0-31,127].

Secvenţele **ESCAPE** sunt precedate de caracterul **backslash** " **** " numit şi caracter **ESCAPE**.

Constante şiruri de caractere: se reprezintă între ghilimele (de exemplu **" xcxvx "**). Sunt păstrate în memorie caracter după caracter, într-o zonă contiguă, prin codurile lor ASCII.

La sfârşitul şirului este inserat caracterul **NULL**.('**\0'**) care marchează sfârşitul şirului.

Observaţie: În şirurile de caractere pot fi incluse şi secvenţe ESCAPE.

Constante simbolice: sunt date caracterizate prin nume, tip şi valoare constantă. Acestea se definesc la începutul programului utilizând cuvântul **const** sau prin intermediul directivei preprocesor **#define**.

1.2. Funcţii de intrare – ieşire

Limbajul **C** nu are instrucţiuni specifice pentru introducerea şi extragerea datelor, astfel că introducerea şi extragerea datelor din program se realizează cu ajutorul unor funcţii din biblioteca standard a limbajului. În limbajul **C** sunt funcţii pentru introducerea datelor (intrare): **getch()**, **getche()**, **gets()**, **scanf()** şi funcţii pentru extragerea datelor (ieşire): **putch()**, **puts()** şi **printf()**. Pe lângă aceste funcţii mai sunt şi două macrouri **getchar()** şi **putchar()**.

Definiţiile acestor funcţii se găsesc în biblioteca de funcţii, deci pentru utilizarea lor în program trebuie să se includă mai întâi fişierele antet corespunzătoare.

1.2.1. Funcţii de intrare-ieşire pentru caractere

Funcțiile pentru citirea unui caracter de la tastatură sunt următoarele: **getch()** și **getche()**, acestea au prototipul în fişierul antet **conio.h**.

Funcția **getch()** – citeşte un caracter de la tastatură (**get ch**aracter). Caracterul citit nu este afişat pe ecran, adică citirea nu se face cu ecou.

Funcția **getche()** – citeşte un caracter de la tastatură, însă citirea se face cu ecou pe ecran (**get ch**aracter with **e**cho).

Cele două funcţii se utilizează pentru citirea atât a caracterelor ASCII cât şi a caracterelor corespunzătoare tastelor funcţionale sau combinaţiilor de taste. Pentru preluarea combinaţiilor de taste, funcţiile trebuie apelate de două ori în program, prima dată se returnează valoarea 0, iar a doua oară se returnează codul asociat combinaţiei de taste.

Funcţiile nu au argumente dar returnează o valoare care reprezintă codul ASCII al caracterului citit.

Funcţiile sunt apelate prin intermediul unei instrucţiuni de apel, dar pot fi utilizate ca şi operanzi în expresii.

Funcţia **getch()** se mai utilizează în mod curent la sfârşitul programului sau în interiorul acestuia atunci când se doreşte vizualizarea rezultatelor pe ecran. Prin inserarea unei instrucţiuni de apel a funcţiei **getch()** programul aşteaptă acţionarea unei taste, acest fapt are ca efect păstrarea datelor afişate pe ecran (până la apăsarea unei taste execuţia programului se opreşte).

Funcția pentru afişarea pe ecran a unui caracter este **putch()** (**put ch**aracter), aceasta are prototipul în fişierul antet **conio.h**. Modalitatea de apelare este următoarea:

putch(expresie);

Funcţia are un argument care poate fi o expresie, o constantă întreagă sau o constantă caracter. Dacă argumentul este o expresie, la apelarea funcţiei se realizează mai întâi evaluarea expresiei, rezultatul trebuie să fie de tip **int** şi reprezintă codul ASCII al caracterului pe care dorim să-l afişăm.

Funcţia returnează o valoarea care reprezintă codul ASCII al caracterului afişat, deci funcţia poate fi utilizată ca şi operand în expresii.

1.2.2. Macrourile getchar() şi putchar()

Cele două macrouri folosite în limbajul **C** sunt definite în fişierul antet **stdio.h**, apelul lor fiind similar cu al funcţiilor.

getchar() – citeşte de la tastatură un caracter, deosebirea faţă de **getch()** sau **getche()** constă în faptul că respectivul caracter nu este citit direct de la tastatură ci dintr-o zonă tampon în care sunt păstrate toate caracterele scrise de utilizator până la apăsarea tastei **ENTER**. Instrucțiunea de apel este următoarea:

getchar();

Avantajul constă în faptul că se pot corecta caracterele tastate înainte de apăsarea tastei **ENTER**.

Macroul **getchar()** poate citi numai caractere ASCII, nu are argumente şi returnează codul ASCII al caracterului citit.

Dacă utilizatorul tastează după introducerea secvenţei de caractere combinaţia **Ctrl + Z**, macroul **getchar()** va returna valoarea constantei **EOF** (**E**nd-**O**f-**F**ile) definită în **stdio.h**.

putchar() – afişează un caracter ASCII pe ecran, apelul fiind sub forma:

putchar(expresie);

Argumentul macro-ului (**expresie**) reprezintă valoarea codului ASCII al caracterului care se doreşte a fi afişat pe ecran şi returnează valoarea codului ASCII al caracterului afişat sau -1 la eroare.

1.2.3. Funcţiile de intrare – ieşire gets() şi puts()

Pentru citirea sau afișarea șirurilor de caractere se folosesc următoarele funcții:

gets() (**get s**tring) se utilizează pentru citirea de la tastatură a unui şir de caractere, în mod asemănător cu **getchar()**. Citirea începe după apăsarea tastei ENTER, caracterele introduse fiind stocate într-o zonă tampon.

Funcţia are prototipul în fişerul antet **stdio.h**.

Funcţia are un argument care reprezintă adresa de început a zonei de memorie în care va fi păstrat şirul de caractere şi returnează aceeaşi adresă de început a şirului.

Deoarece şirurile se păstrează în memorie sub forma unor tablouri unidimensionale de tip **char**, parametrul funcţiei va fi numele tabloului. Apelul se face sub următoarea formă:

gets(sir_de_caractere);

puts() (**put s**tring) se utilizează pentru afişarea pe ecran a unui şir de caractere. Are prototipul în fişierul antet **stdio.h**.

Apelul se realizează sub următoarea formă:

puts(argument);

Acest **argument** trebuie să fie adresa de început a zonei de memorie în care se păstrează şirul. În mod asemănător funcţiei **gets()** putem utiliza numele şirului ca şi argument. Funcţia returnează codul ASCII al ultimului caracter din şir sau -1 la eroare şi poate fi utilizată ca şi operand în expresii.

Observaţie: Prin apăsarea tastei **ENTER** la sfârşitul şirului se adaugă caracterul **NULL** ('**\0**'). Din această cauză la declararea şirurilor de caractere va trebui atribuită o dimensiune cu o unitate mai mare decât lungimea şirului. Funcţia interpretează acest caracter ca şi caracter "new line" ceea ce determină saltul cursorului la începutul rândului următor.

1.2.4. Funcţia de ieşire cu format printf( )

Funcţia **printf( )** (**print** with **f**ormat) se utilizează pentru afişarea datelor pe ecran, indiferent de tipul acestora, specificându-se formatul în care vor fi afişate datele.

Prototipul funcţiei se află în fişierul antet **stdio.h** iar apelul ei se face printr-o construcţie de forma:

printf(sir_control, lista_de_argumente);

unde:

sir_control reprezintă o succesiune de caractere, scrise între ghilimele (" "), care defineşte textele şi formatul datelor care se afişează şi poate conţine:

- **succesiuni de caractere** care se vor afişa ca atare pe ecran;
- **secvenţe ESCAPE** cu ajutorul cărora pot fi realizate câteva acţiuni;
- **specificatori de format** care definesc conversia valorilor argumentelor din listă, din formatul intern în formatul extern (cel de afişare pe ecran);

lista_de_argumente – conţine o listă de argumente, separate prin virgulă, prin care se indică datele ce vor fi afişate. Este opţională şi poate să conţină variabile, constante, expresii sau apeluri de funcţii. Fiecărui argument trebuie să-i corespundă câte un specificator de format.

Specificatorii de format încep cu caracterul **%** şi au forma generală:

%fanion dimensiune . precizie modificator tip

Semnificaţia componentelor este următoarea:

% : reprezintă caracterul prin care se precizează faptul că este vorba de un specificator de format, este obligatoriu;

tip : este o componentă obligatorie din specificatorul de format şi reprezintă o literă care specifică tipul conversiei aplicate datelor care se afişează, astfel: **c** – afişarea unui caracter; **s** – afişarea unui şir de caractere; **i, d** – afişarea unui număr întreg (în baza zece) cu semn; **u** – afişarea unui număr întreg (în baza zece) fără semn; **f** – afişarea unui număr real cu semn - notaţie zecimală (implicit sunt 6 cifre zecimale); **e**, **E** – afişarea unui număr real cu semn (notaţie exponenţială); **g**, **G** – afişarea unui număr real utilizând cea mai scurtă reprezentare între **f** şi **e**; **x**, **X** – afişarea unui număr hexazecimal întreg fără semn; **o** – afişarea unui număr octal întreg fără semn; **p** – afişarea unei adrese (a unui pointer);

fanion : este o componentă opţională şi poate fi unul din caracterele: **+** (*plus*): se afişează semnul datei (plus sau minus); **-** (*minus*): data este aliniată la stânga în câmpul de scriere, implicit alinierea se face la dreapta;   (*spaţiu*): valorile pozitive sunt scrise fără semn lăsându-se un spaţiu în faţa valorii afişate, iar cele negative se scriu cu semn (semnul minus) fără a se lăsa spaţiu; **#** (*diez*): determină folosirea formei alternative de scriere, pentru datele octale şi hexazecimale, adică se scrie un zero nesemnificativ în cazul datelor octale, respectiv **0x** sau **0X** în cazul datelor hexazecimale.

dimensiune : este o componentă opţională şi constă dintr-un număr întreg care defineşte dimensiunea minimă a câmpului în care va fi afişată data. Dacă sunt necesare mai multe poziţii, data va fi afişată pe câte poziţii este necesar, iar dacă data are mai puţine caractere aceasta va fi implicit aliniată la dreapta în câmpul de afişare. Spaţiile rămase libere se completează cu caractere nesemnificative (spaţiu). Dacă se plasează un 0 (zero) nesemnificativ înaintea cifrei ce reprezintă **dimensiune**, spaţiile rămase libere vor fi completate cu cifra 0. Dacă în locul cifrei ce reprezintă **dimensiune** se plasează caracterul *****, atunci valoarea **dimensiune** este precizată la execuţie, printr-un element din lista de argumente.

. precizie : este o componentă opţională (număr întreg) care indică precizia de scriere a datei, astfel pentru:

- **%e**, **%E**, **%f** specifică numărul de zecimale;
- **%g**, **%G** specifică numărul de cifre semnificative;
- **%d**, **%i**, **%u** specifică numărul minim de cifre pe care va fi reprezentată data de tip întreg. Dacă data este mai mică se pun cifre 0 (zero) iar dacă data este mai mare se foloseşte un câmp corespunzător;
- **%s** specifică numărul maxim de caractere care se afişează;
- **%c** nu are efect.

Dacă în locul numărului întreg se plasează *****, valoarea acestuia se va preciza la execuţie prin intermediul unui element din lista de argumente.

modificator : este o componentă opţională, poate fi una din literele **h, l, L**. Împreună cu componenta **tip** defineşte tipul conversiei aplicat datelor, astfel:

h + **d**, **i**, **o**, **u**, **x**, **X** -> date de tip **short int**;
l + **d**, **i**, **o**, **u**, **x**, **X** -> date de tip **long int**;
l + **e**, **E**, **f**, **g**, **G** -> date de tip **double**.
L + **e**, **E**, **f**, **g**, **G** -> date de tip **long double**.

Observaţii:

1. *Dacă în specificatorul de format după caracterul* **%** *se scrie un caracter care nu este dintre cele utilizate şi prezentate anterior, atunci caracterul* **%** *este ignorat, deci nu va fi luat în considerare ca un specificator de format iar caracterele care urmează după* **%** *vor fi afişate pe ecran aşa cum apar ele.*
2. *Între specificatorii de format din* **sir_control** *şi argumentele din* **lista_de_argumente** *trebuie să existe o concordanţă în ceea ce priveşte numărul, ordinea şi tipul acestora. În caz contrar poate apărea una din situaţiile următoare: se face o conversie a datelor la ieşire, se afişează datele incomplet sau se semnalează o eroare.*

1.2.5. Funcţia de intrare scanf( )

Funcţia **scanf()** se utilizează pentru citirea datelor de la tastatură, indiferent de tipul acestora. Funcţia permite specificarea formatului în care vor fi citite datele (**scan** with **f**ormat), are prototipul în fişierul antet **stdio.h** iar apelul ei se face printr-o construcţie de forma:

scanf(sir_control, lista_de_adrese);

unde:

sir_control: este un şir de caractere cuprinse între ghilimele care defineşte formatul datelor de intrare (o succesiune de descriptori de format);

lista_de_adrese: reprezintă adresele de memorie în care vor fi păstrate valorile citite, separate prin virgulă (**,**).

Citirea datelor cu ajutorul funcţiei **scanf** este asemănătoare cu cea de la **getchar**, adică datele se citesc efectiv dintr-o zonă tampon numai după acţionarea tastei **ENTER**.

Adresa unei zone de memorie se specifică utilizând **operatorul adresă** (**&**) plasat înaintea numelui variabilei. În cazul şirurilor de caractere nu este necesară utilizarea **operatorului adresă** deoarece numelui şirului îi este atribuită valoarea adresei de început a zonei de memorie în care aceste este memorat. În cazul variabilelor cu indici (elementele unui tablou) este necesară utilizarea **operatorului adresă**. Pot fi citite mai multe variabile într-o singură secvenţă, separatorii utilizaţi fiind blank (spaţiu), tab-ul ('\t'), new line ('\n') etc.

Observaţie:Trebuie să existe o concordanţă între descriptorii de format din **sir_control** *şi datele ale căror adrese sunt precizate în* **lista_de_adrese** *în ceea ce priveşte numărul, tipul şi ordinea acestora.*

Forma generală a descriptorilor de format este:

% dimensiune modificator tip

unde: **dimensiune** : reprezintă dimensiunea maximă a câmpului de intrare, este o componentă opţională;
modificator : este o componentă opţională şi are aceeaşi semnificaţie ca şi în cazul funcţiei **printf**;
tip : este o literă care indică conversia aplicată datei la intrare (la citire) fiind o componentă obligatorie;

Citirea se realizează dintr-un câmp de intrare. Un câmp de intrare începe de la primul caracter introdus (exclus caracterele albe) şi se termină în una din situaţiile:

- se tastează un caracter alb (spaţiu, tab orizontal, linie nouă, tab vertical);
- se atinge dimensiunea impusă prin componenta **dimensiune** a descriptorului de format;
- se tastează un caracter al cărui tip nu corespunde tipului precizat în descriptorul de format.

Funcţia **scanf** returnează o valoare întreagă care reprezintă numărul de caractere citite corect.

*Observaţie**:** În cazul în care prin descriptorul de format se specifică citirea unui singur caracter, se citeşte caracterul curent indiferent de faptul că acest caracter este un caracter alb sau nu.*

1.3. Expresii, operanzi, operatori

1.3.1. Introducere

O **expresie** este o secvenţă corect formată din **operanzi** (date) şi **operatori**. Ca urmare a evaluării unei expresii, aceasta va avea o **valoare** şi un **tip**. Acestea depind de valorile şi tipurile operanzilor, dar şi de ordinea de efectuare a operaţiilor.

Există trei tipuri de expresii: **matematice** (au ca rezultat date numerice de tip întreg sau real), **caracter** (au ca rezultat un şir de caractere), respectiv **logice** (au ca rezultat noţiunile de **adevărat** = **1** sau **fals** = **0**).

Operanzii sunt date asupra cărora se efectuează **operaţii** şi pot fi:

- constante;
- constante simbolice;
- nume de variabile simple;
- nume de tablouri;
- nume de structuri;
- nume de funcţii;
- numele unui tip;
- elemente de tablouri;
- elementele unei structuri;
- apeluri de funcţii;
- expresii inclusă în paranteze rotunde, numite şi subexpresii.

În procesul de evaluare a unei expresii trebuie să se ţină cont de:

- prioritatea operatorilor: arată cum se evaluează expresiile care conţin doi sau mai mulţi operatori aplicaţi unui acelaşi operand sau unor operanzi diferiţi, în situaţia când nu există paranteze care să impună o anumită ordine a operaţiilor;

- asociativitatea operatorilor: arată cum se evaluează o expresie în care apar de mai multe ori operatori cu aceeaşi prioritate;

- regula conversiilor implicite: acţionează atunci când un operator binar se aplică asupra a doi operanzi de tipuri diferite şi constă în conversia operandului de tip „inferior" spre tipul operandului de tip „superior", rezultatul fiind de tipul „superior".

Aşadar:

- dacă unul dintre operanzi este de tip **long double**, rezultatul expresiei va fi de tip **long double**;
- dacă unul dintre operanzi este de tip **double**, rezultatul expresiei va fi de tip **double**;
- dacă unul dintre operanzi este de tip **float**, rezultatul expresiei va fi de tip **float**;
- dacă unul dintre operanzi este de tip **unsigned long**, rezultatul expresiei va fi de tip **unsigned long**;
- dacă unul dintre operanzi este de tip **long**, rezultatul expresiei va fi de tip **long**;
- dacă unul dintre operanzi este de tip **unsigned**, rezultatul expresiei va fi de tip **unsigned**.

Prin aplicarea regulii conversiilor implicite, tipul variabilelor nu se modifică, astfel că o variabilă de un anumit tip va rămâne de tipul respectiv, doar valoarea acesteia se converteşte către tipul „superior".

În tabelul următor, operatorii sunt ordonaţi în ordinea priorităţii (prima categorie are prioritate maximă).

<table>
<tr><th>Categorie</th><th>Operatori</th><th>Semnificaţie</th><th>Asociativitatea</th></tr>
<tr><td>1. Prioritate maximă</td><td>( )
[ ]
. sau -></td><td>Apel de funcţie
Element de tablou
Selecţie membru al unei structuri</td><td>de la stânga la dreapta</td></tr>
<tr><td>2. Operatori unari</td><td>!
~
+ -
++ --
&
*
sizeof
(tip)</td><td>Negare logică
Negare bit cu bit (complement faţă de 1)
Plus şi minus unari
Incrementare / decrementare (pre şi post)
Operatorul adresă
Operatorul de indirectare
Dimensiune operand (octeţi)
Conversie explicită de tip</td><td>de la dreapta la stânga</td></tr>
<tr><td>3. Operatori de multiplicare</td><td>*
/
%</td><td>Înmulţire
Împărţire
Împărţire modulo împărţitor</td><td>de la stânga la dreapta</td></tr>
<tr><td>4. Operatori aditivi</td><td>+ -</td><td>Plus şi minus binari</td><td>de la stânga la dreapta</td></tr>
<tr><td>5. Deplasări pe biţi</td><td><< >></td><td>Deplasări bit cu bit</td><td>de la stânga la dreapta</td></tr>
<tr><td>6. Operatori de relaţie</td><td>< <=
> >=</td><td>Mai mic, mai mic sau egal, mai mare, mai mare sau egal</td><td>de la stânga la dreapta</td></tr>
<tr><td>7. Operatori de egalitate</td><td>== !=</td><td>Egal, diferit</td><td>de la stânga la dreapta</td></tr>
<tr><td>8. Operatori logici la nivel de bit</td><td>&
^
|</td><td>ŞI logic bit cu bit
SAU EXCLUSIV bit cu bit
SAU logic bit cu bit</td><td>de la stânga la dreapta</td></tr>
<tr><td>9. Operatori logici</td><td>&&
||</td><td>ŞI logic
SAU logic</td><td>de la stânga la dreapta</td></tr>
<tr><td>10. Operatorul condiţional ternar</td><td>?:</td><td></td><td>de la dreapta la stânga</td></tr>
<tr><td>11. Operatori de atribuire</td><td>=
*= /= %=
+= -=
&= ^= |=
<<= >>=</td><td>Atribuire simplă
Atribuire cu produs, cât, rest
Atribuire cu sumă, diferenţă
Atribuire cu şi, sau exclusiv, sau
Atribuire cu deplasare la stânga, dreapta</td><td>de la dreapta la stânga</td></tr>
<tr><td>12. Virgulă</td><td>,</td><td>Evaluare expr1, expr2
Valoarea expresiei este expr2.</td><td>de la stânga la dreapta</td></tr>
</table>

1.3.2. Operatori aritmetici

Pot fi **unari** (adică se aplică unui singur operand) sau **binari** (adică se aplică la doi operanzi).

Operatorii unari sunt: **+** și **–**, aceștia se utilizează pentru indicarea semnului unui operand. Operatorul unar **+** nu are niciun efect. Operatorul unar **–** are ca efect negativarea valorii operandului pe care îl precede. Cei doi operatori au prioritate maximă şi se asociază de la dreapta la stânga.

Operatorii binari sunt de două feluri operatori **aditivi**: **adunarea** (**+**) și **scăderea** (**–**) respectiv operatori **multiplicativi**: **înmulţire** (*****), **împărţire** (**/**) și **împărţire modulo împărţitor** (**%**);

Operatorii multiplicativi au prioritate mai mică decât cei unari şi se asociază de la stânga la dreapta. În cazul operatorului împărțire **/** dacă cei doi operanzi sunt numere întregi, rezultatul împărțirii va fi tot un număr întreg.

Operatorii aditivi au prioritate mai mică decât cei multiplicativi şi se evaluează de la stânga la dreapta.

Operatorul modulo **%** se aplică numai operanzilor de tip întreg, rezultatul reprezintă restul împărţirii întregi între primul şi cel de-al doilea operand.

Observaţie: Limbajul **C** nu are operator de ridicare la putere, pentru efectuarea acestei operaţii se utilizează funcţia **pow** din biblioteca matematică, cu prototipul în fişierul antet **math.h**. Apelul acesteia se realizează astfel:

pow(baza, exponent);

Funcţia returnează o valoare de tip **double**.

1.3.3. Operatori relaţionali

În această categorie intră:

- operatori de relaţie: **<** mai mic, **<=** mai mic sau egal, **>** mai mare, **>=** mai mare sau egal;
- operatori de egalitate: **==** egal cu, **!=** diferit de.

În limbajul **C** nu există valori logice, astfel că prin compararea a două expresii cu ajutorul operatorilor relaţionali valoarea **fals** se reprezintă prin valoarea **0**, iar valoarea **adevărat** se reprezintă printr-o valoare diferită de zero (**≠0**).

Prioritatea operatorilor relaţionali este mai mică decât cea a operatorilor aditivi şi mai mare decât cea a operatorilor de egalitate. Asociativitatea acestor categorii de operatori este de la stânga la dreapta.

1.3.4. Operatori logici

Operatorii logici sunt: **negaţie logică** (**!**) - operator unar; **ŞI logic** (**&&**) - operator binar; **SAU logic** (**||**) - operator binar. Se utilizează pentru crearea condiţiilor obţinute prin combinarea a două sau mai multe expresii relaţionale. Deoarece în limbajul **C** nu există tipul **boolean**, operatorii logici admit operanzi de orice tip simplu pe care îi compară cu **0**. Dacă valoarea expresiei este **0** (nulă) aceasta este considerată **falsă**, iar dacă valoarea expresiei este nenulă, expresia se consideră **adevărată**.

La evaluarea unei expresii logice, ordinea de prioritate a operatorilor este: **!**, **<**, **<=**, **>**, **>=**, **==**, **!=**, **&&**, **||**.

În tabelul următor este ilustrat modul de acţiune al operatorilor logici.

a	b	!a	a&&b	a||b
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Conform tabelului, se pot formula următoarele enunţuri:

- dacă într-o expresie de forma **a && b** unul din operanzi are valoare nulă, întreaga expresie are valoare nulă;
- dacă într-o expresie de forma **a || b** unul din operanzi are valoare nenulă, întreaga expresie are valoare nenulă.

Operatorii logici binari (**&&** şi **||**) se evaluează de la stânga la dreapta.

1.3.5. Operatorul condiţional ternar

Este singurul operator ternar (are 3 operanzi) şi se utilizează pentru construirea unor expresii condiţionale (a căror valoare depinde de valoarea unei condiţii). Forma generală a unei expresii condiţionale este:

(expresie_1)?(expresie_2):(expresie_3);

unde: **expresie_1** – este în general o expresie logică.

Se determină valoarea expresiei **expresie_1**, dacă este diferită de zero (**adevărată**) atunci valoarea şi tipul întregii expresii condiţionale coincide cu valoarea şi tipul expresiei **expresie_2**, iar dacă **expresie_1** este nulă (**falsă**) valoarea şi tipul întregii expresii condiţionale coincide cu valoarea şi tipul expresiei **expresie_3**.

Din punct de vedere al priorităţii, operatorul condițional ternar se găseşte deasupra operatorului de atribuire și se asociază de la dreapta la stânga.

1.3.6. Operatori de incrementare (++) şi decrementare (--)

Sunt operatori unari, adică se aplică unui singur operand. Fac parte din clasa a doua de prioritate şi se asociază de la dreapta la stânga. Operatorul de incrementare (**++**) are ca efect creşterea cu **o unitate** a valorii operandului căruia i se aplică. Operatorul de decrementare (**--**) are ca efect scăderea cu **o unitate** a valorii operandului căruia i se aplică.

Pot fi utilizaţi ca **prefix** sau ca **sufix**, adică pot fi utilizaţi în faţa variabilei sau după aceasta.

Prefix: ++x, --x. În acest caz, mai întâi se aplică operaţia de incrementare sau decrementare a valorii variabilei **x**, valoarea modificată a acesteia fiind utilizată în continuare în calculele din expresia în care apare.

Sufix: x++, x--. În acest caz, mai întâi se fac calculele din expresia în care apare, cu valoarea iniţială a lui **x**, iar la sfârşit se aplică operaţia de incrementare sau decrementare a valorii variabilei **x**.

1.3.7. Operatori pe biţi

Se aplică operanzilor de tip întreg şi se execută bit cu bit.

Operatorii pe biţi sunt:

~ complement faţă de **1:** este un operator unar și se asociază de la dreapta la stânga. Se aplică asupra operandului la nivel de bit, inversând valorile biţilor acestuia, adică **0** devine **1** şi **1** devine **0**.

<< deplasare la stânga**:** este un operator binar și se asociază de la stânga la dreapta. Cu ajutorul operatorului se realizează deplasarea biţilor primului operand la stânga cu un număr de poziţii binare date de valoarea operandului al doilea. Operaţia este echivalentă cu înmulţirea primului operand cu **2** la puterea dată de valoarea celui de al doilea operand. Prin deplasarea la stânga a biţilor primului operand biţii care ies în partea stângă se pierd, iar poziţiile binare eliberate la dreapta sunt completate cu **0**.

>> deplasare la dreapta**:** este tot un operator binar și are aceeaşi prioritate cu operatorul **<<**. Realizează deplasarea la dreapta a biţilor primului operand cu un număr de poziţii date de valoarea operandului al doilea. Prin deplasarea la dreapta biţii care ies în dreapta se pierd, iar poziţiile binare eliberate la stânga se completează cu 0. Rezultatul este echivalent cu împărţirea primului operand cu **2** la puterea dată de valoarea celui de al doilea operand.

& (AND - **ŞI**): este un operator binar și se aplică operanzilor de tip întreg. Se aplică bit cu bit între biţii corespondenţi ai celor doi operanzi după următoarea regulă:

1	&	**1**	=	**1**
1	&	**0**	=	**0**
0	&	**1**	=	**0**
0	&	**0**	=	**0**

^ (XOR - **SAU EXCLUSIV**): este un operator binar şi se aplică bit cu bit între biţii corespondenţi ai celor doi operanzi după următoarea regulă:

1	^	**1**	=	**0**
1	^	**0**	=	**1**
0	^	**1**	=	**1**
0	^	**0**	=	**0**

| (OR - **SAU**): este un operator binar, se asociază de la stânga la dreapta şi se aplică operanzilor de tip întreg, bit cu bit între biţii corespondenţi, după următoarea regulă:

1	|	**1**	=	**1**
1	|	**0**	=	**1**
0	|	**1**	=	**1**
0	|	**0**	=	**0**

1.3.8. Operatori de atribuire

În forma cea mai simplă, operatorii de atribuire, sunt notaţi cu semnul egal “**=**” (atribuire simplă) şi se utilizează în expresii de forma:

v = (expresie)

unde: **v** poate fi o variabilă simplă, un element de tablou sau un element de structură.

expresie conţine operanzi şi operatori sau poate fi un apel de funcţie care returnează o valoare.

Deoarece operatorul de atribuire are prioritate mică (penultima clasă de prioritate) este posibil ca parantezele rotunde între care este încadrată expresia **expresie** să nu fie necesare.

Rezultatul atribuirii are valoarea expresiei **expresie** şi tipul variabilei **v**. În cazul în care tipul expresiei diferă de tipul variabilei la atribuire se face o conversie de tip a valorii expresiei la tipul variabilei **v**.

Este admis să se scrie:

v1 = v2 = v3 = v4 = expresie;

Operatorul de atribuire **=** se asociază de la dreapta la stânga, astfel sunt valabile relaţiile:

v4 = expresie; v3 = v4; v2 = v3; v1 = v2;

Operatorul de atribuire poate fi utilizat şi în combinaţie cu operatori binari aritmetici sau pe biţi (atribuire combinată), modalitatea de operare fiind prezentată în tabelul următor:

Operator	Exemplu	Scriere echivalentă
+=	a **+=** b	a **=** a **+** b
-=	a **-=** b	a **=** a **-** b
***=**	a ***=** b	a **=** a ***** b
/=	a **/=** b	a **=** a **/** b
%=	a **%=** b	a **=** a **%** b

Operator	Exemplu	Scriere echivalentă
<<=	a **<<=** b	a **=** a **<<** b
>>=	a **>>=** b	a **=** a **>>** b
&=	a **&=** b	a **=** a **&** b
^=	a **^=** b	a **=** a **^** b
|=	a **|=** b	a **=** a **|** b

1.3.9. Operatorul de forţare a conversiei la un anumit tip (cast)

Este un operator unar și se scrie sub forma:

(tip_conversie) operand;

unde: **tip_conversie** poate fi un tip de date predefinit al limbajului **C** sau un tip definit de utilizator. Utilizarea acestui operator are rolul de a modifica tipul rezultatului unei expresii. Cel mai adesea acest operator se utilizează atunci când funcţiile au parametri de un anumit tip, care trebuie respectat sau atunci când se doreşte ca rezultatul obţinut ca urmare a unui calcul să fie de un alt tip decât cel care ar rezulta în mod normal (implicit).

1.3.10. Operatorul dimensiune (sizeof)

Este operator unar, se asociază de la dreapta la stânga şi are ca efect determinarea numărului de octeţi pe care se face reprezentarea în memorie a unei date sau a unui tip.

Forma generală este:

sizeof(data); sau **sizeof(tip);**

unde: **data** poate fi un nume de variabilă simplă, de tablou, de structură, element de tablou sau element al unei structuri; **tip** poate fi un cuvânt sau cuvintele cheie care definesc un tip predefinit de date sau o construcție care definește un tip.

1.3.11. Operatorul adresă (&)

Este un operator unar, asociativitatea sa fiind de la dreapta la stânga, se foloseşte sub forma:

&operand

şi indică adresa zonei de memorie (RAM) care este alocată operandului.

1.3.12. Operatorul paranteză ( ), [ ]

Operatorul **( )** se utilizează la apelul funcţiilor, unde lista parametrilor efectivi se include între paranteze rotunde – operatori de apel de funcţie.

De asemenea, operatorului **( )** se utilizează pentru delimitarea subexpresiilor, acestea fiind primele care se evaluează și împreună cu operatorul paranteză **( )** se comportă ca un operand.

Operatorul **[ ]** se numeşte operator de indexare, se întâlneşte la tablouri şi conţine valoarea indicelui elementului unui tablou.

Operatorii **( )** şi **[ ]** fac parte din clasa întâi de prioritate.

1.3.13. Operatorul virgulă ,

Se utilizează pentru a lega două sau mai multe expresii pentru a forma o singură expresie a cărei valoare şi tip sunt identice cu valoarea şi tipul ultimei expresii. Are cea mai mică prioritate și se evaluează de la stânga la dreapta. În general se poate scrie sub forma:

expresie_1, expresie_2, ... , expresie_n

Se utilizează acolo unde sintaxa limbajului **C** permite utilizarea unei singure expresii, dar algoritmul de calcul impune folosirea mai multor expresii. Operatorul virgulă are cea mai mică prioritate şi se asociază de la stânga la dreapta.

1.3.14. Alţi operatori

***** (operatorul de indirectare) este operator unar, se utilizează împreună cu noţiunea de pointer pentru a accesa conţinutul aflat la o anumită adresă de memorie.

. sau **->** se folosesc împreună cu noţiunea de structură pentru a avea acces la membrii unei structuri.

1.4. Instrucţiuni

1.4.1. Introducere

Instrucţiunile unui program controlează execuţia acestuia. Implicit, ordinea de execuţie a instrucţiunilor este cea secvenţială. Unele instrucţiuni se execută în mod repetat, iar altele alternativ. Pe lângă cele trei posibilităţi de execuţie a instrucţiunilor (secvenţială, repetitivă şi alternativă) mai este permisă şi instrucţiunea selectivă.

Instrucţiune, în general, pot fi formate din: cuvinte cheie, expresii şi apeluri de funcţii.

Instrucţiunile limbajului **C** sunt:

- instrucţiuni simple;
- instrucţiuni compuse;
- instrucţiuni de ciclare (**for**, **while**, **do ... while**);
- instrucţiuni de decizie (**if**, **if .... else**, **switch**);
- instrucţiuni de întrerupere şi salt (**break**, **continue**, **goto**).

1.4.2. Instrucţiuni simple

Instrucţiunea expresie: este cea mai utilizată în cadrul programelor, constă dintr-o expresie urmată de caracterul punct și virgulă (**;**) şi are forma generală:

expresie;

unde **expresie** poate fi:

- o expresie de atribuire (instrucţiune de atribuire);
- un apel de funcţie (instrucţiune de apel de funcţie).

Instrucţiunea vidă: este formată doar din caracterul punct si virgulă (**;**), nu are nici un efect şi se utilizează acolo unde sintaxa limbajului necesită utilizarea unei instrucţiuni dar nu trebuie să se execute nimic în locul respectiv.

1.4.3. Instrucţiunea compusă

Instrucţiunea compusă se utilizează unde este necesară prezenţa unei instrucţiuni însă algoritmul presupune utilizarea mai multor instrucţiuni (de obicei în construcţia instrucţiunilor de decizie sau de ciclare).

Instrucţiunea compusă este o succesiune de instrucţiuni, care pot fi precedate de declaraţii, incluse între acolade:

```
{       declaraţii;
        instrucţiuni;
}
```

Declaraţiile din instrucţiunea compusă se referă la declararea variabilelor utilizate doar în interiorul acestei instrucţiuni compuse, la ieşirea din aceasta variabilele se pierd.

1.4.4. Instrucţiuni de decizie

Atunci când executarea unor instrucţiuni depinde de îndeplinirea unei condiţii este necesară utilizarea instrucţiunilor de decizie.

Instrucţiunile de decizie ale limbajului **C** sunt: **if, if … else** şi **switch**.

Instrucţiunea if: are forma generală:

```
...
if (expresie)
    instrucţiune
...
```

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucţiune** poate fi o instrucţiune simplă sau compusă.

La întâlnirea instrucţiunii **if** se evaluează mai întâi expresia din paranteze (adică **expresie**), iar dacă aceasta are valoare diferită de zero (adevărată), se execută **instrucţiune** şi se continuă cu instrucţiunea care urmează după **if**.

Dacă **expresie** are valoarea 0 (falsă) nu se execută nimic şi se continuă cu instrucţiunea care urmează după **if**.

Instrucţiunea if … else: are următoarea formă generală:

```
...
if (expresie)
        instrucţiune1
else
        instrucţiune2
...
```

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucţiune1** şi **instrucţiune2** pot fi instrucţiuni simple sau compuse.

La întâlnirea instrucţiunii **if** se evaluează mai întâi expresia din paranteze (adică **expresie**).

Dacă **expresie** are valoare diferită de zero (adevărată), se execută **instrucţiune1** şi se continuă cu instrucţiunea care urmează după **if**.

Dacă **expresie** are valoarea 0 (falsă), se execută **instrucţiune2** şi se continuă cu instrucţiunea care urmează după **if**.

Oricare din cele două instrucţiuni (**instrucţiune1** sau **instrucţiune2**) pot fi sau pot conţine alte instrucţiuni **if**, situaţie în care se spune că instrucţiunile **if** sunt imbricate.

Instrucţiunea de decizie multiplă

Instrucţiunea **if ... else** permite selecţia unei variante din două posibile. O selecţie multiplă (a unei variante din mai multe posibile) poate fi realizată cu mai multe instrucţiuni de decizie în cascadă. În cazul general în care există „**n**" alternative posibile selectate pe baza a „**n-1**" condiţii, se recomandă folosirea unei construcţii de forma:

```
...
    if( expresie_1 )
            instrucţiune_1
    else if( expresie_2 )
            instrucţiune_2
    ...
    else if( expresie_n-1 )
            instrucţiune_n-1
    else
            instrucţiune_n
...
```

La început se evaluează **expresie_1**, dacă este nenulă (adevărată) se execută **instrucţiune_1**, iar dacă este falsă se trece la evaluarea **expresie_2**. Dacă este adevărată se execută **instrucţiune_2**, iar dacă este falsă se trece la evaluarea **expresie_3**, ş.a.m.d. Dacă niciuna din cele **n-1** expresii nu sunt adevărate se execută **instrucţiune_n**.

1.4.5. Instrucţiuni de ciclare

Unele probleme pe care le rezolvăm cu ajutorul calculatorului presupun executarea unor etape în mod repetat, la fiecare execuţie putând exista variabile care îşi modifică valorile după anumite relaţii. Pentru aceasta se utilizează instrucţiunile de ciclare. Instrucţiunile de ciclare ale limbajului **C** sunt: **for, while, do…while.**

Instrucţiunea de ciclare for: este destinată pentru programarea ciclurilor care se execută de un număr cunoscut de ori, prin utilizarea unei variabile conducătoare de ciclu (contor). Are forma generală:

```
…
for ( expresie1 ; expresie2 ; expresie3 )
        instrucţiune
…
```

Semnificaţia elementelor componente ale instrucţiunii de ciclare **for** este următoarea:

expresie1 – se evaluează o singură dată, la intrarea în instrucţiunea **for**, înaintea primei iteraţii şi reprezintă expresia în cadrul căreia se iniţializează variabila conducătoare de ciclu (contorul).

expresie2 – corespunde testului cu ajutorul căruia se controlează execuţia. De obicei este o expresie logică, de a cărei valoare depinde execuţia corpului ciclului, se evaluează de fiecare dată înaintea fiecărei iteraţii.

expresie3 – reprezintă expresia prin care se modifică valoarea variabilei conducătoare de ciclu. Aceasta se evaluează de fiecare dată după executarea instrucţiunii ce formează corpul ciclului (**instrucţiune**). Este, în general, o expresie de incrementare sau decrementare a valorii variabilei conducătoare de ciclu.

instrucţiune – poate fi o instrucţiune simplă sau compusă şi reprezintă corpul ciclului.

La întâlnirea cuvântului cheie **for** se realizează următoarele:

- se evaluează expresia **expresie1**;
- se evaluează **expresie2**. Dacă valoarea acesteia nenulă (adevărată) se execută **instrucţiune** (corpul ciclului).

În continuare, după executarea corpului ciclului, se evaluează **expresie3** după care se revine evaluarea expresiei **expresie2**. Procedeul continuă cât timp **expresie2** este diferită de zero (adevărată). Dacă valoarea expresiei **expresie2** este nulă (falsă) se părăseşte ciclul, execuţia programului continuând cu următoarea instrucţiune care urmează după ciclul **for**.

Observaţii: Corpul ciclului **for** nu se execută niciodată dacă expresia **expresie2** are valoarea zero de la început. Dacă **expresie2** lipseşte, ea se consideră adevărată (ciclu for infinit). Oricare din cele trei expresii poate lipsi, însă caracterele punct și virgulă (**;**) trebuie scrise.

Cicluri for suprapuse: dacă instrucţiunea care reprezintă corpul unui ciclu **for** (numit ciclu for exterior) conţine un alt ciclu **for** (numit ciclu for interior), cele două cicluri **for** se numesc suprapuse, imbricate sau incluse.

Forma generală a ciclurilor **for** suprapuse este următoarea:

```
…
for(expresie1_1 ; expresie1_2 ; expresie1_3 )
{       …
        instrucţiune1
        for(expresie2_1 ; expresie2_2 ; expresie2_3 )
                {       instrucţiune2       }
        …
}
…
```

Observație: Ciclul interior trebuie să fie cuprins complet în corpul ciclului exterior.

Instrucţiunea de ciclare while: se utilizează pentru programarea unor operaţii de un număr de ori, de obicei necunoscut. Este o instrucţiune de ciclare *condiţionată anterior*.

Are forma generală:

```
…
while(expresie)
        instrucţiune
…
```

Interpretarea sintaxei este următoarea: "atât timp cât … ".

La întâlnirea instrucţiunii **while**, se evaluează **expresie**. Dacă aceasta este nenulă (adevărată) se execută **instrucţiune**, după care se revine la evaluarea **expresiei**. În cazul în care **expresie** este nulă (falsă) se trece la instrucţiunea care urmează în program.

Instrucţiunea de ciclare do … while: se utilizează la fel ca şi instrucţiunea **while** pentru programarea ciclurilor care se execută de un număr de ori, de obicei necunoscut. Diferenţa dintre cele două instrucţiuni constă în faptul că **do … while** este o instrucţiune de ciclare condiţionată posterior. Aceasta înseamnă că mai întâi se execută corpul ciclului şi abia apoi se verifică condiţia.

Are forma generală:

```
…
do
        instrucţiune
while( expresie);
…
```

Această instrucțiune conţine două cuvinte cheie: **do** (execută), respectiv **while**. Interpretarea sintaxei este următoarea: „execută … atât timp cât … ". Instrucţiunea se încheie cu caracterul punct și virgulă (**;**).

La întâlnirea instrucţiunii **do ... while** se execută mai întâi **instrucţiune** după care se evaluează **expresie**. Dacă valoarea acesteia este nenulă (adevărată) se revine la început şi se execută **instrucţiune**. În cazul în care valoarea **expresie** este nulă (falsă), se părăseşte ciclul, execuţia programului continuând cu instrucţiunea imediat următoare.

Observaţii:

- diferenţa dintre cele două instrucțiuni **while** şi **do … while** constă în aceea că în cazul ciclului **do … while** instrucţiunea se execută cel puţin o dată chiar dacă **expresie** este falsă, spre deosebire de **while** unde **instrucţiune** nu se execută dacă **expresie** este falsă;
- dacă **expresie** este întotdeauna nenulă (adevărată) ciclul este **infinit**;
- între cele trei instrucțiuni de ciclare există următoarea corespondență:

for (expresie1; expresie2; expresie3)		**expresie1;**		**expresie1;**
instrucțiune	⇔	**while (expresie2)**	⇔	**do**
		instrucțiune		**{ instrucțiune**
		expresie3;		**expresie3;**
				}while(expresie2);

1.4.6. Instrucţiunea break

Are forma generală:

```
break;
```

şi se utilizează:

- în corpul unei instrucţiuni de ciclare (**for, while sau do…while**) pentru a întrerupe forţat ciclul şi a ieşi din acesta, execuţia continuând cu instrucţiunea care urmează după ciclu.
- în cadrul instrucţiunii **switch**, utilizată la sfâşitul unui bloc de instrucţiuni corespunzătoare unui caz, determină ieşirea în afara instrucţiunii **switch** după efectuarea acelui bloc.

1.4.7. Instrucţiunea continue

Această instrucţiune se utilizează numai în corpul unui ciclu, cu scopul de abandonare a iteraţiei curente a ciclului. Efectul pe care îl produce utilizarea instrucţiunii **continue** diferă în funcţie instrucţiunea în care este utilizată, astfel:

- în corpul instrucţiuni de ciclare **for** se abandonează iteraţia curentă şi se trece la execuţia reiniţializării;
- în corpul instrucţiunilor de ciclare **while** sau **do…while** se abandonează iteraţia curentă şi se revine la evaluarea expresiei pe baza căreia se stabileşte continuarea sau terminarea ciclului (**expresie**).

Forma de utilizare a acestei instrucţiuni este următoarea:

```
continue;
```

1.4.8. Instrucţiunea goto

Este o instrucţiune de salt necondiţionat şi are următoarea sintaxă:

```
…
goto eticheta;
…
```

unde **eticheta** este un identificator al limbajului **C**.

Instrucţiunea are ca efect saltul necondiţionat în program la instrucţiunea în faţa căreia se găseşte **eticheta** urmată de caracterul două puncte (**:**). Saltul se execută numai în corpul funcţiei în care apare **goto**, adică are o acţiune locală. După executarea saltului nu se mai poate reveni la locul unde era plasată instrucţiunea **goto** decât eventual cu o altă instrucţiune de salt necondiţionat.

1.4.9. Instrucţiunea switch

Instrucţiunea **switch** permite realizarea structurii selective cu mai multe opţiuni. Are forma generală:

```
 switch(expresie)
{ case c_1: instrucțiune _1  break;
  case c_2: instrucțiune _2  break;
…
  case c_n: instrucțiune _n  break;
  default: instrucțiune
}
```

unde: **expresie** este o expresie al cărei rezultat trebuie să fie de tip întreg, **c_1, c_2, …, c_n** sunt constante întregi, iar **instrucțiune _1, instrucțiune _2… instrucțiune _n, instrucțiune** pot fi instrucţiuni simple sau compuse dar care nu trebuie cuprinse între acolade.

La întâlnirea cuvântului **switch** se realizează următoarele:

- se evaluează expresia scrisă între paranteze rotunde (**expresie**);
- se compară pe rând valoarea **expresie** cu valorile constantelor **c_1, c_2, … , c_n**:

Dacă valoarea **expresie** este egală cu valoarea uneia dintre constante (de exemple **c_i**), se execută instrucţiunea corespunzătoare (**instrucțiune_i**).

Dacă valoarea **expresie** nu este egală cu nici una dintre constante se execută instrucţiunea aferentă variantei **default:** adică **instrucțiune**.

Observaţii:

1. utilizarea instrucţiunii **break** după **instrucțiune_i** are ca efect părăsirea instrucţiunii **switch** după executarea **instrucțiune_i**. În cazul în care instrucţiunea **break** lipseşte, după executarea **instrucțiune_i** se trece la executarea **instrucțiune_i+1**;
2. dacă ramura **default** lipseşte şi valoarea **expresie** nu este egală cu nici una din constantele **c_1, .. , c_n** atunci instrucţiunea **switch** nu are niciun efect;
3. instrucţiunea **switch** utilizată cu instrucţiunea **break** inclusă pe fiecare variantă este echivalentă cu utilizarea instrucţiunilor **if ... else … if** imbricate:

...		**...**
if(expresie == c_1)		**switch**(**expresie**)
instrucțiune _1		**{**
else if (expr == c_2)		**case c_1: instrucțiune _1 break**;
instrucțiune _2	⇔	**case c_2: instrucțiune _2 break**;
…		…
else if (expr == c_n)		**case c_n: instrucțiune _n break**;
instrucțiune _n		**default**: **instrucțiune**
else		**}**
instrucțiune		**...**
...		

1.5. Funcţii de bibliotecă matematică. Directive preprocesor. Macroinstrucţiuni.

1.5.1. Funcţii de bibliotecă matematică

În exemplele prezentate în capitolele următoare, în cadrul relaţiilor de calcul, se utilizează o serie de funcţii matematice, de exemplu funcţii trigonometrice. Biblioteca de funcţii a limbajului **C** reprezintă o colecţie de funcţii grupate după acţiunea lor, având prototipurile în diverse fişiere antet.

Funcţiile matematice uzuale sunt reprezentate în biblioteca de funcţii a limbajului **C**. Acestea au prototipurile în fişierul antet **math.h**, astfel că pentru a putea fi apelate şi utilizate fişierul antet va trebui să fie inclus la începutul fişierului în care se apelează aceste funcţii. În tabelul următor sunt prezentate o parte din funcţiile de bibliotecă matematică.

Prototipul funcţiei	Apelul	Semnificaţia matematică
double **sin**(double x);	sin(x);	funcţia sinus
double **cos**(double x);	cos(x);	funcţia cosinus
double **tan**(double x);	tan(x)	funcţia tangentă
double **asin**(double x);	asin(x);	funcţia arcsinus
double **acos**(double x);	acos(x);	funcţia arccosinus
double **atan**(double x);	atan(x)	funcţia arctangentă
double **exp**(double x);	exp(x);	funcţia exponenţială - e^x
double **log**(double x);	log(x);	logaritmul natural al lui x, ln(x)
double **log10**(double x);	log10(x);	logaritmul în baza 10, $\log_{10}(x)$
double **sqrt**(double x);	sqrt(x);	rădăcina pătrată a lui x
double **pow**(double x, double y);	pow(x, y);	x ridicat la puterea y, x^y
double **pow10**(double x);	pow10(x);	10 ridicat la puterea x, 10^x
double **ceil**(double x);	ceil(x);	calculează cel mai mic întreg mai mare sau egal cu x
double **floor**(double x);	floor(x);	calculează cel mai mare întreg mai mic sau egal cu x
int **abs**(int x);	abs(x);	valoarea absolută a variabilei întregi x
double **fabs**(double x);	fabs(x);	valoarea absolută a variabilei reale x
double **atan2**(double y, double x);	atan2(y, x);	arctangentă din raportul y/x
double **poly**(double x, int n, double c[]);	poly(x, n, c);	calculează valoarea în **x** a polinomului de gradul **n** cu coeficienţi daţi de elementele tabloului **c** astfel: c[0] – coeficientul termenului liber, c[1] – coeficientul lui x^1, …, c[n] coeficientul lui x^n.

Observaţii:
- la apelul acestor funcţii trebuie să se ţină cont de domeniile de definiţie ale acestora, altfel vor apărea erori;
- argumentele funcţiilor trigonometrice trebuie să fie în radiani.

1.5.2. Directive preprocesor

Directivele preprocesor au fost prezentate pe scurt la începutul acestui capitol, atunci când s-a prezentat forma generală a unui program în **C**.

Directivele preprocesor sunt instrucţiuni care se execută înaintea compilării unui program. Ele constau în general în substituţii cu ajutorul cărora pot fi realizate următoarele acţiuni:

- includeri de fişiere antet sau de fişiere sursă ale utilizatorilor;
- definiţii şi apeluri de macroinstrucţiuni;
- compilare condiţionată.

Directivele preprocesor încep cu caracterului **diez** (**#**), pot fi plasate oriunde în program însă în general se plasează la începutul programului deoarece acţiunea lor are loc din poziţia în care sunt scrise.

Observaţie: după o directivă preprocesor nu se pune niciodată caracterul punct și virgulă (**;**)

Cele mai utilizate directive preprocesor sunt: **#include** şi **#define**.

Directiva **#include**

Are două forme de utilizare:

#include<nume_fişier.h>

#include" cale nume_fişier_utilizator"

Prima variantă se foloseşte pentru includerea în fişierul sursă a unor fişiere antet (header) care conţin prototipurile funcţiilor din biblioteca limbajului **C**. Prin utilizarea parantezelor unghiulare (**<>**) între care se scrie numele fişierului antet se indică compilatorului că acesta se găseşte într-un director special şi anume în subdirectorul **INCLUDE** din directorul unde este instalat mediul de programare.

A doua variantă se foloseşte pentru a include în programul sursă fişiere cu funcţii definite de programator.

cale – este un argument opţional, cu ajutorul căruia se indică calea de la directorul rădăcină la directorul în care se găseşte fişierul care trebuie inclus în program. În cazul în care nu este specificată calea, fişierul antet va fi căutat în directorul curent.

Observaţie: este permisă includerea unui fişier text care la rândul său conţine includeri de alte fişiere text.

Directiva **#define**

Se utilizează cel mai frecvent pentru definirea constantelor simbolice. Poate fi plasată oriunde în program, acţionând de la linia în care apare până la sfârşitul programului sau până la întâlnirea unei directive **#undef**. Efectul directivei preprocesor **#define** constă în înlocuirea la preprocesare a numelui constantei simbolice cu valoarea acesteia.

1.5.3. Macroinstrucţiuni (macro-uri)

Pentru definirea unei macroinstrucţiuni se foloseşte tot directiva preprocesor **#define**, forma generală fiind:

#define NUME(p1, p2, … , pn) text

Observaţii:

- **NUME** reprezintă numele macro-ului, se scrie cu litere mari (dar nu este însă obligatoriu);
- între numele macroinstrucţiunii şi paranteză deschisă (**(**) nu trebuie să existe niciun caracter alb;
- **p1, p2, … , pn** reprezintă parametrii formali ai macro-ului, prin intermediul cărora se pot face înlocuiri în program cu succesiuni de caractere diferite, la diferite apeluri ale macro-ului;
- **text** este textul de substituţie, trebuie să conţină parametri formali şi să efectueze nişte operaţii;
- dacă **text** este un apel de funcţie sau o instrucţiune oarecare, la sfârşit nu se pune caracterul punct și virgulă (**;**);
- dacă **text** are o lungime mare, programatorul poate să continue scrierea pe linia următoare punând la sfârşitul primei linii caracterul backslash (****).

O asemănare între macrouri şi funcţii ar fi aceea că atât macrourile cât şi funcţiile au o definiţie şi mai multe apeluri. Deosebirea dintre cele două constă în modul de lucru al acestora:

- la apelul unei funcţii se înlocuiesc valorile parametrilor formali cu valorile parametrilor efectivi şi se transferă execuţia într-o zonă de memorie alocată codului executabil al funcţiei,
- în cazul macro-ului valoarea parametrului formal este succesiunea de caractere corespunzătoare din apel, aceasta înlocuieşte parametrul formal peste tot unde acesta apare în **text**-ul de substituţie al macro-ului.

1.6. Noţiuni generale despre funcţii

1.6.1. Introducere

Noţiunea de funcţie este foarte importantă în limbajul **C**, datorită faptului că un program **C** este compus în principal din funcţii. Un program scris în **C** conţine o funcţie principală, numită **main()**, execuţia programului fiind asigurată de această funcţie.

Din corpul funcţiei **main()** pot fi apelate alte funcţii care să realizeze anumite acţiuni specifice. Aceste funcţii apelate pot fi funcţii de bibliotecă sau pot fi funcţii definite de programator.

La apelul unei funcţii de bibliotecă trebuie ca înainte de apel să se includă în program (prin intermediul directivelor preprocesor **#include**) fişierul antet care conţine prototipul funcţiei. Datorită acestei includeri, în etapele de compilare şi link-editare funcţia de bibliotecă va fi recunoscută, fişierul antet fiind căutat şi legat la programul sursă.

Un programator care doreşte să-şi scrie propriile funcţii trebuie să aibă în vedere trei elemente: **definiţiile**, **prototipul** şi **apelul** funcţiilor.

Noţiunea de funcţie a apărut în contextul modularizării programelor, astfel încât fiecare program conține mai multe module care vor executa anumite acțiuni. Aceste module trebuie să fie coordonate din cadrul unui modul principal.

Avantajele utilizării funcţiilor sunt: funcţiile pot fi apelate din diferite părţi ale unui program şi pentru valori diferite ale argumentelor acestora, funcţiile pot fi utilizate şi de alţi programatori, funcţiile pot fi utilizate şi în alte programe.

1.6.2. Definiţia funcţiei

Forma generală a unei funcţii este:

```
tip nume_funcţie (listă_parametri)
{
    corp
}
```

unde: **tip** – poate fi un tip de date predefinit sau un tip definit de programator și reprezintă tipul valorii returnate de funcţie, o funcţie poate returna cel mult o valoare. Dacă funcţia nu returnează nici o valoare în loc de **tip** se scrie cuvântul cheie **void**;

nume_funcţie – reprezintă numele funcţiei, prin intermediul căruia funcţia este apelată;

listă_parametri cuprinde parametrii funcţiei de la definiţia acesteia. Pentru fiecare dintre acestea trebuie declarate în cadrul listei tipul şi numele parametrului. Dacă funcția nu are parametri, cuvântul cheie **void** se utilizează şi în interiorul parantezelor rotunde de după numele funcţiei. Dacă **tip** lipseşte din faţa numelui funcţiei se consideră implicit că aceasta returnează o valoare de tip **int**.

{ } – cele două acolade marchează începutul şi sfârşitul corpului funcţiei, acesta cuprinde în general declaraţii de variabile şi instrucţiuni.

Definiţiile de funcţii pot fi plasate în program înaintea funcţiei **main()** sau după aceasta, în orice ordine.

1.6.3. Apelul funcţiei

Apelul unei funcţii poate fi făcut din corpul funcţiei **main()** sau din corpul oricăror funcţii, existând posibilitatea de a defini o funcție care se poate apela pe ea însăşi (funcţie recursivă).

În limbajul **C**, apelul unei funcții poate fi făcut în două moduri:

A. Printr-o instrucţiune de apel, sub forma:

nume_funcţie(listă_argumente);

Este specifică funcţiilor care nu returnează valori la revenire. Lista argumentelor conţine parametrii efectivi de la apelul funcţiei, scriși separați prin virgulă. Valorile acestora trebuie să fie cunoscute în momentul apelului.

Observație: Între lista parametrilor de la definiţie şi lista argumentelor de la apel trebuie să existe concordanţă în ceea ce priveşte **tipul**, **numărul** şi **ordinea** acestora, altfel va apare eroare la compilare.

Tot printr-o instrucţiune de apel pot fi apelate şi funcţiile care returnează o valoare la revenire. În acest caz, valoarea returnată se pierde.

B. Al doilea mod de apelare a funcţiilor este specific funcţiilor care returnează valori, apelul acestora realizându-se ca şi operand într-o expresie. La revenirea din funcţie cu valoarea returnată de aceasta se vor efectua calculele din expresia în care s-a făcut apelul.

1.6.4. Prototipul funcţiei

Se formează prin scrierea primei linii din definiţia funcţiei urmată de caracterul punct și virgulă (**;**), astfel:

tip nume_funcţie (listă_ parametri);

Prototipul se mai numeşte şi *declaraţia funcţiei* deoarece are un rol similar cu cel al declarării variabilelor. Prin intermediul prototipului o funcţie este declarată înainte de funcţia **main()**, astfel fiind vizibilă în tot programul.

Dacă definiţia funcţiei a fost plasată în program înaintea funcţiei **main()** atunci prototipul nu mai este necesar.

Pentru prototipul unei funcţii există şi o formă prescurtată de scriere:

tip nume_funcţie (lista_tip_parametri);

Observaţie: între listele parametrilor din prototip şi listele argumentelor de la apel trebuie să existe corespondenţă de **tip**, **număr** şi **ordine**.

1.6.5. Funcţii care returnează valoare

În limbajul **C** o funcţie poate returna cel mult o valoare. Pentru aceasta, funcţiile care returnează valori au în corpul lor, la definiţie, instrucţiunea **return**. Apelul acestei instrucţiuni are ca efect două acţiuni:

- determină revenirea din corpul funcţiei apelate în corpul funcţiei în care s-a făcut apelul (apelante);
- returnează funcţiei din care s-a făcut apelul valoarea scrisă după cuvântul cheie **return**.

Dacă tipul valorii returnate nu coincide cu tipul valorii returnate scris la definiţie şi în prototip, la revenirea din funcţie se face o conversie de tip spre tipul din definiţie (sau prototip).

1.6.6. Funcţii cu parametri

Parametrii unei funcţii permit transferul de valori de la funcţia care a făcut apelul (funcţia apelantă) spre funcţia apelată.

În limbajul **C**, transferul valorilor de la funcţia apelantă către cea apelată se poate realiza în două moduri:

- prin intermediul argumentelor se transmit **valori** (transfer prin valoare);
- se utilizează ca şi argumente **pointeri** (variabile care memorează adresele de memorie ale altor variabile) astfel că, prin intermediul pointerilor, funcţia apelată are acces la adresele unor variabile fiind posibilă modificarea valorilor acestora.

În acest mod se realizează un pseudotransfer prin referinţă (adică prin adresele de memorie).

Capitolul 2. Reprezentarea algoritmilor

2.1. Introducere

Pentru rezolvarea unei probleme tehnice cu ajutorul calculatorului, trebuie parcurse următoarele etape:
- analiza problemei: constă în alegerea datelor de intrare / ieşire precum şi a modelului matematic de rezolvare;
- descrierea algoritmului de rezolvare: poate fi realizată **grafic** (scheme logice) sau **literal** (limbajul pseudocod);
- scrierea programului într-un limbaj de programare, utilizând un editor de texte, obţinându-se un fişier cu extensia **C** sau **CPP**;
- compilarea programului: constă în „traducerea” programului în limbajul cod-obiect, rezultând un fişier cu extensia **.obj**. În această etapă se realizează detectarea şi depanarea erorilor de sintaxă;
- asamblarea programului: constă în realizarea legăturilor dintre fişierul sursă şi bibliotecile de funcţii, obţinându-se o formă compactă a programului (fişier cu extensia **.exe**);
- rularea programului.

Algoritmul (originea cuvântului provine de la numele matematicianului Abu Ja’far Mohammed ibn Musâ al-Khowârizmî) reprezintă în matematică şi informatică o metodă sau o procedură de calcul, alcătuită din paşi elementari necesari pentru rezolvarea unei probleme sau a unei categorii de probleme. Algoritmul reprezintă un concept fundamental atât în matematică cât şi în informatică.

Printr-o definiţie mai elaborată a noţiunii de algoritm, acesta reprezintă o succesiune finită de paşi executabili, realizaţi într-o ordine bine definită, astfel încât pornind de la anumite date cunoscute (date de intrare), se obţin rezultate dorite (date de ieşire).

Cele mai importante proprietăţi ale algoritmilor sunt:
- **generalitatea**: un algoritm trebuie să rezolve o categorie (clasă) de probleme şi nu doar o problemă particulară a acelei categorii;
- **finitudinea**: reprezintă proprietatea algoritmilor de a se termina într-un număr finit de paşi;
- **eficienţa**: se referă la proprietatea unui algoritm de a se termina într-un număr rezonabil de paşi;
- **optimalitatea**: un algoritm este optim atunci când se termină după un număr minim de paşi;
- **corectitudinea**: proprietatea unui algoritm de a furniza o soluţie corectă;
- **caracterul univoc**: se referă la faptul că, plecând de la un anumit set de date iniţiale, rezultatul este unic, adică repetarea execuţiei algoritmului cu aceleaşi date de intrare conduce la acelaşi rezultat;
- **claritatea**: proprietatea algoritmului de a descrie cu exactitate şi fără ambiguităţi a paşilor care trebuie parcurşi pentru a rezolva problema;
- **verificabilitatea**: se referă la posibilitatea de a verifica fiecare pas al algoritmului.

Observaţie: un algoritm trebuie să fie independent de limbajul de programare în care este transpus, precum şi de calculatorul pe care este executat.

Scrierea programelor trebuie să se realizeze sistematic, respectând anumite reguli, fapt care conduce la obţinerea unor programe clare, uşor de înţeles şi depanat. Programarea structurată reprezintă un anumit mod de concepere a programelor utilizând reguli bine stabilite şi un set redus de structuri de control. Conform teoremei lui Bohm-Jacopini, orice algoritm poate fi compus din numai trei structuri de control:
- **structura secvenţială**: instrucţiunile se derulează una după alta;
- **structura alternativă**: instrucţiunile se derulează după un criteriu de selecţie;
- **structura repetitivă** (cu test iniţial, cu test final, cu număr cunoscut de paşi): instrucţiunile din cadrul structurii se repetă în funcţie de rezultatul unui test. Testul poate fi plasat la începutul sau la sfârşitul structurii.

2.2. Tipuri de date şi expresii

Algoritmii lucrează cu date.

Din punct de vedere logic, datele sunt definite prin trei elemente:
- **identificator**: reprezintă numele datei, format din unul sau mai multe caractere;
- **valoarea**: reprezintă conţinutul zonei de memorie în care este păstrată data;
- **tip**: descrie apartenenţa datei la o anumită clasă de date.

Clasificarea datelor:

A. În funcţie de momentul în care se introduc în fluxul de date:
- de intrare
- de ieşire
- de manevră

B. În funcţie de valoare:
- constante
- variabile

C. În funcţie de modul de compunere:
- date elementare
- structuri de date

D. În funcţie de tip:
- date numerice (reale sau întregi)
- date logice
- date şiruri de caractere

O **expresie** conţine **operanzi** şi **operatori**. **Operanzii** pot fi date constante, variabile sau alte expresii încadrate între paranteze rotunde. **Operatorii** desemnează operaţiile care se execută asupra operanzilor. Operatorii utilizaţi la întocmirea algoritmilor pot fi grupaţi astfel: operatori aritmetici, operatori relaţionali şi operatori logici.

Operatorii aritmetici definesc operaţiile aritmetice şi pot fi unari (adică se aplică unui singur operand) sau binari (acţionează asupra a doi operanzi).

Operatorii aritmetici utilizaţi sunt:

- operatori aditivi: **+** adunare, **-** scădere;
- operatori multiplicativi: ***** înmulţire, **/** (**DIV**) împărţire, **%** (**MOD**) restul împărţirii întregi;

Operatorii relaţionali sunt operatori binari şi se aplică operanzilor de tip numeric sau şir de caractere, rezultatul operaţiei fiind unul de tipul logic.

Operatorii relaţionali sunt: **=** egal, **≠** diferit, **<** mai mic, **>** mai mare, **≤** mai mic sau egal, **≥** mai mare sau egal;

Operatorii logici definesc operaţiile logice şi acţionează doar asupra operanzilor logici, rezultatul fiind unul de tip logic.

Operatorii logici sunt: **NOT** (**!**) – negare logică, **AND** (**&&**) – ŞI logic, **OR** (**||**) – SAU logic. În tabelul 2.1. este ilustrat modul de acţiune al operatorilor logici:

Tabelul 2.1. Operatori logici

a	b	!a	a || b	a && b
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

2.3. Reprezentarea algoritmilor prin scheme logice şi pseudocod

Modalităţile de reprezentare al algoritmilor sunt: scheme logice sau limbajul pseudocod.

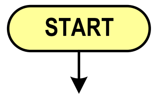
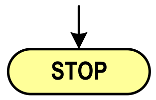
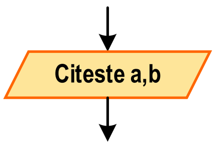
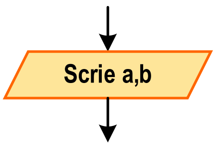
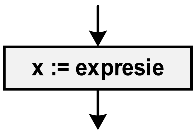
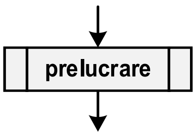
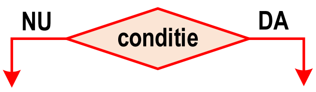
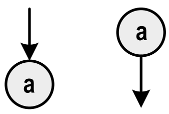
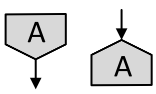
Schema logică reprezintă o modalitate de ilustrare a algoritmilor sub formă grafică, care permite vizualizarea succesiunii şi subordonării secvenţelor de operaţii utilizate, pentru fiecare din acestea utilizându-se blocuri grafice specifice.

Un dezavantaj al utilizării schemelor logice constă în faptul că, în cazul unor probleme mai dificile schemele logice pot fi stufoase, deci mai greu de urmărit.

Un avantaj al învăţării schemelor logice constă în faptul că acestea sunt utilizate şi în alte reprezentări, nu numai la algoritmi de rezolvare a problemelor din domeniul informaticii.

Blocurile utilizate în reprezentarea grafică al algoritmilor sunt ilustrate în tabelul următor (tabelul 2.2):

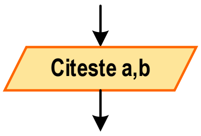
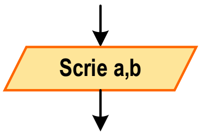
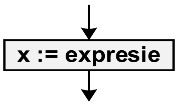
Tabelul 2.2. Blocurile utilizate în reprezentarea grafică a algoritmilor

Simbol:	Denumire:	Semnificaţie:
	Bloc terminal	Marchează începutul algoritmului.
	Bloc terminal	Marchează sfârşitul algoritmului.
	Bloc de **intrare** (citirea datelor de intrare)	Se utilizează pentru transferul datelor de la utilizator către algoritm.
	Bloc de **ieşire** (scrierea datelor de ieşire)	Se utilizează pentru transferul datelor de la algoritm către utilizator.
	Bloc de **atribuire**	Se utilizează pentru atribuirea valorii expresiei către variabila **a.**
	Bloc de **prelucrare (procedură)**	O prelucrare conţine mai multe instrucţiuni elementare. Se scriu instrucţiunile sau un nume care desemnează un grup de instrucţiuni.
	Bloc de **decizie**	Se evaluează condiţia obţinându-se o valoare logică „Adevărat" sau „Fals". Dacă condiţia este adevărată se execută ramura **DA**, iar dacă condiţia este falsă se execută ramura **NU.**
	Bloc conector logic	Se utilizează pentru conectarea unor puncte din schema logică situate în aceeaşi pagină.
	Bloc conector de pagină	Se utilizează pentru conectarea părţilor din schema logică care sunt reprezentate pe pagini diferite.

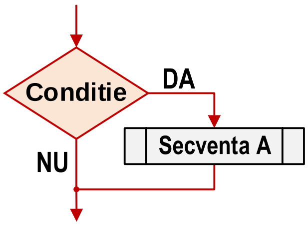
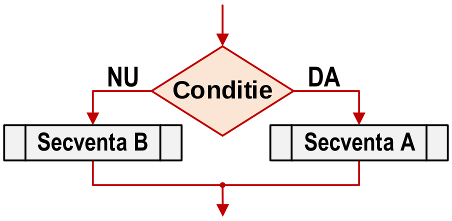
Limbajul pseudocod reprezintă un ansamblu de codificări cu ajutorul cărora se definesc operaţiile (instrucţiunile) utilizate pentru reprezentarea algoritmilor. Limbajul pseudocod conţine cuvinte cheie cu anumite semnificaţii.

Aşa cum s-a arătat anterior, orice algoritm poate fi alcătuit din trei structuri de control, corespondenţa dintre limbajul pseudocod şi reprezentarea grafică prin schema logică fiind ilustrată în continuare. De asemenea, sunt prezentate şi instrucţiunile aferente ale limbajului C.

I. Structura secvenţială: poate conţine o înşiruire de una sau mai multe instrucţiuni, ce se execută secvenţial (una după alta). Instrucţiunile pot fi: de citire / scriere de date, de atribuire, de prelucrare (procedură), sau combinaţii ale acestora.

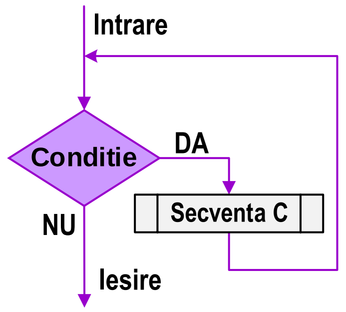
Schema logică	Limbajul pseudocod	Limbajul C
	Citeşte a,b	printf(”\n Introdu a = ”); scanf(”%d”,&a); printf(”\n Introdu b = ”); scanf(”%d”,&b); sau printf(”\n Introdu a = ”); scanf(”%f”,&a); printf(”\n Introdu b = ”); scanf(”%f”,&b);
	Scrie a,b	printf(”\n a = %d”,a); printf(”\n b = %d”,a); sau printf(”\n a = %f”,a); printf(”\n b = %f”,a);
	**x ← a + b**	**x = a + b;**

II. Structura alternativă (sau **decizia**): reprezintă alegerea unei operaţii sau a unei secvenţe de operaţii dintre două alternative posibile. Structura alternativă poate fi întâlnită în două variante, conform tabelului de mai jos:

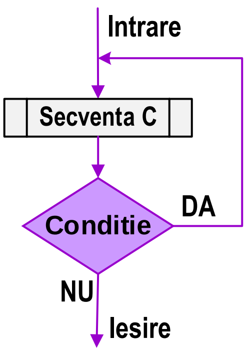
Schema logică	Limbajul pseudocod	Limbajul C
Varianta I:		
	**Dacă** *condiţie* **atunci** *Secventa A* **Sfârşit dacă**	**if (** *condiţie* **)** *Secventa A*;
Mod de lucru: - se evaluează **condiţie**, care de obicei este o expresie logică; - dacă aceasta este **adevărată** (ramura **DA**) se execută secvenţa A, iar dacă este **falsă** (ramura **NU**) nu se execută nimic;		
Varianta II:		
	**Dacă** *condiţie* **atunci** *Secventa A* **altfel** *Secventa B* **Sfârşit dacă**	**if (** *condiţie* **)** *Secventa A*; **else** *Secventa B*;
Mod de lucru: - se evaluează **condiţie**, care de obicei este o expresie logică; - dacă aceasta este **adevărată** (ramura **DA**) se execută secvenţa A, iar dacă este **falsă** (ramura **NU**) se execută secvenţa B;		

III. Structura repetitivă (sau **bucla**): constă în executarea repetată a unei operaţii sau a unei secvenţe de operaţii, în funcţie de o anumită condiţie. Dacă condiţia este adevărată, se realizează execuţia operaţiei sau a secvenţei de operaţii, iar dacă condiţia este falsă se părăseşte structura repetitivă, continuând cu următoarea structură din schema logică sau program. Sunt trei tipuri de structuri repetitive: cu **test iniţial**, cu **test final**, cu **contor** (variabilă conducătoare).

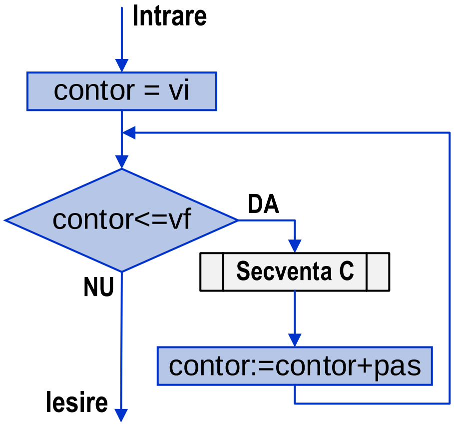
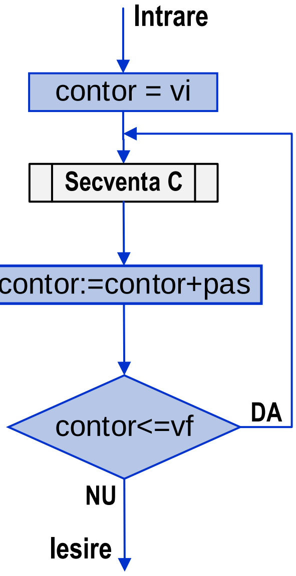
III.1. Structura repetitivă cu test iniţial (condiţionată anterior)

<table>
<tr><th>Schema logică</th><th>Limbajul pseudocod</th><th>Limbajul C</th></tr>
<tr><td>
</td><td>Cât timp condiţie execută
Secventa C
Sfârşit cât timp</td><td>while (condiţie)
Secventa C;</td></tr>
<tr><td colspan="3">Mod de lucru:
- se evaluează condiţie, care de obicei este o expresie logică;
- dacă condiţie este adevărată (ramura DA) se execută secvenţa C care reprezintă secvenţa de operaţii care formează corpul structurii repetitive, după care se revine la evaluarea condiţiei, ş.a.m.d.;
- dacă condiţie este falsă (ramura NU) se părăseşte structura repetitivă, algoritmul continuând cu structura imediat următoare;
Observaţii:
- structura repetitivă cu test iniţial se utilizează preponderent atunci când nu se cunoaşte numărul de repetări;
- dacă condiţie este falsă de la început, Secvenţa C nu se execută;</td></tr>
</table>

III.2. Structura repetitivă cu test final (condiţionată posterior)

<table>
<tr><th>Schema logică</th><th>Limbajul pseudocod</th><th>Limbajul C</th></tr>
<tr><td>
</td><td>Execută
Secventa C
Cât timp condiţie</td><td>do
Secventa C;
while (condiţie);</td></tr>
<tr><td colspan="3">Mod de lucru:
- se execută secvenţa C, care reprezintă secvenţa de operaţii care formează corpul structurii repetitive, după care se evaluează condiţie, care de obicei este o expresie logică;
- dacă condiţie este adevărată (ramura DA) se revine la execuţia secvenţei C, ş.a.m.d.;
- dacă condiţie este falsă (ramura NU) se părăseşte structura repetitivă, algoritmul continuând cu structura imediat următoare;
Observaţii:
- structura repetitivă cu test final se utilizează preponderent atunci când nu se cunoaşte numărul de repetări;
- dacă condiţie este falsă de la început, Secvenţa C se execută o singură dată;</td></tr>
</table>

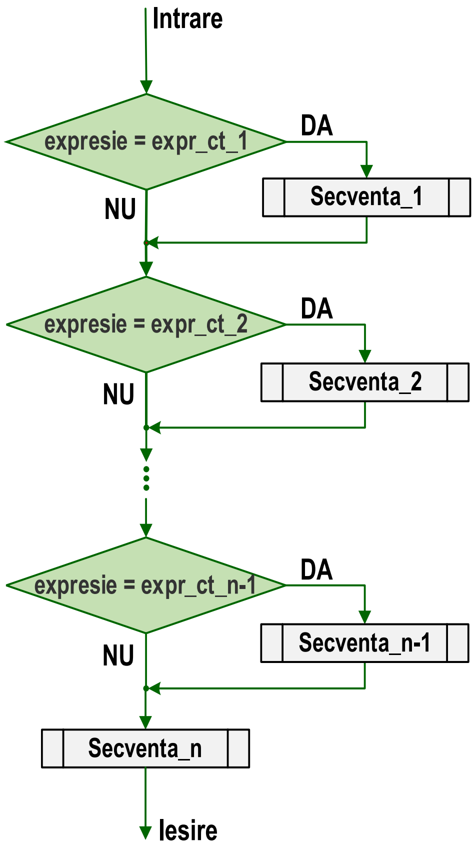
III.3. Structura repetitivă cu contor (cu variabilă conducătoare)

<table>
<tr><th colspan="2">Schema logică</th></tr>
<tr><td>Varianta I:
</td><td>Varianta II:
</td></tr>
<tr><td>Mod de lucru:
1. variabila contor primeşte valoarea iniţială, vi;
2. se verifică condiţia contor <= vf, iar dacă valoarea variabilei contor este mai mică sau egală decât valoarea finală, vf, (ramura DA) se execută secvenţa C. Dacă condiţia este falsă (ramura NU) se părăseşte structura repetitivă
3. se modifică valoarea variabilei contor, cu pasul pas, astfel contor := contor + pas;
4. se revine la pasul 2, prin care se verifică condiţia contor <= vf;</td><td>Mod de lucru:
1. variabila contor primeşte valoarea iniţială, vi;
2. se execută secvenţa C după care se modifică valoarea variabilei contor, cu pasul pas, astfel: contor := contor + pas;
3. se verifică condiţia contor <= vf, iar dacă valoarea variabilei contor este mai mică sau egală decât valoarea finală, vf, (ramura DA), se revine la pasul 2. Dacă condiţia este falsă (ramura NU) se părăseşte structura repetitivă</td></tr>
<tr><td colspan="2">Observaţii:
- structura repetitivă cu contor se utilizează atunci când se cunoaşte numărul de repetări;
- structura repetitivă cu contor poate fi utilizată în două variante: condiţionată anterior sau condiţionată posterior;</td></tr>
<tr><th>Limbajul pseudocod</th><th>Limbajul C</th></tr>
<tr><td>Pentru contor = vi, vf [, pas]
Secventa C
Sfârşit pentru</td><td>for (contor=vi ; contor<=vf ; contor=contor+pas)
Secventa C;</td></tr>
</table>

Structura repetitivă cu contor poate fi scrisă utilizând celelalte două structuri repetitive, astfel:

	Structura repetitivă cu contor	Structura repetitivă cu test iniţial	Structura repetitivă cu test final
Pseudocod	**Pentru** *contor = vi, vf, pas* *Secventa C* **Sfârşit pentru**	*contor = vi* **Cât timp** *contor <= vf* **execută** *Secventa C* *contor = contor + pas* **Sfârşit cât timp**	*contor = vi* **Repetă** *Secventa C* *contor = contor + pas* **până când** *contor <= vf*
Limbajul C	**for (**contor=vi **;** contor<=vf **;** contor=contor+pas **)** *Secventa C*;	contor=vi; **while(** contor<=vf **)** **{** *Secventa C*; contor=contor+pas; **}**	contor=vi; **do** **{** *Secventa C*; contor=contor+pas; **} while(** contor<=vf **);**

III.4. Structura de decizie cu ramuri multiple

<table>
<tr><th>Schema logică</th><th>Limbajul pseudocod</th><th>Limbajul C</th></tr>
<tr><td>
</td>
<td>Dacă expresie = expr_ct_1
Secvenţa_1;
[iesire;]
Altfel dacă expresie = expr_ct_2
Secvenţa_2;
[iesire;]
...
Altfel dacă expresie = expr_ct_n-1
Secvenţa_n-1;
[iesire;]
Altfel Secvenţa_n;</td>
<td>switch(expresie)
case expr_ct_1:
Secvenţa_1;
break;
case expr_ct_2:
Secvenţa_2;
break;
...
case expr_ct_n-1:
Secvenţa_n-1;
break;
default:
Secvenţa_n;</td></tr>
</table>

Mod de lucru:

- se evaluează **expresie**;
- se testează daca valoarea obţinută pentru **expresie** este egală cu constanta specificată **expr_ct_1**. Dacă cele două au valori egale se execută **Secvenţa_1**. În continuare, dacă se găseşte o instrucţiune care determină părăsirea structurii, execuţia continuă cu următoarea structură care urmează după structura de decizie cu ramuri multiple, în caz contrar, execuţia continuă cu următoarea ramură din structură, adică ramura corespunzătoare **expr_ct_2**. Dacă valoarea obţinută pentru **expresie** nu este egală cu constanta specificată **expr_ct_1** se trece la pasul următor;
- se verifică dacă valoarea **expresie** este egală cu constanta specificată **expr_ct_2** procedându-se ca în cazul precedent;
- procedeul continuă până la epuizarea tuturor ramurilor, în final executându-se **Secvenţa_n** dacă aceasta există.

Observaţii:

- **Secvenţa_n** poate să lipsească;
- dacă **expresie** nu este egală cu niciuna dintre expresiile constante, atunci se execută **instrucţiune_n**, dacă aceasta există.

Capitolul 3. Probleme de complexitate redusă

3.1. Interschimbarea valorilor a două variabile

În anumite probleme este necesar să se schimbe între ele valorile a două variabile.

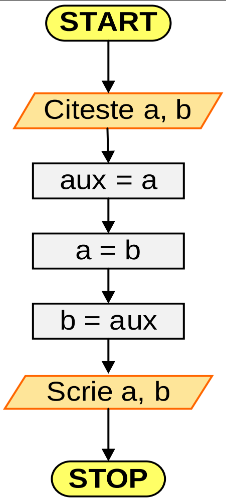
Să presupunem că sunt două variabile **a** şi **b**, de exemplu cu valorile **a** = **5**, respectiv **b** = **11**. Se doreşte elaborarea unui algoritm cu ajutorul căruia să se realizeze interschimbarea valorilor celor două variabile, astfel încât să avem **a** = **11**, respectiv **b** = **5**. La realizarea algoritmului, trebuie să se ţină cont de faptul că valoarea numerică a fiecărei variabile este reţinută într-o anumită zonă de memorie iar în urma atribuirii unei alte valori, valoarea anterioară se pierde.

Prima variantă necesită utilizarea unei variabile auxiliare (variabila **aux**), algoritmul fiind următorul:

- se citesc cele două valori pentru **a** şi **b**;
- se atribuie variabilei **aux** valoarea variabilei **a**;
- se atribuie variabilei **a** valoarea variabilei **b**;
- se atribuie variabilei **b** valoarea variabilei **aux**;
- se afişează valorile celor două variabile **a** şi **b**.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.1a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.1b.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte a,b
  aux ← a
  a ← b
  b ← aux
  Scrie a,b
 Sfârşit
```

Figura 3.1a. Reprezentarea algoritmului pentru interschimbarea valorilor a două variabile – varianta 1

Programul C:

```
#include<stdio.h>
int main(void)
{  int a,b,aux;
   printf("\n Introdu a = "); scanf("%d",&a);
   printf("\n Introdu b = "); scanf("%d",&b);
   aux = a;  a = b;  b = aux;
   printf("\n a = %d \t b = %d",a,b);
}
```

Rularea programului:

```
Introdu a = 5
Introdu b = 11

a = 11      b = 5
```

Figura 3.1b. Programul C şi rularea acestuia pentru interschimbarea valorilor a două variabile – varianta 1

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

A doua variantă a algoritmului nu necesită utilizarea unei variabile auxiliare, astfel:

- se citesc cele două valori pentru **a** şi **b**;
- se atribuie variabilei **a** valoarea **a + b**;
- se atribuie variabilei **b** valoarea **a - b**;
- se atribuie variabilei **a** valoarea **a - b**;
- se afişează valorile celor două variabile **a** şi **b**.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.1c, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.1d.

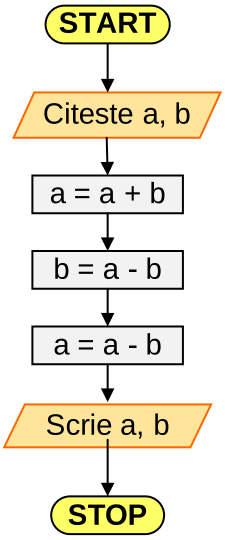
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a,b a ← a + b b ← a - b a ← a - b **Scrie** a,b **Sfârşit**

Figura 3.1c. Reprezentarea algoritmului pentru interschimbarea valorilor a două variabile – varianta 2

Programul C:

```c
#include<stdio.h>
int main(void)
{  int a,b,aux;
   printf("\n Introdu a = "); scanf("%d",&a);
   printf("\n Introdu b = "); scanf("%d",&b);
   a = a + b;
   b = a - b;
   a = a - b;
   printf("\n a = %d \t b = %d",a,b);
}
```

Rularea programului:

Introdu a = **5**
Introdu b = **11**

a = 11   b = 5

Figura 3.1d. Programul C şi rularea acestuia pentru interschimbarea valorilor a două variabile – varianta 2

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.2. Conversia unui unghi din grade în radiani

Radianul, simbolizat **rad**, reprezintă o unitate de măsură pentru unghiuri şi face parte din Sistemul Internaţional de Unităţi. Un radian reprezintă unghiul la centrul unui cerc care subîntinde un arc de lungime egală cu raza cercului. Un radian este egal cu 180° / π, sau aproximativ 57,2958° sau 57°17'45".

În informatică, argumentele funcţiilor trigonometrice sunt exprimate în radiani.

Pentru transformarea unui unghi din grade în radiani se utilizează relaţia de calcul:

$$ur = ug \cdot \frac{\pi}{180}$$

unde: **ug –** reprezintă unghiul în grade, **ur –** reprezintă unghiul în radiani;

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.2a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.2b.

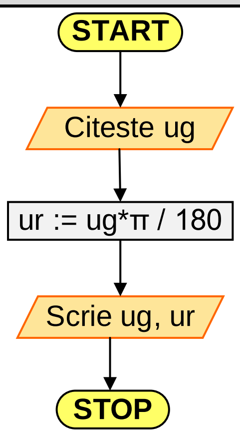
Schema logică	Rularea programului
	**Început** **Citeşte** ug ur ← ug*π/180 **Scrie** ug, ur **Sfârşit**

Figura 3.2a. Reprezentarea algoritmului pentru conversia unui unghi din grade în radiani

Programul C

```
#include<stdio.h>
#define PI 3.14159
int main(void)
{  float ug, ur;
   printf("\n Introdu unghiul [grade] = "); scanf("%f",&ug);
   ur = ug * PI / 180;
   printf("\n ug = %7.3f \t ur = %7.3f ",ug,ur);
}
```

Rularea programului

Introdu unghiul [grade] = **35**
ug = 35.000   ur = 0.611

Figura 3.2b. Programul C şi rularea acestuia pentru conversia unui unghi din grade în radiani

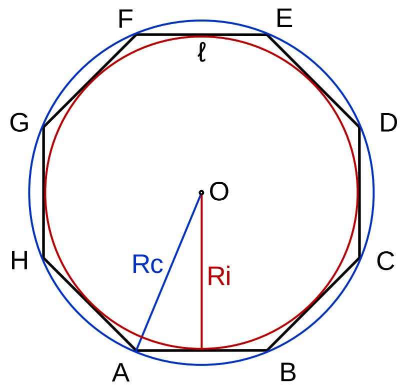
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.3. Calculul perimetrului şi ariei unui poligon regulat cu „n” laturi

În figura alăturată este reprezentat un poligon regulat cu „**n**” laturi pentru care se cunosc numărul de laturi „**n**” şi lungimea unei laturi, „**ℓ**”.

Pentru calculul ariei **A**, perimetrului **P**, razei cercului circumscris **Rc**, respectiv a razei cercului înscris **Ri** se utilizează relaţiile:

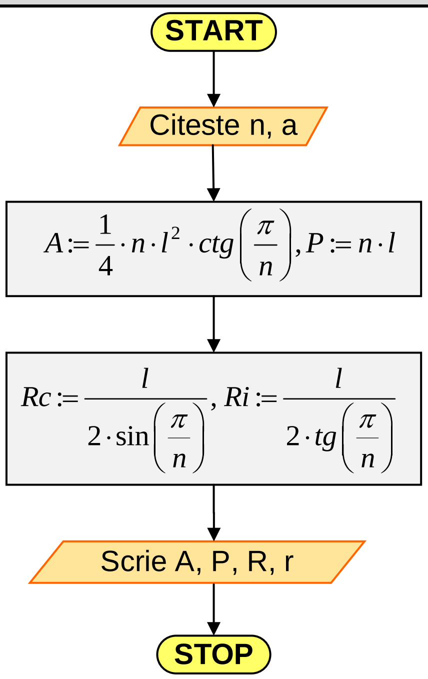
$$A := \frac{1}{4}\cdot n\cdot l^2\cdot ctg\left(\frac{\pi}{n}\right), P := n\cdot l,\ R := \frac{l}{2\cdot \sin\left(\frac{\pi}{n}\right)},\ r := \frac{l}{2\cdot tg\left(\frac{\pi}{n}\right)}$$



Algoritmul de calcul presupune parcurgerea următoarelor etape:
- citirea numărului de laturi **n** şi a lungimii unei laturi **ℓ**;
- calculul pe baza relaţiilor de mai sus a ariei - **A**, perimetrului - **P**, razei cercului circumscris - **Rc**, respectiv a razei cercului înscris - **Ri**;
- afişarea valorilor calculate.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.3a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.3b.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte n,a
   A ← n*l*l*ctg(π/n) / 4
   P ← n*l
   Rc ← l / 2 / sin(π/n)
   Ri ← l / 2 / tg(π/n)
   Scrie A,P,R,r
 Sfârşit
```

Figura 3.3a. Reprezentarea algoritmului pentru calculul ariei unui poligon regulat cu n laturi

Programul C

```
#include<stdio.h>
#include<math.h>
int main(void)
{  int n; float l, A, P, Rc, Ri;
   printf("\n Introdu numarul de laturi, n = ");
   scanf("%d",&n);
   printf("\n Introdu lungimea laturii, lat = ");
   scanf("%f",&l);
```

Rularea programului

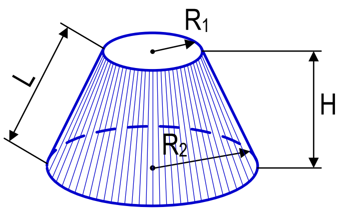
```
Introdu numarul de laturi, n = 6
Introdu lungimea laturii,  lat = 10
Aria poligonului          A = 259.808
Perimetrul poligonului P =   60.000
Raza cerc circumscris R =  10.000
Raza cerc inscris          r =     8.660
```

```
    A = n * l * l / tan(M_PI/n) / 4;  P = n * l;
    Rc = l/2/sin(M_PI/n);  Ri = l/2/tan(M_PI/n);
    printf("\n Aria poligonului          A = %7.3f ",A);
    printf("\n Perimetrul poligonului P = %7.3f ",P);
    printf("\n Raza cerc circumscris R = %7.3f ",Rc);
    printf("\n Raza cerc inscris          r = %7.3f ",Ri);
}
```

Figura 3.3b. Programul C şi rularea acestuia pentru calculul ariei unui poligon regulat cu n laturi

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.4. Calculul volumului, ariei laterale şi a ariei totale ale unui trunchi de con



Se consideră trunchiul de con din figura alăturată, pentru care se cunosc raza bazei mari – **R1**, raza bazei mici – **R2** şi înălţimea - **H**. Se cere să se determine volumul – **V**, aria laterală – **Alat** şi aria totală – **A**.

Pentru calcule se utilizează relaţiile de calcul:

$$L := \sqrt{H^2 + (R1 - R2)^2}, V := \frac{\pi H}{3} \cdot \left(R1^2 + R2^2 + R1 \cdot R2\right),$$

$$A_{lat} := \pi L(R1 + R2), A := \pi \cdot \left[L(R1 + R2) + R1^2 + R2^2\right]$$

Algoritmul de calcul presupune parcurgerea următoarelor etape:

- citirea datelor de intrare, adică a razei bazei mari - **R1**, a razei bazei mici - **R2** şi a înălţimii - **H**;
- calculul volumului - **V**, ariei laterale - **Alat** şi a ariei totale - **A** pe baza relaţiilor de mai sus;
- afişarea valorilor calculate.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.4a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.4b.

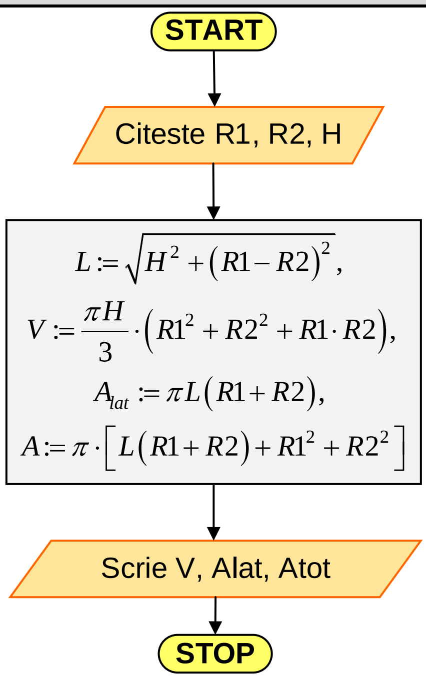
Schema logică	Limbajul pseudocod
	**Început** Citeşte R1,R2,H L ← sqrt(H*H+(R1-R2)*(R1-R2)) V ← π*H/3*(R1*R1+R2*R2+R1*R2) Alat ← π*L*(R1+R2) A ← π *(L*(R1+R2)+R1*R1+R2*R2) Scrie V, Alat, A **Sfârşit**

Figura 3.4a. Reprezentarea algoritmului pentru calculul volumului, ariei laterale şi ariei totale ale unui trunchi de con

Programul C	Rularea programului
``#include<stdio.h>``` ```#include<math.h>``` ```int main(void)``` ```{ float R1, R2, H, L, V, Alat, A;``` ```printf("\n Raza bazei mari, R1 [cm] = ");``` ```scanf("%f",&R1);``` ```printf("\n Raza bazei mici, R2 [cm]= ");``` ```scanf("%f",&R2);``` ```printf("\n Inaltimea, H [cm] = ");``` ```scanf("%f",&H);``` ```L = sqrt(H*H+pow(R1-R2,2));``` ```V = H*M_PI*(R1*R1+R2*R2+R1*R2)/3;``` ```Alat = M_PI*L*(R1+R2);``` ```A = M_PI*(L*(R1+R2)+R1*R1+R2*R2);``` ```printf("\n Volumul V = %7.3f [cm^3]",V);``` ```printf("\n Aria laterala Alat = %7.3f [cm^2]",Alat);``` ```printf("\n Aria totala A = %7.3f [cm^2]",A);``` ```}```	Raza bazei mari, R1 [cm] = **8** Raza bazei mici, R2 [cm] = **5** Inaltimea, H [cm] = **4** Volumul V = 540.354 [cm^3] Aria laterala Alat = 204.204 [cm^2] Aria totala A = 483.805 [cm^2]

Figura 3.4b. Programul C şi rularea acestuia pentru calculul volumului, ariei laterale şi ariei totale ale unui trunchi de con

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.5. Progresia aritmetică

O progresie aritmetică este un şir de numere reale a_n (n≥1) pentru care fiecare termen, începând cu al doilea, se obţine prin însumarea termenului precedent cu un număr **r**, numit raţia progresiei.

Se notează: a_1 – primul termen al progresiei, **r** – raţia progresiei aritmetice, a_n – termenul general al progresiei aritmetice.

Termenul general al unei progresii aritmetice se obţine pe baza relaţiei:

$$a_n := a_1 + (n-1)\cdot r$$

Suma primilor **n** termeni ai progresiei aritmetice, notată S_n se obţine pe baza relaţiei:

$$S_n := \frac{(a_1 + a_n)\cdot n}{2}$$

În continuare, este prezentat un program pentru calculul termenului de rang **k**, precum şi suma primilor **n** termeni ai unei progresii aritmetice pentru care se cunosc primul termen a_1 şi raţia **r**.

Algoritmul de calcul presupune parcurgerea următoarelor etape:

- citirea datelor de intrare: a_1 primul termen, raţia **r**, numărul de termeni **n**, precum şi rangul **k** al termenului pentru care se doreşte determinarea valorii;
- calculul valorii elementului a_k şi suma primilor **n** termeni, pe baza relaţiilor de mai sus;
- afişarea valorilor calculate.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.5a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.5b.

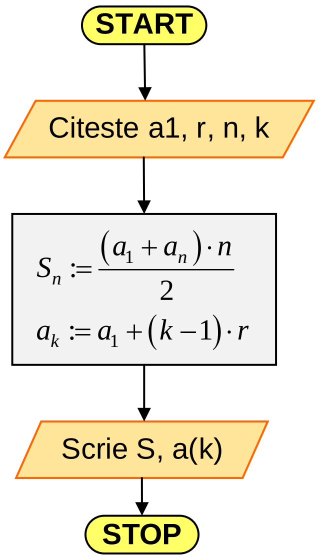
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a1,r,n,k Sn ← (2*a1+(n-1)*r)*n/2 ak ← a1+(k-1)*r **Scrie** Sn, ak **Sfârşit**

Figura 3.5a. Reprezentarea algoritmului pentru calculul termenului de rang **k** şi a sumei primilor **n** termeni dintr-o progresie aritmetică

Programul C	Rularea programului
`#include<stdio.h>` `#include<conio.h>` `int main(void)` `{ int a1,r,n,k,ak,Sn;` `printf("\n Introdu a1:");scanf("%d",&a1);` `printf("\n Introdu r:");scanf("%d",&r);` `printf("\n Introdu n:");scanf("%d",&n);` `printf("\n Introdu k:");scanf("%d",&k);` `Sn=(2*a1+(n-1)*r)*n/2; ak=a1+(k-1)*r;` `printf("\n Suma(%d) = %d",n,Sn);` `printf("\n a(%d) =%d",k,ak);` `getch();` `}`	Introdu a1= **2** Introdu r = **3** Introdu n = **10** Introdu k = **7** Suma(10) = 155 a(7) = 20

Figura 3.5b. Programul C şi rularea acestuia pentru calculul termenului de rang **k** şi a sumei primilor **n** termeni dintr-o progresie aritmetică

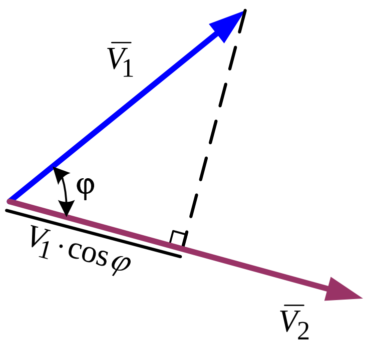
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.6. Produsul scalar a doi vectori

Produsul scalar a doi vectori $\overline{V}_1(V_{1x}\cdot\overline{\imath}+V_{1y}\cdot\overline{\jmath}+V_{1z}\cdot\overline{k})$, respectiv $\overline{V}_2(V_{2x}\cdot\overline{\imath}+V_{2y}\cdot\overline{\jmath}+V_{2z}\cdot\overline{k})$, este definit de relaţia:

$$\overline{V}_1\cdot\overline{V}_2 = |\overline{V}_1|\cdot|\overline{V}_2|\cdot cos\varphi$$

unde φ reprezintă unghiul dintre cei doi vectori.



Rezultatul acestui produs este o mărime scalară, interpretarea geometrică fiind aceea că produsul scalar a doi vectori reprezintă produsul dintre mărimea unui vector şi proiecţia celuilalt pe direcţia primului vector.

Expresia analitică a produsului scalar este: $\overline{V_1} \cdot \overline{V_2} = V_{1x} \cdot V_{2x} + V_{1y} \cdot V_{2y} + V_{1z} \cdot V_{2z}$

Unghiul format de cei doi vectori este dat de relaţia: $cos\varphi = \frac{\overline{V_1} \cdot \overline{V_2}}{|\overline{V_1}| \cdot |\overline{V_2}|} = \frac{V_{1x} \cdot V_{2x} + V_{1y} \cdot V_{2y} + V_{1z} \cdot V_{2z}}{\sqrt{V_{1x}^2 + V_{1y}^2 + V_{1z}^2} \cdot \sqrt{V_{2x}^2 + V_{2y}^2 + V_{2z}^2}}$

Algoritmul de calcul pentru produsul scalar, respectiv pentru unghiul dintre cei doi vectori presupune parcurgerea următorilor paşi: citirea datelor de intrare, adică a componentelor celor doi vectori: **V_{1x}, V_{1y}, V_{1z}, V_{2x}, V_{2y}, V_{2z}**; calculul produsului scalar şi a unghiului dintre cei doi vectori, pe baza relaţiilor de mai sus; afişarea valorilor calculate.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.6a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.6b.

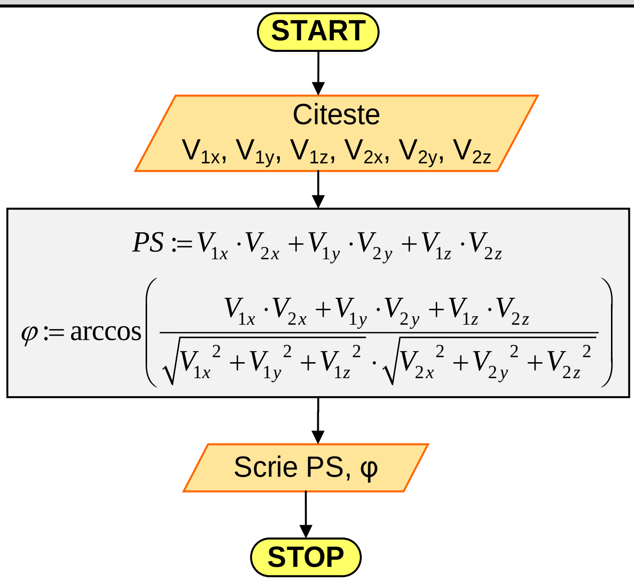
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** V1x,V1y,V1z,V2x,V2y,V2z PS ← V1x*V2x+V1y*V2y+V1z*V2z φ← arccos(PS/(sqrt(V1x*V1x+V1y*V1y+ V1z*V1z)*sqrt(V2x*V2x+V2y*V2y+V2z*V2z))) **Scrie** PS, φ **Sfârşit**

Figura 3.6a. Reprezentarea algoritmului pentru calculul produsului scalar

Programul C	Rularea programului
#include<stdio.h> **#include<math.h>** **int main(void)** **{ float** V1x,V1y,V1z,V2x,V2y,V2z,PS,fi; **printf(**"\n V1x = "**);scanf(**"%f",&V1x**);** **printf(**"\n V1y = "**);scanf(**"%f",&V1y**);** **printf(**"\n V1z = "**);scanf(**"%f",&V1z**);** **printf(**"\n V2x = "**);scanf(**"%f",&V2x**);** **printf(**"\n V2y = "**);scanf(**"%f",&V2y**);** **printf(**"\n V2z = "**);scanf(**"%f",&V2z**);** PS=V1x*V2x+V1y*V2y+V1z*V2z; fi=PS/(sqrt(V1x*V1x+V1y*V1y+V1z*V1z)* sqrt(V2x*V2x+V2y*V2y+V2z*V2z)); **printf(**"\n Produsul scalar PS = %6.3f",PS**);** **printf(**"\n Unghiul fi = %6.3f [grade]",acos(fi)*180/M_PI**);** **}**	V1x = **1** V1y = **2** V1z = **3** V2x = **5** V2y = **2** V2z = **4** Produsul scalar PS = 21.000 Unghiul fi = 33.211 [grade]

Figura 3.6b. Programul C şi rularea acestuia pentru calculul produsului scalar

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.7. Rezolvarea ecuaţiei de gradul al doilea

Ecuaţia algebrică de gradul al doilea este o ecuaţie polinomială de gradul doi. Forma generală a ecuaţiei este:

$$ax^2 + bx + c = 0$$

unde **x** este variabila, iar **a**,**b**,**c** sunt coeficienţi.

Rădăcinile ecuaţiei algebrice de gradul doi se obţin cu ajutorul formulei:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Algoritmul de calcul al rădăcinilor presupune parcurgerea următorilor paşi:

- citirea coeficienţilor **a**, **b** şi **c**;
- se verifică dacă **a** este nul, caz în care se repetă citirea lui **a** până când acesta are o valoare nenulă;
- se verifică condiţia ca $b^2 - 4ac \geq 0$. Dacă condiţia este falsă înseamnă că nu există rădăcini reale ale ecuaţiei şi se va afişa un mesaj corespunzător. Dacă condiţia este adevărată, se verifică dacă $b^2 - 4ac = 0$. Dacă această condiţie este adevărată, ecuaţia de gradul al doilea are o singură rădăcină reală, dată de relaţia:

$$x = \frac{-b}{2a}$$

După calculul rădăcinii reale se afişează valoarea acesteia.

Dacă condiţia este falsă atunci înseamnă că $b^2 - 4ac > 0$ atunci ecuaţia de gradul al doilea are două rădăcini reale. După calculul rădăcinilor se afişează valorile acestora.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.7a şi 3.7b, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.7c.

Schema logică

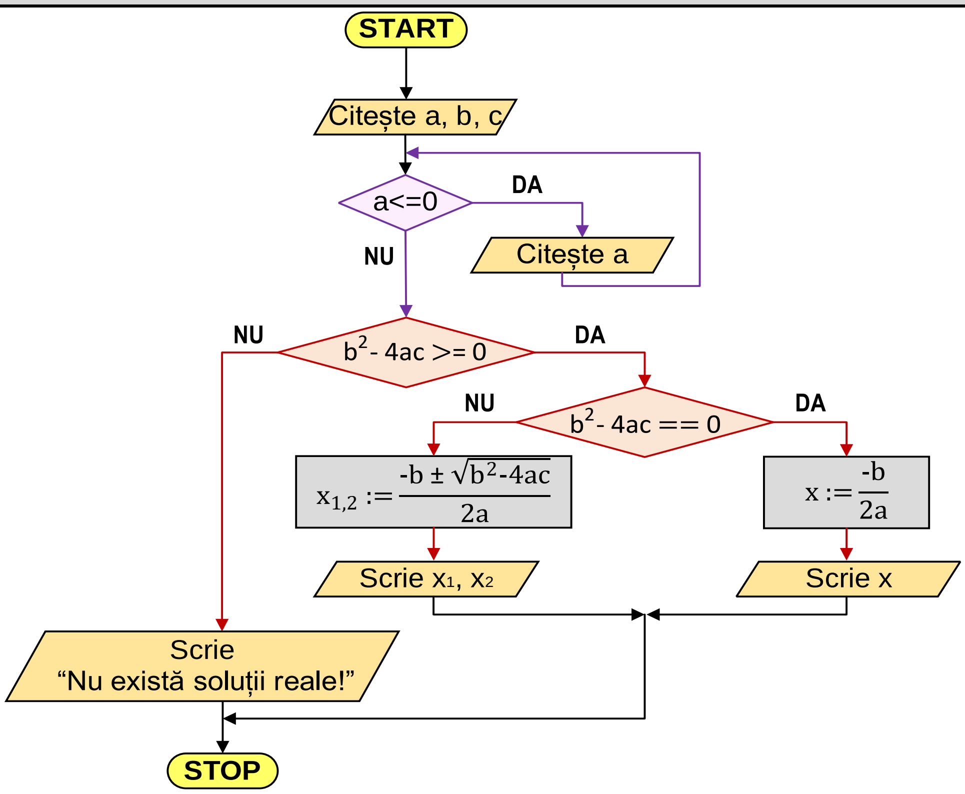


Figura 3.7a. Reprezentarea algoritmului pentru calculul rădăcinilor ecuaţiei de gradul al doilea – schema logică

Limbajul pseudocod

```
Început
  Citeşte a, b, c
  Cât timp a = 0 execută
  Citeşte a
  Sfârşit cât timp
  Dacă b * b – 4 * a * c ≥ 0 atunci
   Dacă b * b – 4 * a * c = 0 atunci
      x ← -b / ( 2 * a )
      Scrie x
   altfel
      x1 ← (-b + sqrt( b * b – 4 * a * c )) / ( 2 * a )
      x2 ← (-b - sqrt( b * b – 4 * a * c )) / ( 2 * a )
      Scrie x1, x2
   Sfârşit dacă
  altfel
      Scrie „Nu exista solutii reale!”
  Sfârşit dacă
Sfârşit
```

Figura 3.7b. Reprezentarea algoritmului pentru calculul rădăcinilor ecuaţiei de gradul al doilea - pseudocod

Programul C	Rularea programului
`#include<stdio.h>` `#include<math.h>` `int main(void)` `{    float x, x1, x2, a, b, c;` `     printf("\n a = "); scanf("%f",&a);` `     printf("\n b = "); scanf("%f",&b);` `     printf("\n c = "); scanf("%f",&c);` `     while(a==0)` `       { printf("\n a = "); scanf("%f",&a); }` `     if( b*b - 4*a*c >= 0 )` `          if( b*b - 4*a*c == 0 )` `               {  x = -b / (2 * a);` `                  printf("\n x = %6.3f",x);` `               }` `          else` `               {  x1 = (-b+sqrt(b*b-4*a*c)) / (2*a);` `                  x2 = (-b-sqrt(b*b-4*a*c)) / (2*a);` `                  printf("\n x1 = %6.3f \t x2 = %6.3f",x1,x2);` `               }` `          else` `               printf("\n Nu exista solutii reale!");` `}`	**Rularea programului – cazul 1:** a = **1** b = **-5** c = **6** x1 = 3.000 x2 = 2.000 **Rularea programului – cazul 2:** a = **1** b = **2** c = **1** x = -1.000 **Rularea programului – cazul 3:** a = **0** a = **3** b = **2** c = **5** Nu există solutii reale!

Figura 3.7c. Programul C şi rularea acestuia pentru calculul rădăcinilor ecuaţiei de gradul al doilea

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.8. Maximum a trei numere

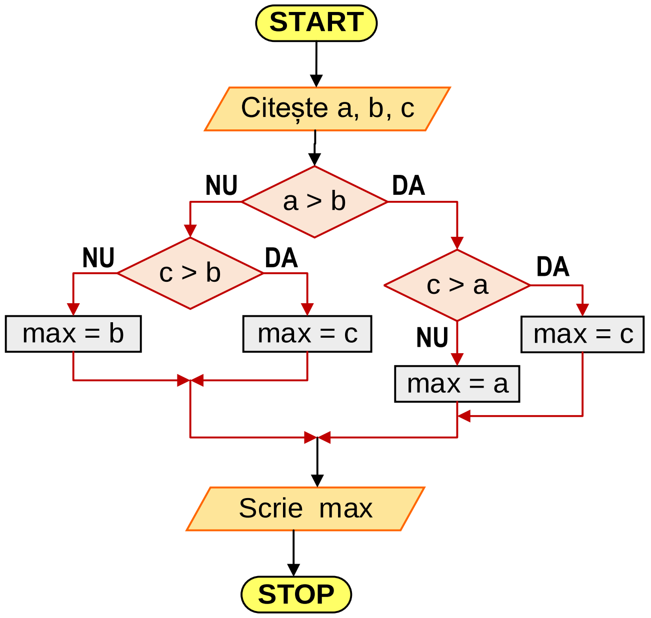
Se consideră trei valori: **a**, **b**, **c** şi se cere să se determine valoarea maximă dintre acestea.

Algoritmul este următorul:

- se compară prima dată valoarea variabilei **a** cu valoarea variabilei **b** punând condiţia **a > b**.
- dacă condiţia este adevărată atunci valoarea maximă obţinută până acum este **a**. În continuare, se compară valoarea variabilei **c** cu valoarea variabilei **a** punând condiţia **c > a**. Dacă condiţia este adevărată, maximul este variabila **c**, iar dacă condiţia este falsă atunci maximul este variabila **a**;
- dacă condiţia **a > b** este falsă, atunci valoarea maximă obţinută până acum este **b**. În continuare, se compară valoarea variabilei **c** cu valoarea variabilei **b** punând condiţia **c > b**. Dacă condiţia este adevărată, maximul este variabila **c**, iar dacă condiţia este falsă atunci maximul este variabila **b**.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.8a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.8b.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte a,b,c
  Dacă a>b atunci
    Dacă c>a atunci
      max←c
    altfel
      max←a
    Sfârşit dacă
  altfel
    Dacă c>b atunci
      max←c
    altfel
      max←b
    Sfârşit dacă
  Sfârşit dacă
Scrie max
Sfârşit
```

Figura 3.8a. Reprezentarea algoritmului pentru determinarea maximului dintre trei valori

Programul C

```
#include<stdio.h>
int main(void)
{ int a, b, c;
  printf("\n Introdu a, b, c:");   scanf("%d %d %d",&a, &b, &c);
  if(a > b)
        if(c > a) max = c;
        else max = a;
  else
        if(c > b) max = c;
        else  max = b;
  printf("\n max = %d",max);
}
```

Rularea programului

Exemplu numeric – cazul 1:
Introdu a, b, c :
1 2 3
max = 3

Exemplu numeric – cazul 2:
Introdu a, b, c :
3 2 1
max = 3

Exemplu numeric – cazul 3:
Introdu a, b, c :
1 3 2
max = 3

Figura 3.8b. Programul C şi rularea acestuia pentru determinarea maximului dintre trei valori

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.9. Rezolvarea unui sistem de două ecuaţii cu două necunoscute

Se consideră un sistem de două ecuaţii cu două necunoscute:

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}, \qquad \text{unde: } a, b, c, d, e, f \in \mathbb{R}.$$

Pentru rezolvarea sistemului se utilizează regula lui Cramer, care spune că dacă determinantul matricei principale obţinute pe baza coeficienţilor ecuaţiilor este diferit de zero, atunci soluţia sistemului de ecuaţii se determină cu ajutorul relaţiilor:

$$x = \frac{c \cdot e - b \cdot f}{a \cdot e - b \cdot d}, \quad y = \frac{a \cdot f - d \cdot c}{a \cdot e - b \cdot d};$$

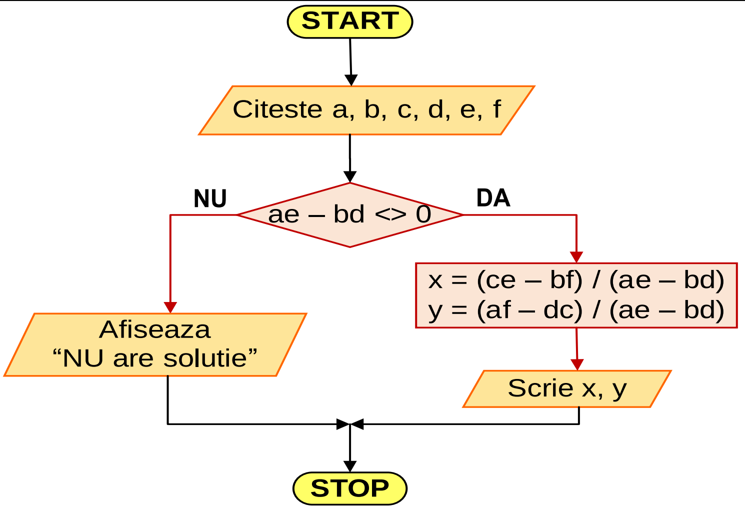
Algoritmul de rezolvare este următorul:

1. se citesc coeficienţii ecuaţiilor şi termenii liberi, adică: **a**, **b**, **c**, **d**, **e**, **f**;

2. se verifică dacă determinantul matricei principale este diferit de zero. Dacă condiţia impusă ($a \cdot e - b \cdot d \neq 0$) este **adevărată** se calculează soluţia cu relaţiile de mai sus şi se afişează, în caz contrar se afişează un mesaj corespunzător şi se încheie algoritmul;

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.9a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.9b.

Schema logică



Limbajul pseudocod

Început
  Citeşte a,b,c,d,e,f
  Dacă a*e-b*d≠0 **atunci**
    x ← (c * e – b * f) / (a * e – b * d)
    y ← (a * f – d * c) / (a * e – b * d)
    Scrie x, y
  altfel
    Scrie „NU are solutie!"
  Sfârşit dacă
Sfârşit

Figura 3.9a. Reprezentarea algoritmului pentru rezolvarea unui sistem de două ecuaţii cu două necunoscute

Programul C

```c
#include<stdio.h>
int main(void)
{ float a,b,c,d,e,f,x,y;
  printf("\n Introdu a, b, c:");  scanf("%f %f %f",&a,&b,&c);
  printf("\n Introdu d, e, f:");  scanf("%f %f %f",&d,&e,&f);
  if( a*e-b*d != 0 )
   { x=(c*e-b*f)/(a*e-b*d); y=(a*f-d*c)/(a*e-b*d);
      printf("\n x = %6.3f \t y = %6.3f",x,y);  }
  else printf("\n NU are solutie!!!");
}
```

Rularea programului

Rularea programului – cazul 1:
Introdu a, b, c: **2 -3 5**
Introdu d, e, f: **1 1 10**
x = 7.000   y = 3.000

Rularea programului – cazul 2:
Introdu a, b, c: **1 1 1**
Introdu d, e, f: **1 1 1**
NU are solutie!!!

Figura 3.9b. Programul C şi rularea acestuia pentru rezolvarea unui sistem de două ecuaţii cu două necunoscute

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.10. Verificarea condiţiei de coliniaritate a trei puncte

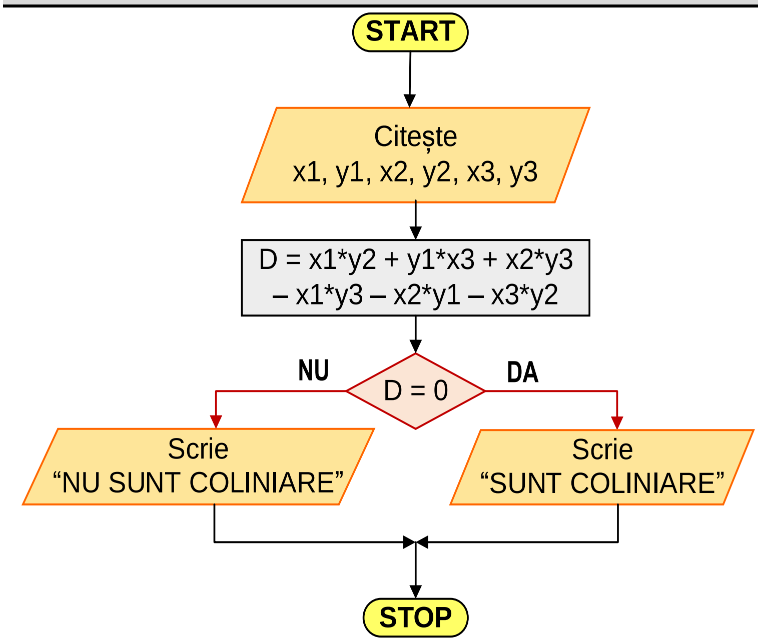
Se consideră trei puncte $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, $A_3(x_3, y_3)$ situate în planul Oxy. Se doreşte să se verifice dacă cele trei puncte sunt coliniare. Condiţia de coliniaritate este:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

Algoritmul de verificare constă în parcurgerea următorilor paşi:
- se citesc coordonatele celor trei puncte A_1, A_2, A_3;
- se calculează valoarea determinantului conform relaţiei de mai sus;
- se verifică dacă determinantul este nul. Dacă determinantul este nul cele trei puncte sunt coliniare, iar dacă determinantul este diferit de zero atunci cele trei puncte nu sunt coliniare;

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.10a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.10b.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte x1,y1,x2,y2,x3,y3
   D← x1*y2+y1*x3+x2*y3-x1*y3
          -x2*y1-x3*y2
   Dacă D=0 atunci
      Scrie „SUNT COLINIARE”
   altfel
      Scrie „NU SUNT COLINIARE”
   Sfârşit dacă
Sfârşit
```

Figura 3.10a. Reprezentarea algoritmului pentru verificarea condiţiei ca trei puncte să fie coliniare

Programul C

```
#include<stdio.h>
int main(void)
{ float x1,y1,x2,y2,x3,y3,D;
  printf("\n Introdu x1, y1 :");  scanf("%f %f",&x1,&y1);
  printf("\n Introdu x2, y2 :");  scanf("%f %f",&x2,&y2);
  printf("\n Introdu x3, y3 :");  scanf("%f %f",&x3,&y3);
  D=x1*y2 + y1*x3 + x2*y3 - x1*y3 - x2*y1 - x3*y2 ;
  if( D==0 )
    printf("\n SUNT COLINIARE!!!");
  else
    printf("\n NU SUNT COLINIARE!!!");
}
```

Rularea programului

Rularea programului – cazul 1:
Introdu x1, y1 : **1 1**
Introdu x2, y2 : **2 2**
Introdu x3, y3 : **3 3**
SUNT COLINIARE!!!

Rularea programului – cazul 2:
Introdu x1, y1 : **0 0**
Introdu x2, y2 : **2 0**
Introdu x3, y3 : **0 3**
NU SUNT COLINIARE!!!

Figura 3.10b. Programul C şi rularea acestuia verificarea condiţiei ca trei puncte să fie coliniare

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.11. Determinarea coordonatelor punctului de intersecţie a două drepte

Se consideră două drepte de ecuaţii: $\begin{cases} a_1 x + b_1 y + c_1 = 0 \\ a_2 x + b_2 y + c_2 = 0 \end{cases}$

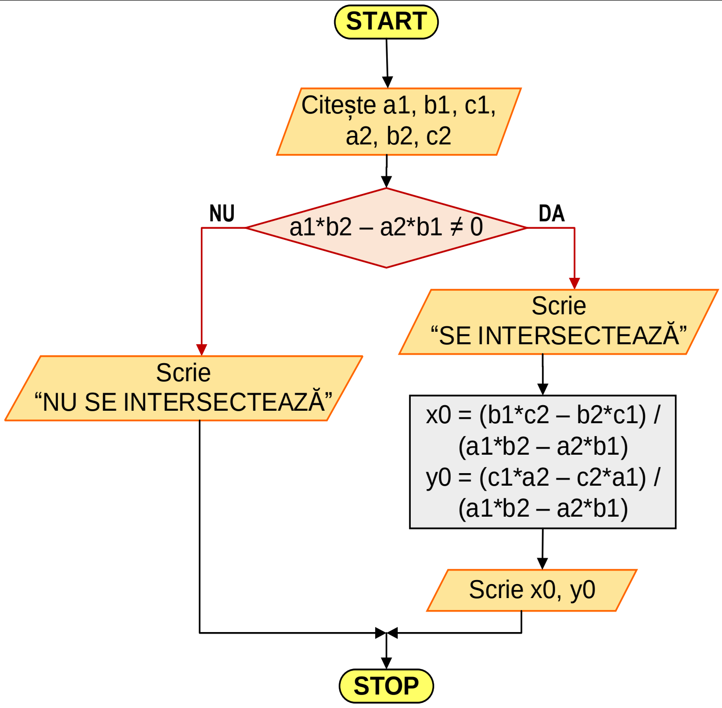
Condiţia ca cele două drepte să se intersecteze este: $a_1 \cdot b_2 - a_2 \cdot b_1 \neq 0$

Coordonatele punctului de intersecţie sunt: $x_0 := \dfrac{b_1 \cdot c_2 - b_2 \cdot c_1}{a_1 \cdot b_2 - a_2 \cdot b_1}$; $y_0 := \dfrac{c_1 \cdot a_2 - c_2 \cdot a_1}{a_1 \cdot b_2 - a_2 \cdot b_1}$

Algoritmul de calcul al coordonatelor punctului de intersecţie presupune parcurgerea următorilor paşi:
- se citesc coeficienţii ecuaţiilor celor două drepte, adică: **a_1**, **b_1**, **c_1**, respectiv **a_2**. **b_2**. **c_2**;
- se verifică dacă se îndeplineşte condiţia ca cele două drepte să se intersecteze:
  - dacă condiţia este **adevărată** se calculează cu relaţiile de mai sus coordonatele punctului de intersecţie;
  - dacă condiţia este **falsă** se afişează un mesaj corespunzător.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.11a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.11b.

Schema logică	Limbajul pseudocod




```
Început
   Citeşte a1,b1,c1,a2,b2,c2
   Dacă a1*b2-a2*b1≠0 atunci
      Scrie „SE INTERSECTEAZA”
      x0← (b1*c2-b2*c1)/(a1*b2-a2*b1)
      y0← (c1*a2-c2*a1)/(a1*b2-a2*b1)
      Scrie x0,y0
   altfel
      Scrie „NU SE INTERSECTEAZA”
   Sfârşit dacă
Sfârşit
```

Figura 3.11a. Reprezentarea algoritmului pentru calculul coordonatelor punctului de intersecție a două drepte

Programul C

```c
#include<stdio.h>
int main(void)
{ float a1,a2,b1,b2,c1,c2,x0,y0;
   printf("\n Introdu a1, b1, c1:"); scanf("%f %f %f",&a1,&b1,&c1);
   printf("\n Introdu a2, b2, c2:"); scanf("%f %f %f",&a2,&b2,&c2);
    if( a1*b2 - a2*b1 == 0 )  printf("\n NU SE intersecteaza!!!");
    else   { printf("\n SE INTERSECTEAZA!!!");
        x0 = (b1*c2-b2*c1) / (a1*b2-a2*b1);
        y0 = (c1*a2-c2*a1) / (a1*b2-a2*b1);
        printf("\n x0 = %6.3f \t y0 = %6.3f",x0,y0);   }
 }
```

Rularea programului

Rularea programului – cazul 1:
Introdu a1, b1, c1 : **5 2 -4**
Introdu a2, b2, c2 : **1 -3 -11**
SE INTERSECTEAZA!!!
x0 = 2.000 y0 = -3.000

Rularea programului – cazul 2:
Introdu a1, b1, c1 : **1 2 3**
Introdu a2, b2, c2 : **2 4 6**
NU SE INTERSECTEAZA!!!

Figura 3.11b. Programul C şi rularea acestuia pentru calculul coordonatelor punctului de intersecţie a două drepte

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.12. Transformarea din coordonate carteziene în coordonate polare

Se consideră un punct **P**, pentru care se cunosc coordonatele **x** şi **y** ale punctului într-un sistem de coordonate cartezian **Oxy**. Se doreşte să se determine coordonatele polare (**r** şi **θ**) ale punctului.

Algoritmul de transformare din coordonate carteziene în coordonate polare constă în: citirea coordonatelor carteziene, calculul coordonatelor polare pe baza relaţiilor de mai jos, respectiv afişarea acestora.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.12a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 3.12b.

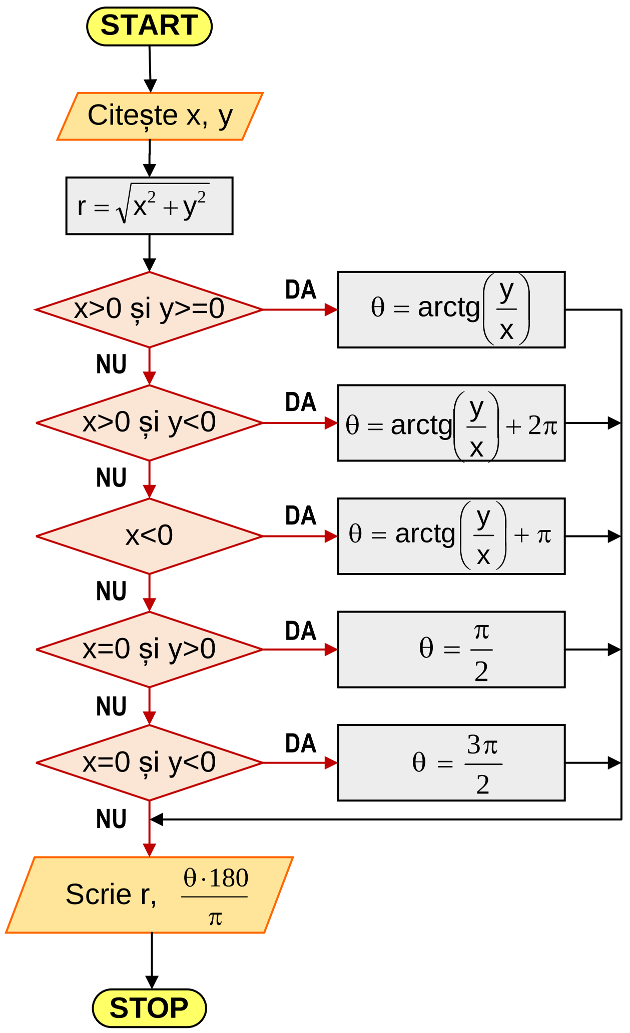
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** x,y r ← sqrt(x*x+y*y) **Dacă** x>0 ȘI y≥0 **atunci** θ ← arctg(y/x) **Sfârşit dacă** **Dacă** x>0 ȘI y<0 **atunci** θ ← arctg(y/x)+2*π **Sfârşit dacă** **Dacă** x<0 **atunci** θ ← arctg(y/x)+π **Sfârşit dacă** **Dacă** x=0 ȘI y>0 **atunci** θ ← π/2 **Sfârşit dacă** **Dacă** x=0 ȘI y<0 **atunci** θ ← 3*π/2 **Sfârşit dacă** **Scrie** r, θ*180/ π **Sfârşit**

Figura 3.12a. Reprezentarea algoritmului pentru transformarea din coordonate carteziene în coordonate polare

Sistemul de coordonate polar este un sistem de coordonate bidimensional în care fiecărui punct „**i**" i se asociază un unghi (**θ**) şi o distanţă (**r**). Astfel, fiecare punct este determinat de două coordonate polare:

- coordonata radială (notată cu **r**) care reprezintă distanţa unui punct faţă de un punct central numit pol (echivalent cu originea sistemului de coordonate cartezian);
- coordonata unghiulară (denumită unghi polar sau azimut şi notată cu **θ**) care reprezintă unghiul măsurat în sens trigonometric de la direcţia de 0°, numită axa polară (echivalentă cu axa absciselor din coordonatele carteziene).

Relaţiile de calcul pentru coordonatele polare sunt:

$$r=\sqrt{x^2+y^2}\text{ , respectiv: }\theta=\begin{cases}\arctan\left(\dfrac{y}{x}\right) & ,\quad daca\ x>0\ si\ y\ge 0\\ \arctan\left(\dfrac{y}{x}\right)+2\pi, & daca\ x>0\ si\ y<0\\ \arctan\left(\dfrac{y}{x}\right)+\pi\ , & daca\ x<0\\ \dfrac{\pi}{2} & daca\ x=0\ si\ y>0\\ \dfrac{3\cdot\pi}{2} & daca\ x=0\ si\ y<0\end{cases}$$

Programul C	Rularea programului
(see code below)	Introdu x [cm]: **18** Introdu y [cm]: **-12** Raza polara r = 21.633 [cm] Unghiul polar = 326.310 [grade]

```
#include<stdio.h>
#include<math.h>
int main(void)
{   float x,y,r,teta;
    printf("\n Introdu x [cm]:"); scanf("%f",&x);
    printf("\n Introdu y [cm]:"); scanf("%f",&y);
    r = sqrt(x*x+y*y);
    if( x>0 && y>=0 ) teta = atan(y/x);
    if( x>0 && y< 0 ) teta = atan(y/x)+2*M_PI;
    if( x<0 ) teta = atan(y/x)+M_PI;
    if( x==0 && y>0 ) teta = M_PI/2;
    if( x==0 && y<0 ) teta = 3*M_PI/2;
    printf("\n Raza polara r = %6.3f [cm]",r);
    printf("\n Unghiul polar = %6.3f [grade]",teta*180/M_PI);
}
```

Figura 3.12b. Programul C şi rularea acestuia pentru transformarea din coordonate carteziene în coordonate polare

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3.13. Descompunerea în factori primi a unui număr natural

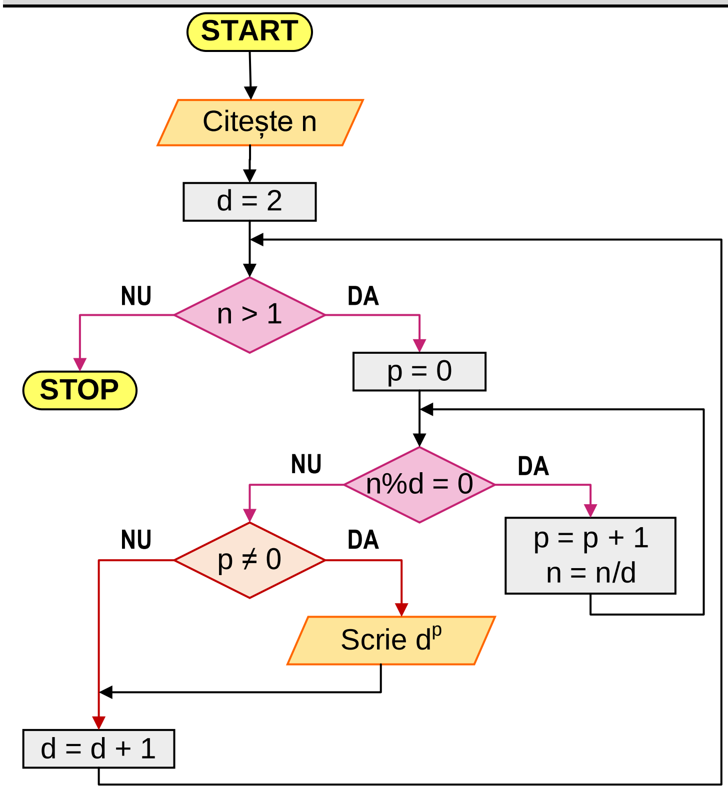
Descompunerea în factori primi a unui număr natural se bazează pe faptul că orice număr natural **n > 1** poate fi scris în mod unic sub forma:

$$n = p_1^{e1} \cdot p_2^{e2} \cdot \ldots \cdot p_k^{ek}$$

unde: $p_1 < p_2 < \ldots < p_n$ sunt numere prime (divizori ai numărului n), iar $ei > 0, \quad i = \overline{1,k}$

Reprezentarea algoritmului prin schema logică şi limbaj pseudocod este prezentată în figura 3.13a, iar programul C aferent şi rularea acestuia sunt ilustrate în figura 3.13b.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  d← 2
  Cât timp n>1 **execută**
    p← 0
    Cât timp n MOD d=0 **execută**
      p← p+1
      n← n DIV d
    Sfârşit cât timp
    Dacă p ≠ 0 **atunci**
      Scrie d^p
    Sfârşit dacă
    d ← d+1
  Sfârşit cât timp
Sfârşit

Figura 3.13a. Reprezentarea algoritmului pentru descompunerea unui număr în factori primi

Programul C	Rularea programului

```
#include<stdio.h>
int main(void)
{ int n, d = 2, p;
printf("\n Introdu n:");  scanf("%d",&n);
while( n > 1 )
 { p = 0;
   while( n % d == 0 )
      {  p = p + 1; n = n / d ;  }
   if( p )  printf("\n %d la puterea %d",d,p);
   d = d + 1;
 }
}
```

```
Introdu n: 2520
2 la puterea 3
3 la puterea 2
5 la puterea 1
7 la puterea 1
```

Figura 3.13b. Programul C şi rularea acestuia pentru descompunerea unui număr în factori primi

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Algoritmul de descompunere a unui număr natural în factori primi este următorul:

- se consideră pe rând divizorii lui **n** (notaţi cu **d**), începând cu **2**;
- atât timp cât **n > 1** se parcurg etapele următoare:
  1. se iniţializează puterea divizorului curent (notată cu **p**) cu **0**;
  2. se verifică de câte ori **n** se divide la **d** prin împărţire directă, crescându-se puterea **p** cu **1**. În acelaşi timp se modifică valoarea curentă a lui **n**, prin împărţirea acestuia la **d**, atât timp cât **d** divide pe noul **n**;
  3. dacă **p** este diferit de zero, se afişează divizorul lui **n** şi puterea acestuia.
- când **n** devine egal cu **1** algoritmul este încheiat;

Capitolul 4. Calculul valorilor unor funcţii

4.1. Calculul valorii unui polinom

Un polinom de gradul **n** în nedeterminata **X** se scrie în formă canonică astfel:

$$P(X) = c_n \cdot X^n + c_{n-1} \cdot X^{n-1} + \ldots + c_1 \cdot X^1 + c_0$$

unde: c_0, c_1, c_2, ... , c_{n-1}, c_n se numesc coeficienţii polinomului.

Numărul $P(a) = c_n \cdot a^n + c_{n-1} \cdot a^{n-1} + \ldots + c_1 \cdot a^1 + c_0$ se numeşte valoare a polinomului **P(X)** pentru **X = a**.

Algoritmul pentru calculul valorii unui polinom presupune parcurgerea următorilor paşi:
- se citeşte gradul polinomului, adică variabila **n**;
- se citeşte valoarea variabilei **a**;
- se iniţializează valoarea polinomului cu **0**;
- utilizând un ciclu cu contor, se citesc coeficienţii polinomului şi se calculează valoarea polinomului adunând în fiecare etapă câte un termen, relaţia de calcul fiind:

$$P := P + c_i \cdot a^i, \quad i = \overline{0, n}$$

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.1a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.1b.

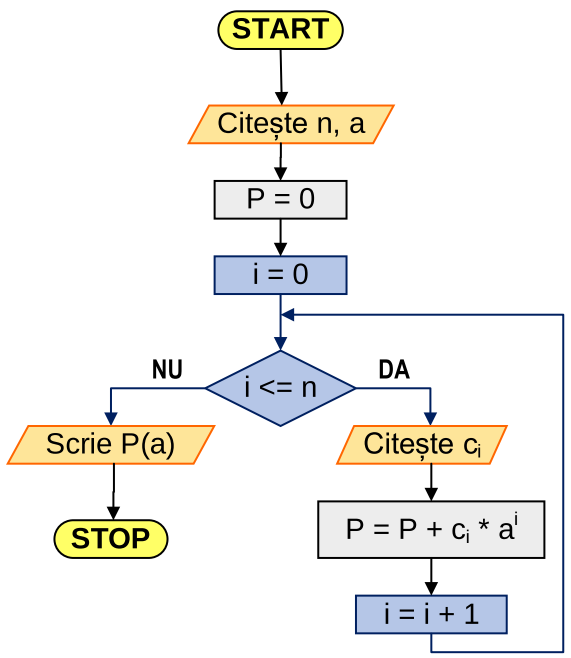
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n,a P ← 0 **Pentru** i ← 0, n **Citeşte** c_i P ← P +c_i*a^i i ← i + 1 **Sfârşit pentru** **Scrie** P **Sfârşit**

Figura 4.1a. Reprezentarea algoritmului pentru calculul valorilor unui polinom

Programul C	Rularea programului

```c
#include<stdio.h>
#include<math.h>
int main(void)
{ int i,n; float P = 0,a,c[20];
  printf("\n Gradul polinomului  n = ");  scanf("%d",&n);
  printf("\n Valoarea variabilei a = ");   scanf("%f",&a);
  for( i = 0 ; i <= n ; i++ )
   {  printf("\n c[%d] = ",i);  scanf("%f",&c[i]);
      P = P + c[i] * pow(a,i);
   }
  printf("\n P(%6.3f) = %6.3f",a,P);
}
```

```
Gradul polinomului n = 3
Valoarea variabilei  a = 5
c[0] = -120
c[1] = 74
c[2] = -15
c[3] = 1
P( 5.000) = 0.000
```

Figura 4.1b. Programul C şi rularea acestuia pentru calculul valorilor unui polinom

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

4.2. Calculul valorilor unei funcţii cu două ramuri

Se consideră funcţia: $y = f(x) = \begin{cases} x - 8 & daca \quad x \geq 1 \\ x^2 + 2x + 3 & daca \quad x < 1 \end{cases}$.

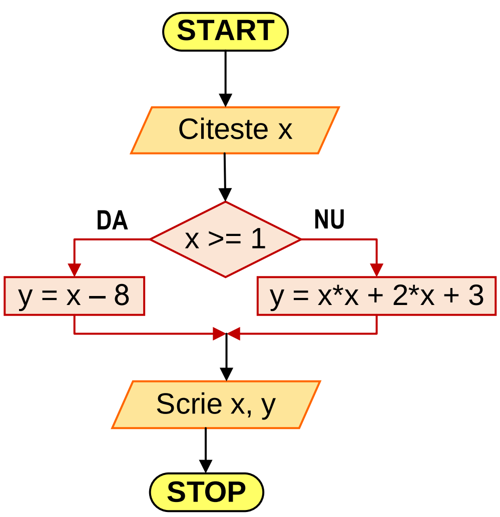
Se cere să se determine valoarea lui **y** pentru un **x** dat.

Algoritmul de calcul este următorul:

- se citeşte valoarea lui **x**;
- se compară valoarea lui **x** cu **1**, punând condiţia **x ≥ 1**. Dacă condiţia este adevărată, adică **x ≥ 1**, **y** se calculează cu relaţia: $y = x - 8$, iar dacă condiţia este falsă, adică **x < 1**, atunci **y** se calculează cu relaţia: $y = x^2 + 2x + 3$;
- se afişează valoarea lui **x** şi a lui **y**.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.2a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.2b.

Schema logică	Limbajul pseudocod




```
Început
   Citeşte x
   Dacă x ≥ 1 atunci
        y ← x - 8
   altfel
        y ← x * x + 2 * x + 3
   Sfârşit dacă
   Scrie x, y
Sfârşit
```

Figura 4.2a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu două ramuri

Programul C	Rularea programului
`#include<stdio.h>` `int main(void)` `{ float x,y;` `printf("\n Introdu x = "); scanf("%f",&x);` `if( x >= 1 )  y = x - 8;` `else   y = x*x + 2*x + 3;` `printf("\n x = %f \t y = %f",x,y);` `}`	**Exemplu numeric – cazul 1:** Introdu x = **4** x = 4 y = -4 **Exemplu numeric – cazul 2:** Introdu x = **0** x = 0 y = 3

Figura 4.2b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii cu două ramuri

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

4.3. Calculul valorilor unei funcţii cu trei ramuri – varianta 1

Se consideră funcţia: $y = f(x) = \begin{cases} x^2 - 3x + 4 & daca \quad x \geq 0 \\ \frac{x^2+2x+3}{x+2} & daca \quad x < 0 \end{cases}$

Se cere să se determine valoarea lui **y** pentru un **x** dat.

Algoritmul de calcul este următorul:

- se citeşte valoarea lui **x**;
- se compară valoarea lui **x** cu **0**, punând condiţia **x ≥ 0**. Dacă condiţia este adevărată, adică **x ≥ 0**, **y** se calculează cu relaţia: $y = x^2 - 3x + 4$ şi se afişează valorile lui **x** şi a lui **y**;
- dacă condiţia **x ≥ 0** este falsă, se observă că numitorul expresiei se anulează pentru **x = -2**, situaţie în care **y** nu se poate calcula. Astfel, pentru varianta în care **x < 0** trebuie să se impună o nouă condiţie, cea prin care se verifică dacă valoarea lui **x** este diferită de **-2**. Dacă această condiţie este adevărată atunci **y** se calculează cu relaţia: $y = \frac{x^2+2x+3}{x+2}$ şi se afişează valoarea lui **x** şi a lui **y**. Dacă a doua condiţie este falsă atunci nu se poate calcula **y** şi se afişează un mesaj corespunzător.

Reprezentarea algoritmului prin schema logică şi limbaj pseudocod este prezentată în figura 4.3a, iar programul C aferent şi rularea acestuia sunt ilustrate în figura 4.3b.

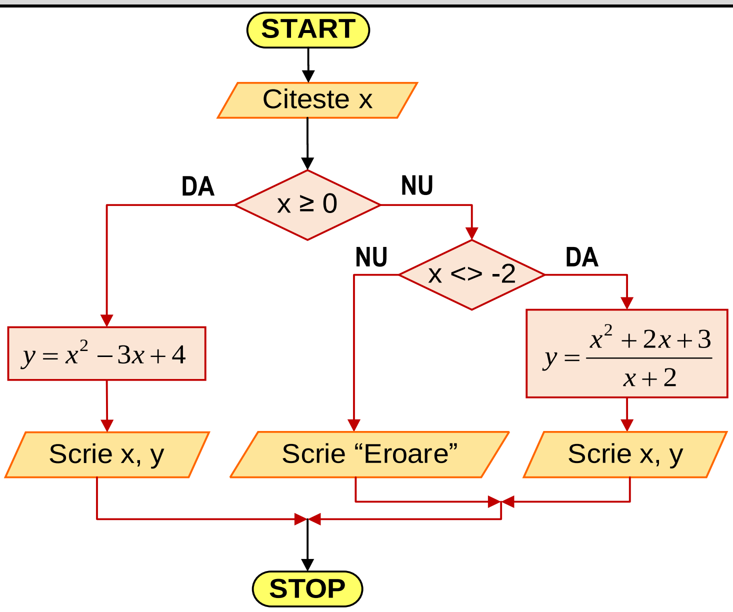
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** x **Dacă** x ≥ 0 **atunci** y ← x * x – 3 * x + 4 **Scrie** x, y **altfel** **Dacă** x ≠ -2 **atunci** y ← (x*x + 2*x + 3) / (x + 2) **Scrie** x, y **altfel** **Scrie** „Eroare” **Sfârşit dacă** **Sfârşit dacă** **Sfârşit**

Figura 4.3a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri – varianta 1

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { float x,y; printf("\n Introdu x = "); scanf("%f",&x); if(x >= 0) { y = x*x - 3*x + 4; printf("\n x = %f \t y = %f",x,y); } else if(x != -2) { y = (x*x+2*x+3) / (x+2); printf("\n x = %f \t y = %f",x,y); } else printf("\n Eroare!"); }</pre>	**Rularea programului – cazul 1:** Introdu x = **1** x = 1 y = 2 **Rularea programului – cazul 2:** Introdu x = **-1** x = -1 y = 2 **Rularea programului – cazul 3:** Introdu x = **-2** Eroare!

Figura 4.3b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii cu trei ramuri – varianta 1

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

4.4. Calculul valorilor unei funcţii cu trei ramuri – varianta 2

Se consideră funcţia: $y = f(x) = \begin{cases} x^2 - 3x + 2 & x > 2 \\ x - 8 & -3 \le x \le 2 \\ x^3 + 2x^2 - 4x - 5 & x < -3 \end{cases}$

Se cere să se determine valoarea lui **y** pentru un **x** dat.

Algoritmul de calcul este următorul:

- se citeşte valoarea lui **x**;
- se compară valoarea lui **x** cu **2**, punând condiţia **x > 2**. Dacă condiţia este adevărată, **y** se calculează cu relaţia: $y = x^2 - 3x + 2$;

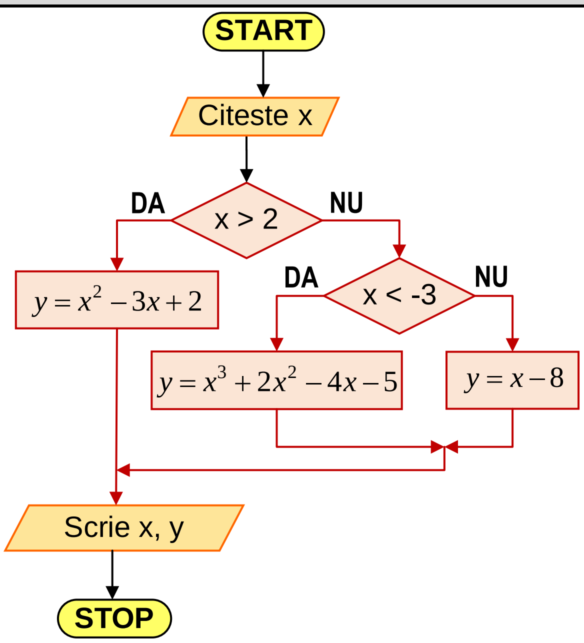
Schema logică	Limbajul pseudocod
	**Început** Citeşte x **Dacă** x > 2 **atunci** y ← x*x - 3*x + 2 **altfel** **Dacă** x < -3 **atunci** y ← x*x*x + 2*x*x - 4*x - 5 **altfel** y ← x - 8 **Sfârşit dacă** **Sfârşit dacă** **Scrie** x, y **Sfârşit**

Figura 4.4a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri – varianta 2

- dacă condiţia **x > 2** este falsă, se observă că pentru calculul valorii lui **y** avem în continuare două ramuri (două variante). Se impune astfel condiţia ca **x < -3**. Dacă această condiţie este adevărată **y** se calculează cu relaţia: $y = x^3 + 2x^2 - 4x - 5$, iar dacă condiţia este falsă se calculează **y** cu relaţia: $y = x - 8$;
- se afişează valorile lui **x** şi a lui **y**.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.4a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.4b.

Programul C	Rularea programului
`#include<stdio.h>` `int main(void)` `{ float x,y;` `  printf("\n Introdu x = "); scanf("%f",&x);` `  if( x > 2 )  y = x*x - 3*x + 4;` `  else` `     if( x < -3 )  y = x*x*x+2*x*x-4*x-5;` `     else y = x - 8;` `  printf("\n x = %f \t y = %f",x,y);` `}`	**Exemplu numeric – cazul 1:** Introdu x = **-5** x = -5.000, y = -40.000 **Exemplu numeric – cazul 2:** Introdu x = **0** x = 0.000, y = -8.000 **Exemplu numeric – cazul 3:** Introdu x = **4** x = 4.000, y = 6.000

Figura 4.4b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii cu trei ramuri – varianta 2

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

4.5. Calculul valorilor unei funcţii într-un interval

Se consideră funcţia: $y = x^2 + 2x + 3$.

Se cere să se calculeze toate valorile lui y pentru $x \in [a, b]$, parcurs cu pasul h.

Algoritmul de calcul constă în:

- citirea limitelor intervalului în care variabila **x** ia valori (adică variabilele **a** şi **b**), respectiv a pasului cu care se parcurge intervalul (variabila **h**);

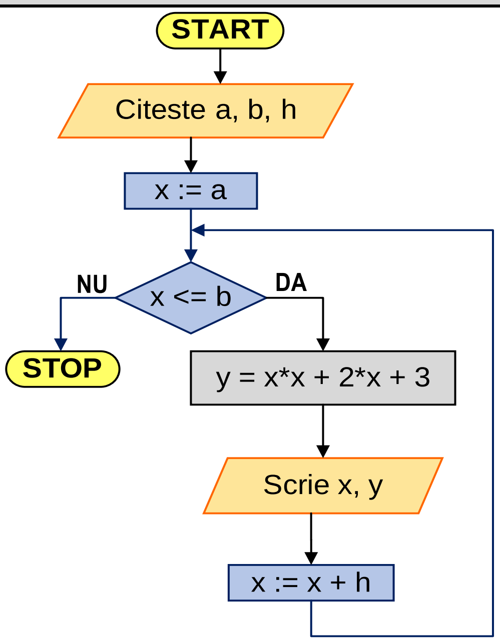
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a,b,h **Pentru** x←a,b,h y ← x*x +2*x+3 **Scrie** x,y **Sfârşit pentru** **Sfârşit**

Figura 4.5a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii pe un interval

- pentru atribuirea către variabila **x** a valorilor din intervalul **[a,b]**, parcurs cu pasul **h**, se utilizează un ciclu cu contor, astfel:
    1. se iniţializează valoarea variabilei **x** cu prima valoare din interval, adică cu **a**, deci **x := a**;
    2. se verifică dacă valoarea variabilei x este în intervalul **[a,b]**, adică se verifică condiţia **x<=b**. Dacă condiţia este adevărată, se atribuie lui y valoarea $y = x^2 + 2x + 3$ se afişează **x** şi **y** şi se trece la pasul 3.
    3. Se modifică valoarea contorului cu pasul h, adică: **x := x + h** şi se revine la pasul 2;
    4. dacă condiţia de la pasul 2 nu este adevărată, se părăseşte ciclul cu contor.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.5a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.5b.

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { float a,b,h,x,y; printf("\n Introdu a, a = "); scanf("%f",&a); printf("\n Introdu b, b = "); scanf("%f",&b); printf("\n Introdu h, h = "); scanf("%f",&h); for(x = a ; x <= b ; x = x+h) { y = x*x + 2*x + 3 ; printf("\n x = %6.3f \t y = %6.3f",x,y); } }</pre>	Introdu a, a = **-5** Introdu b, b = **5** Introdu h, h = **1** x = -5.000 y = 18.000 x = -4.000 y = 11.000 x = -3.000 y = 6.000 x = -2.000 y = 3.000 x = -1.000 y = 2.000 x = 0.000 y = 3.000 x = 1.000 y = 6.000 x = 2.000 y = 11.000 x = 3.000 y = 18.000 x = 4.000 y = 27.000 x = 5.000 y = 38.000

Figura 4.5b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii pe un interval

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

4.6. Calculul valorilor unei funcţii cu două ramuri într-un interval

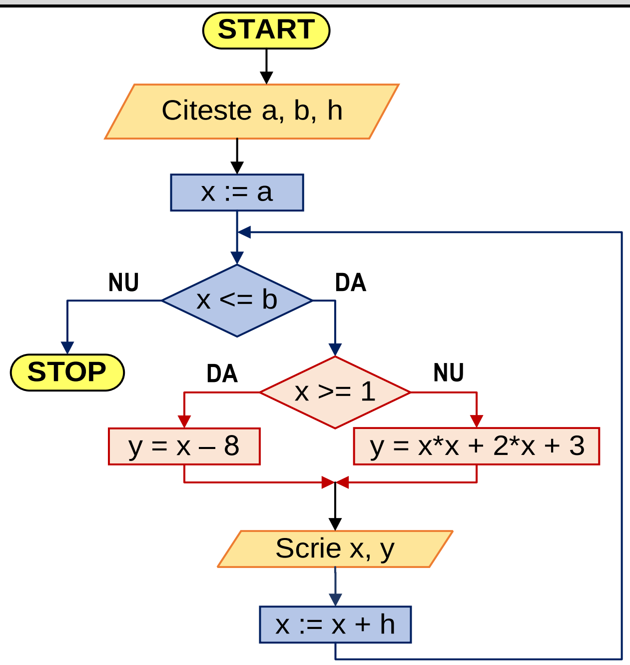
Calculul valorii funcţiei: $y = f(x) = \begin{cases} x - 8 & daca \quad x \geq 1 \\ x^2 + 2x + 3 & daca \quad x < 1 \end{cases}$ pentru $x \in [a,b]$, parcurs cu pasul h.

Algoritmul de calcul constă în:

- citirea limitelor intervalului în care variabila **x** ia valori (adică variabilele **a** şi **b**), respectiv a pasului cu care se parcurge intervalul (variabila **h**);
- pentru atribuirea către variabila **x** a valorilor din intervalul **[a,b]**, parcurs cu pasul **h**, se utilizează un ciclu cu contor, astfel:
    1. se iniţializează valoarea variabilei **x** cu prima valoare din interval, adică cu **a**, deci **x := a**;
    2. se verifică dacă valoarea variabilei **x** este în intervalul **[a,b]**, adică se verifică condiţia **x<=b**. Dacă condiţia este adevărată se trece la pasul următor, iar dacă condiţia este falsă se părăseşte ciclul, continuând cu secvenţa următoare ciclului;
    3. pentru calculul valorii lui **y** se verifică condiţia **x >= 1**. Dacă condiţia este adevărată **y** se calculează cu relaţia: $y = x - 8$, iar dacă condiţia este falsă **y** se calculează cu relaţia: $y = x^2 + 2x + 3$;
    4. se afişează **x** şi **y** şi se trece la pasul următor.
    5. se modifică valoarea contorului cu pasul h, adică: **x := x + h**, după care se revine la pasul 2.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.6a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.6b.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte a, b, h
   Pentru x ← a, b, h
         Dacă x ≥ 1 atunci
            y ← x - 8
         altfel
            y ← x*x + 2*x + 3
         Sfârşit dacă
         Scrie x, y
   Sfârşit pentru
Sfârşit
```

Figura 4.6a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu două ramuri într-un interval

Programul C

```c
#include<stdio.h>
int main(void)
{   float x,y,a,b,h;
    printf("\n Introdu a, a  = "); scanf("%f",&a);
    printf("\n Introdu b, b = "); scanf("%f",&b);
    printf("\n Introdu h, h = "); scanf("%f",&h);
    for( x = a ; x <= b ; x = x+h )
    {    if( x >= 1) y = x - 8;
         else y = x*x + 2*x + 3;
         printf("\n x = %f \t y = %f",x,y);
     }
}
```

Rularea programului

```
Introdu a, a = -3
Introdu b, b = 3
Introdu h, h = 1
x = -3.000        y =  6.000
x = -2.000        y =  3.000
x = -1.000        y =  2.000
x =  0.000        y =  3.000
x =  1.000        y = -7.000
x =  2.000        y = -6.000
x =  3.000        y = -5.000
```

Figura 4.6b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii cu două ramuri într-un interval

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

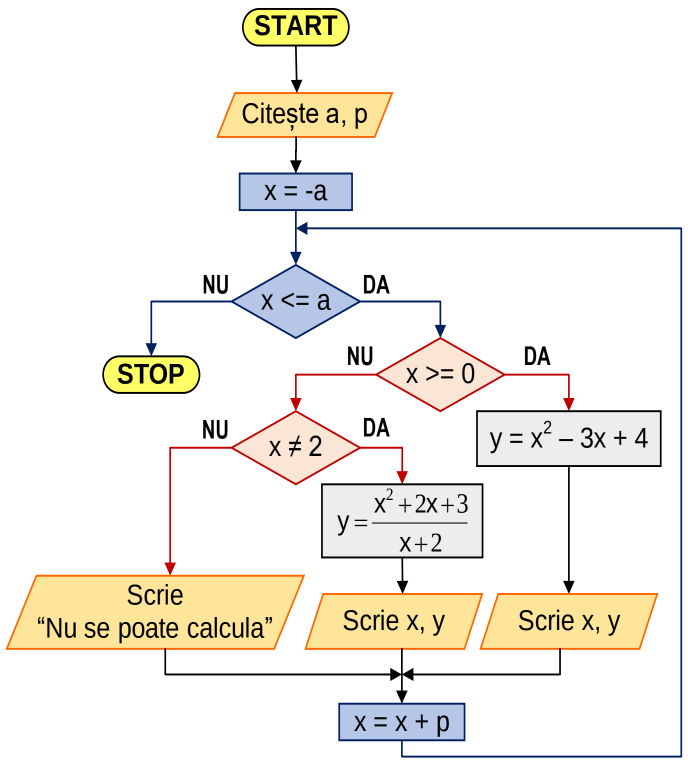
4.7. Calculul valorilor unei funcţii cu trei ramuri într-un interval – varianta 1

Se consideră funcţia: $y = f(x) = \begin{cases} x^2 - 3x + 4 & daca \;\; x \geq 0 \\ \frac{x^2+2x+3}{x+2} & daca \;\; x < 0 \end{cases}$, $x \in [-a, a]$, parcurs cu pasul *p*.

Algoritmul de calcul constă în:

- se citeşte variabila **a,** respectiv pasul cu care se parcurge intervalul (variabila **p**);
- pentru atribuirea către variabila **x** a valorilor din intervalul **[-a,a],** parcurs cu pasul **p**, se utilizează un ciclu cu contor, astfel:

1. se iniţializează valoarea variabilei **x** cu prima valoare din interval, adică cu **-a**, deci **x := -a**;
2. se verifică dacă valoarea variabilei **x** este în intervalul **[-a,a]**, adică se verifică condiţia **x<=a**. Dacă condiţia este adevărată se trece la pasul următor, iar dacă condiţia este falsă se părăseşte ciclul, continuând cu secvenţa următoare ciclului;

Schema logică	Limbajul pseudocod
	see below


```
Început
   Citeşte a, p
   Pentru x ← -a, a, p
      Dacă x≥0 atunci
         y ← x*x - 3*x + 4
         Scrie x, y
      altfel
         Dacă x ≠ 2 atunci
            y ← (x*x + 2*x + 3) / (x + 2)
            Scrie x, y
         altfel
            Scrie „Nu se poate calcula”
         Sfârşit dacă
      Sfârşit dacă
   Sfârşit pentru
Sfârşit
```

Figura 4.7a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri într-un interval – varianta 1

Programul C

```c
#include<stdio.h>
int main(void)
{  float a,p,x,y;
 printf("Introdu a si p:");   scanf("%f  %f",&a, &p);
 for( x = -a ; x <= a ; x = x + p )
 if( x >= 0 )
    {  y = x*x - 3*x + 4; printf("\n x=%6.3f y=%6.3f",x,y);   }
 else if( x != -2 )
         {  y = (x*x + 2*x + 3) / (x + 2);
            printf("\n x = %6.3f \t y = %6.3f",x,y);
         }
       else printf("\n Nu se poate calcula!");
}
```

Rularea programului

Introdu a si p: **3 1**
x = -3.000   y =   6.000
Nu se poate calcula!
x = -1.000   y =   2.000
x = 0.000   y =   4.000
x = 1.000   y =   2.000
x = 2.000   y =   2.000
x = 3.000   y =   4.000

Figura 4.7b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii cu trei ramuri într-un interval – varianta 1

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

3. pentru calculul valorii lui **y** se verifică condiţia condiţia **x ≥ 0**. Dacă condiţia este adevărată, adică **x ≥ 0**, **y** se calculează cu relaţia: $y = x^2 - 3x + 4$ şi se afişează valorile lui **x** şi a lui **y**; Dacă condiţia **x ≥ 0** este falsă, se observă că numitorul expresiei se anulează pentru **x = -2**, situaţie în care **y** nu se poate calcula. Astfel, pentru varianta în care **x < 0** trebuie să se impună o nouă condiţie, cea prin care se verifică dacă valoarea lui **x** este diferită de **-2**. Dacă această condiţie este adevărată atunci **y** se calculează cu relaţia: $y = \frac{x^2+2x+3}{x+2}$ şi se afişează valoarea lui **x** şi a lui **y**. Dacă a doua condiţie este falsă atunci nu se poate calcula **y** şi se afişează un mesaj corespunzător.

4. se modifică valoarea contorului cu pasul h, adică: **x := x + p**, după care se revine la pasul 2.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.7a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.7b.

4.8. Calculul valorilor unei funcţii cu trei ramuri într-un interval – varianta 2

Numerele din şirul lui Fibonacci sunt definite astfel: $F_n = \begin{cases} 0 & daca\ n = 0 \\ 1 & daca\ n = 1 \\ F_{n-1} + F_{n-2} & daca\ n > 1 \end{cases}$.

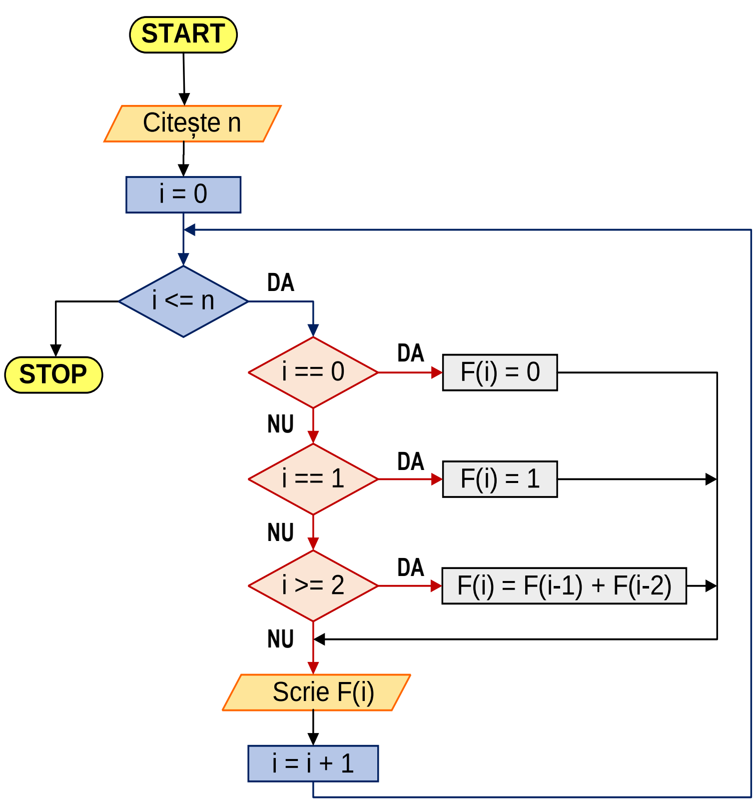
Primele 17 numere din şirul lui Fibonacci sunt: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987.

Prin împărţirea unui element (de la elementul 14 în sus) al şirului lui Fibonacci la precedentul său se obţine valoarea **1.61803** (233 : 144 = 1.61803**,** 377 : 233 = 1.61803, etc), denumită **numărul de aur**. Acesta se regăseşte în arhitectură, pictură, sculptură, estetică şi artă în general, la formarea unor proporţii armonioase, plăcute ochiului.

Algoritmul de determinare a numerelor din şirul lui Fibonacci este următorul:

- se citeşte numărul de elemente, **n**;
- se parcurge intervalul **[0,n]**, cu pasul **1**, pentru generarea celor **n** elemente, utilizând un ciclu cu contor, astfel:
  1. se iniţializează contorul **i**, cu valoarea iniţială, adică **i = 0**;
  2. se verifică dacă **i ≤ n**, dacă condiţia este adevărată se trece la pasul următor, iar dacă condiţia este falsă se părăseşte ciclul cu contor, continuând cu secvenţa următoare ciclului;

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  Pentru i ← 0, n
    Dacă i = 0 **atunci**
      F(i) ← 0
    Sfârşit dacă
    Dacă x = 1 **atunci**
      F(i) ← 1
    Sfârşit dacă
    Dacă x ≥ 2 **atunci**
      F(i) ← F(i-1) + F(i-2)
    Sfârşit dacă
    Scrie F(i)
  Sfârşit pentru
Sfârşit

Figura 4.8a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri într-un interval – varianta 2

3. se verifică dacă **i** este egal cu **0**, caz în care **F(0) = 0**;
4. se verifică dacă **i** este egal cu **1**, caz în care **F(1) = 1**;
5. se verifică dacă **i** este mai mare sau egal cu **2**, caz în care **F(i) = F(i-1) + F(i-2)**;
6. se modifică valoarea contorului **i**, astfel: **i = i +1** şi se revine la pasul 2.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 4.8a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 4.8b.

<table>
<tr><th>Programul C</th><th>Rularea programului</th></tr>
<tr><td>

```
#include<stdio.h>
int main(void)
{  int n, i, F[30];
   printf("\n Introdu n = ");   scanf("%d",&n);
   for( i = 0 ; i <= n ; i++ )
     { if( i == 0 ) F[i] = 0 ;
       if( i == 1) F[i] = 1;
       if( i >= 2 ) F[i] = F[i-1] + F[i-2] ;
         printf("\n F[%2d] = %6d",i,F[i]);
     }
}
```

</td><td>

```
Introdu n = 7
F[  0] =    0
F[  1] =    1
F[  2] =    1
F[  3] =    2
F[  4] =    3
F[  5] =    5
F[  6] =    8
F[  7] =   13
```

</td></tr>
</table>

Figura 4.8b. Programul C şi rularea acestuia pentru calculul valorilor unei funcţii cu trei ramuri într-un interval– varianta 2

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Capitolul 5. Operaţii cu tablouri unidimensionale

Tablourile reprezintă colecţii de date de acelaşi tip grupate sub un nume comun ale căror elemente sunt identificate prin indici. Elementele tablourilor sunt aşezate în memorie într-o zonă contiguă, unul după altul.

Tablourile unidimensionale sunt denumite în mod greşit „vectori”, denumirea corectă fiind cea de „mulţime”.

Elementele tablourilor se identifică prin indici, care sunt numere întregi, pozitive, indicele primului element al tabloului fiind **0**.

Aplicarea unei operaţii (citire, afişare, adunare, înmulţire, etc) fiecărui element al unui tablou unidimensional se realizează de obicei utilizând cicluri cu contor. În exemplele următoare sunt prezentate o serie de algoritmi utilizaţi în rezolvarea unor probleme ce implică tablouri unidimensionale sau probleme a căror rezolvare conduce la utilizarea tablourilor unidimensionale.

5.1. Introducerea / afişarea elementelor unui tablou unidimensional

În acest exemplu sunt prezentate operaţiile de introducere (citire), respectiv afişare (scriere) a elementelor unui tablou unidimensional cu „ **n** ” elemente, numărul acestora fiind introdus de la tastatură.

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se utilizează un ciclu cu contor pentru citirea valorilor elementelor tabloului unidimensional, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente ale tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunea care reprezintă corpul ciclului cu contor, în acest caz se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** $\mathbf{a_i}$;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- se utilizează un ciclu cu contor pentru afişarea valorilor elementelor tabloului unidimensional, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

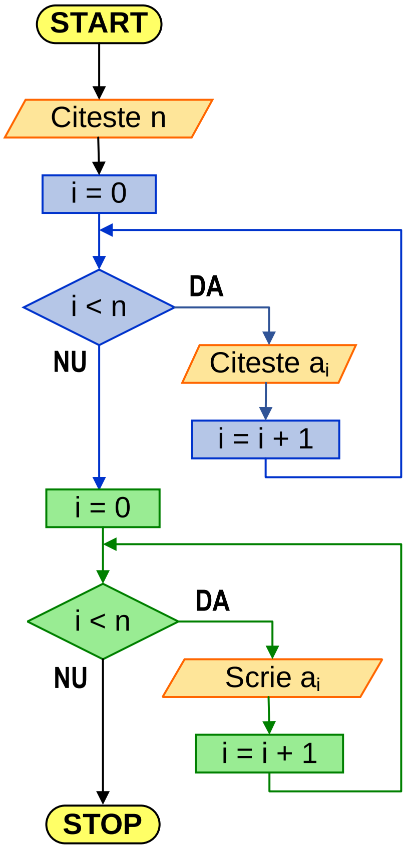
3. se execută instrucţiunea care reprezintă corpul ciclului cu contor, în acest caz se execută operaţia de afişare a valorii elementului curent al tabloului, adică **Scrie** $\mathbf{a_i}$;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**.

Observaţie: Cu ajutorul primului ciclu cu contor se realizează citirea elementelor tabloului unidimensional, iar prin utilizarea celui de-al doilea ciclu cu contor se realizează afişarea elementelor tabloului unidimensional.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.1a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.1b.

Schema logică



Limbajul pseudocod

Început
  Citeşte n
  Pentru i ← 0, n - 1
    Citeşte a_i
  Sfârşit pentru
  Pentru i ← 0, n - 1
    Scrie a_i
  Sfârşit pentru
Sfârşit

Figura 5.1a. Reprezentarea algoritmului pentru introducerea/afişarea elementelor unui tablou unidimensional

Programul C

```c
#include<stdio.h>
int main(void)
{ int i, n, a[20];
  printf("\n Introduceti n, n = ");
  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
   { printf(" a[%d] = ",i);  scanf("%d",&a[i]);
   }
  for( i = 0 ; i < n ; i++ )
   printf("%4d",a[i]);
}
```

Rularea programului

Introduceti n, n = **6**
a[0] = **2**
a[1] = **4**
a[2] = **5**
a[3] = **3**
a[4] = **7**
a[5] = **1**
  2   4   5   3   7   1

Figura 5.1b. Programul C şi rularea acestuia pentru introducerea/afişarea elementelor unui tablou unidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.2. Calculul sumei elementelor unui tablou unidimensional

5.2.1. Calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu contor

Se consideră următoarea problemă: un strungar execută pe parcursul a **n** - zile, aceeaşi piesă dar în cantităţi diferite de la o zi la alta. Se doreşte să se determine numărul total de piese pe care le-a realizat strungarul în cele **n** zile. Problema constă în calculul sumei elementelor unei mulţimi de valori, adică în programare, calculul sumei elementelor unui tablou unidimensional.

Pentru calculul sumei, se iniţializează suma cu valoarea **0** (zero) şi se repetă pentru fiecare element al tabloului unidimensional două operaţii: citirea valorii elementului curent al tabloului şi adăugarea acestuia la sumă utilizând o relaţie de forma:

$$S := S + a_i$$

Relaţia de mai sus trebuie interpretată astfel: valoarea nouă a sumei este egală cu valoarea (sau primeşte valoarea) veche a sumei la care se adaugă valoarea elementului curent al tabloului unidimensional.

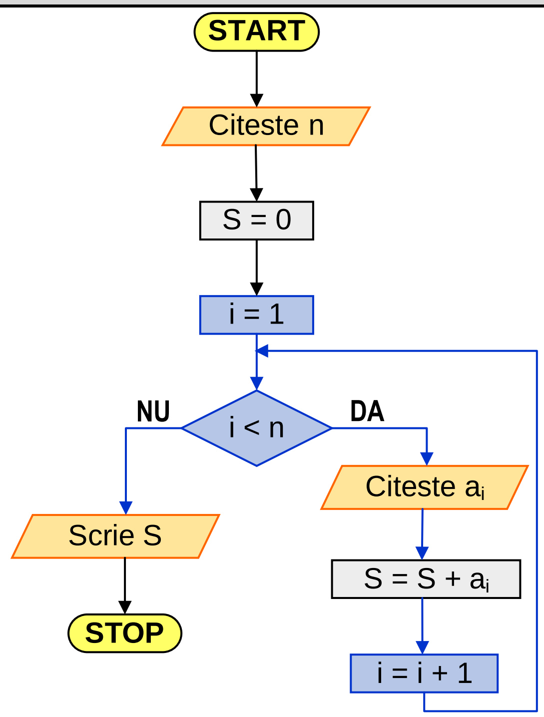
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n $S \leftarrow 0$ **Pentru** $i \leftarrow 1, n - 1$ **Citeşte** a_i $S \leftarrow S + a_i$ **Sfârşit pentru** **Scrie** S **Sfârşit**

Figura 5.2a. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional utilizând un ciclu cu contor

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma cu valoarea **0** (zero – element neutru pentru operaţia de adunare);
- se utilizează un ciclu cu contor în cadrul căruia se va realiza citirea valorilor elementelor tabloului unidimensional şi adăugarea acestei valori sumei. În cadrul ciclului cu contor se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente ale tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor, adică:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** $\mathbf{a_i}$;
- se adaugă valoarea elementului curent al tabloului la sumă, utilizând relaţia: $\mathbf{S := S + a_i}$;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**.

- după ieşirea din ciclu cu contor (adică s-au citit cele **n** elemente ale tabloului şi s-au adunat la sumă) se afişează valoarea calculată a sumei, adică **Scrie S**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.2a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.2b.

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { int i, n, a[20], S; printf("\n Introduceti n, n = "); scanf("%d",&n); S = 0; for(i = 0 ; i < n ; i++) { printf(" Ziua %d = ",i+1); scanf("%d",&a[i]); S = S + a[i] ; } printf("Suma elementelor este S = %d",S); }</pre>	Introduceti n, n = **5** Ziua 1 = **23** Ziua 2 = **21** Ziua 3 = **24** Ziua 4 = **22** Ziua 5 = **23** Suma elementelor este S = 113

Figura 5.2b. Programul C şi rularea acestuia pentru calculul sumei elementelor unui tablou unidimensional utilizând un ciclu cu contor

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.2.2. Calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test iniţial

În general, prelucrarea elementelor tablourilor unidimensionale se realizează utilizând cicluri cu contor, însă se pot utiliza şi celelalte variante de cicluri. În acest exemplu este prezentat calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test iniţial. Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma cu **0** (zero – element neutru pentru operaţia de adunare);
- se atribuie variabilei „ **i** ” valoarea **0**;
- se utilizează un ciclu cu test iniţial în corpul căruia se vor realiza următoarele operaţii: citirea valorilor elementelor tabloului unidimensional şi adăugarea acestora la sumă. Pentru aceasta trebuie parcurşi următorii paşi:

1. se evaluează condiţia „**i < n** ” prin care se verifică dacă valoarea curentă a lui „ **i** ” este mai mică decât **n**. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 2). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

2. se execută instrucţiunile care reprezintă corpul ciclului cu contor, adică:

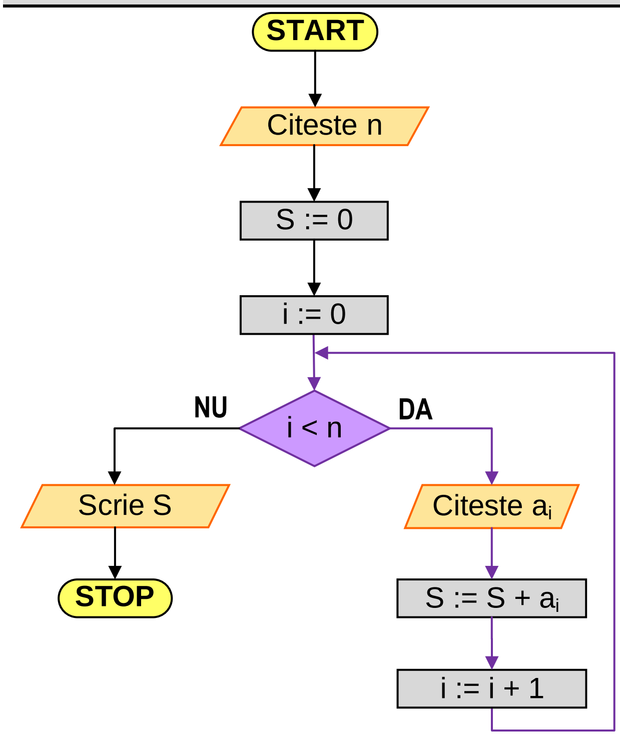
- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** $\mathbf{a_i}$;
- se adaugă valoarea elementului curent al tabloului la sumă, utilizând relaţia: $\mathbf{S := S + a_i}$;

3. se execută instrucţiunea prin care se modifică valoarea variabilei **i**, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **1**, adică la verificarea condiţiei **i < n**;

- după ieşirea din ciclu cu test iniţial (adică s-au citit cele **n** elemente ale tabloului şi s-au adunat la sumă) se afişează valoarea calculată a sumei, adică **Scrie S**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.2c, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.2d.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  S ← 0
  i ← 0
  Cât timp i < n **execută**
    Citeşte a_i
    S ← S + a_i
    i ← i + 1
  Sfârşit cât timp
  Scrie S
Sfârşit

Figura 5.2c. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test iniţial

Programul C

```
#include<stdio.h>
int main(void)
{ int i, n, S, a[20];
  printf("\n Introduceti n, n = ");  scanf("%d",&n);
  S = 0; i = 0;
  while( i < n )
   { printf(" a[%d] = ",i);  scanf("%d",&a[i]);
     S = S + a[i]; i = i + 1;    }
  printf(" Suma elementelor este S = %d",S);
}
```

Rularea programului

Introduceti n, n = **5**
a[0] = **2**
a[1] = **4**
a[2] = **5**
a[3] = **7**
a[4] = **3**
Suma elementelor este S = 21

Figura 5.2b. Programul C şi rularea acestuia pentru calculul sumei elementelor unui tablou unidimensional utilizând un ciclu cu test iniţial

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.2.3. Calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test final

În acest exemplu este prezentat calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test final. Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma cu **0** (zero – element neutru pentru operaţia de adunare), se atribuie variabilei „ **i** ” valoarea **0**;
- se utilizează un ciclu cu test final în corpul căruia se vor realiza următoarele operaţii: citirea valorilor elementelor tabloului unidimensional şi adăugarea acestora la sumă. Pentru aceasta trebuie parcurşi următorii paşi:

1. se execută instrucţiunile care reprezintă corpul ciclului cu contor, adică:
- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i**;
- se adaugă valoarea elementului curent al tabloului la sumă, utilizând relaţia: **$S := S + a_i$**.

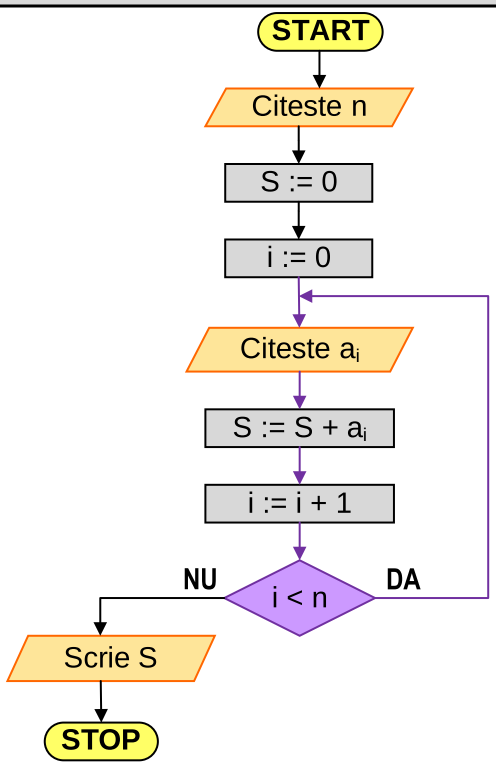
2. se execută instrucţiunea prin care se modifică valoarea variabilei **i**, adică **i = i + 1**;

3. se evaluează condiţia „**i < n** ” prin care se verifică dacă valoarea curentă a lui **i** este mai mică decât **n**. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, adică se revine la pasul **1** pentru executarea instrucţiunilor care formează corpul ciclului. Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

- după ieşirea din ciclul cu test iniţial (adică s-au citit cele **n** elemente ale tabloului şi s-au adunat la sumă) se afişează valoarea calculată a sumei, adică **Scrie S**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.2e, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.2f.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte n
   S ← 0
   i ← 1
   Execută
      Citeşte ai
      S ← S + ai
       i ← i + 1
   Cât timp i < n
   Scrie S
Sfârşit
```

Figura 5.2e. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test final

Programul C

```
#include<stdio.h>
int main(void)
{ int i, n, S, a[20];
  printf("\n Introduceti n, n = ");   scanf("%d",&n);
  S = 0;  i = 0;
  do {   printf(" a[%d] = ",i); scanf("%d",&a[i]);
         S = S + a[i];  i = i + 1;
     } while( i < n );
  printf(" Suma elementelor este S = %d",S);
  }
```

Rularea programului

```
Introduceti n, n = 5
 a[0] = 2
 a[1] = 4
 a[2] = 5
 a[3] = 7
 a[4] = 3
 Suma elementelor este S = 21
```

Figura 5.2f. Programul C şi rularea acestuia pentru calculul sumei elementelor unui tablou unidimensional utilizând un ciclu cu test final

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.3. Calculul sumei elementelor strict pozitive ale unui tablou unidimensional

Se consideră următoarea problemă tehnică: din sistemul de preambalare a unui produs care se livrează la vrac rezultă cutii cu produse a căror greutate variază faţă de greutatea nominală. Se constată că unele cutii au abatere pozitivă, altele abatere negativă, iar unele cutii au exact greutatea nominală. Se măsoară greutatea fiecărei cutii şi din valoarea obţinută se scade greutatea nominală, obţinându-se un şir de valori care pot fi pozitive, negative sau nule (tabelul 5.3.1). Se doreşte să se determine cantitatea totală cu care cutiile cu abatere pozitivă depăşesc greutatea nominală.

Problema se reduce la calculul sumei elementelor strict pozitive din şirul de valori.

Tabelul 5.3.1. Şirul de valori

Element	1	2	3	4	5	6	7	8	9	10
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valoare	**15**	**-12**	**0**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**

Algoritmul este următorul:
- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma elementelor cu **0** (zero – element neutru pentru adunare);
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** a_i;

- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este strict pozitiv, impunând condiţia ca „ $a_i > 0$ ". Dacă condiţia este adevărată înseamnă că elementul curent este strict pozitiv, astfel că pe ramura „ **DA** ”, se adaugă elementul curent la sumă. Ramura „ **NU** ” corespunde valorilor negative sau nule ale elementelor tabloului, astfel că pe aceasta nu se execută nimic.

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt strict pozitive, iar cele strict pozitive s-au adăugat la sumă) se afişează valoarea calculată a sumei, adică **Scrie S**.

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.3.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Suma primeşte iniţial valoarea **0** (zero).

La început (**i = 0**) se citeşte valoarea primului element al tabloului: a_0 **= 15**, şi se verifică dacă este strict pozitiv. Primul element fiind strict pozitiv, se adună la sumă, valoarea nouă a sumei fiind obţinută prin adunarea valorii anterioare a sumei (adică **0**) cu valoarea elementului curent (a_0**= 15**), astfel: **S =** 0 + 15 = **15**;

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (a_1 **= -12**) şi se verifică dacă este strict pozitiv.

Tabelul 5.3.2. Calculul sumei

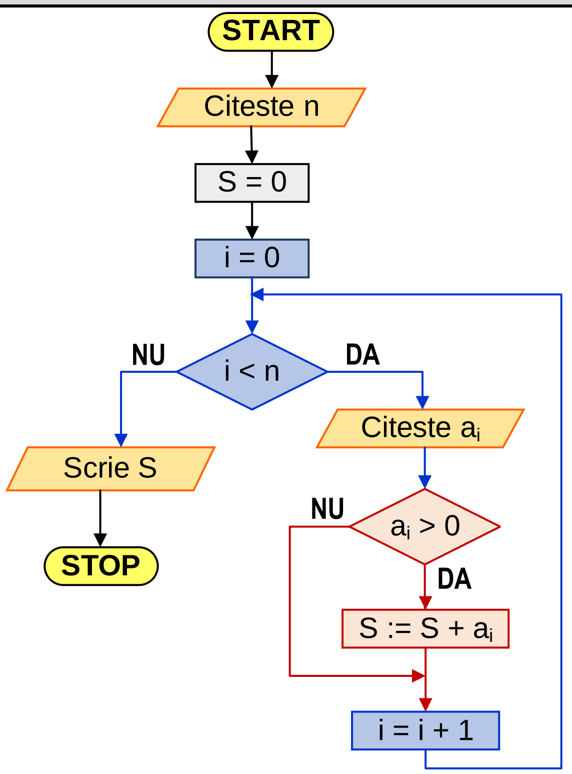
Valori obţinute pentru: **n = 10**				
i	a_i	$a_i > 0$	**S = 0**	**Ecran**
0	15	**DA**	S:= 0 + **15** = 15	
1	-12	NU		
2	0	NU		
3	8	**DA**	S:= 15 + **8** = 23	
4	14	**DA**	S:= 23 + **14** = 37	
5	-9	NU		
6	-15	NU		
7	0	NU		
8	10	**DA**	S:= 37 + **10** = 47	
9	-20	NU		**47**

Deoarece valoarea acestuia nu este strict pozitivă, se continuă parcurgerea pe ramura **NU** a instrucţiunii de selecţie, în continuare mărindu-se valoarea contorului **i** cu **1** (**i = 2**).

Procedeul se repetă până când valoarea lui **i** devine **n**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea sumei **S** (**Scrie S**).

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.3a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.3b.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte n
   S ← 0
   Pentru i ← 0, n-1
       Citeşte ai
       Dacă ai > 0 atunci
           S ← S + ai
       Sfârşit dacă
   Sfârşit pentru
   Scrie S
Sfârşit
```

Figura 5.3a. Reprezentarea algoritmului pentru calculul sumei elementelor strict pozitive ale unui tablou unidimensional

Programul C

```c
#include<stdio.h>
int main(void)
{ int i, n, a[20], S;
  printf("\n Introduceti n, n = ");   scanf("%d",&n);
  S = 0;
  for( i = 0 ; i < n ; i++ )
   { printf(" a[%d] = ",i); scanf("%d",&a[i]);
     if( a[i] > 0 )
         S = S + a[i];
   }
  printf("Suma elementelor este S = %d",S);
}
```

Rularea programului

Introduceti n, n = **10**
a[0] = **15**
a[1] = **-12**
a[2] = **0**
a[3] = **8**
a[4] = **14**
a[5] = **-9**
a[6] = **-15**
a[7] = **0**
a[8] = **10**
a[9] = **-20**
Suma elementelor este S = 47

Figura 5.3b. Programul C şi rularea acestuia pentru calculul sumei elementelor strict pozitive unui tablou unidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.4. Produsul elementelor strict pozitive ale unei mulţimi

Se consideră şirul de valori din tabelul 5.4.1. Se doreşte calculul produsului elementelor strict pozitive din şir.

Tabelul 5.4.1. Şirul de valori

Element	1	2	3	4	5	6	7	8
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Valoare	**-5**	**2**	**0**	**1**	**-3**	**4**	**-2**	**3**

Algoritmul este următorul:
- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează produsul elementelor cu **1, P = 1** (unu – element neutru pentru înmulţire);
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** a_i;

- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este strict pozitiv, impunând condiţia ca „ $a_i > 0$ ". Dacă condiţia este adevărată înseamnă că elementul curent este strict pozitiv, se continuă pe ramura „ **DA** ”, se înmulţeşte la produs valoarea elementului curent. Ramura „ **NU** ” corespunde valorilor negative sau nule ale elementelor tabloului, în consecinţă pe această ramură nu se execută nimic;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;
- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt strict pozitive, iar cele strict pozitive s-au înmulţit la produs) se afişează valoarea calculată a produsului, adică **Scrie P**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.4.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Produsul primeşte iniţial valoarea **1** (unu).

La început (**i = 0**) se citeşte valoarea primului element al tabloului: $a_0 = -5$, şi se verifică dacă este strict pozitiv. Primul element nefiind strict pozitiv, nu se verifică condiţia $a_i > 0$, deci se trece la modificarea valorii contorului;

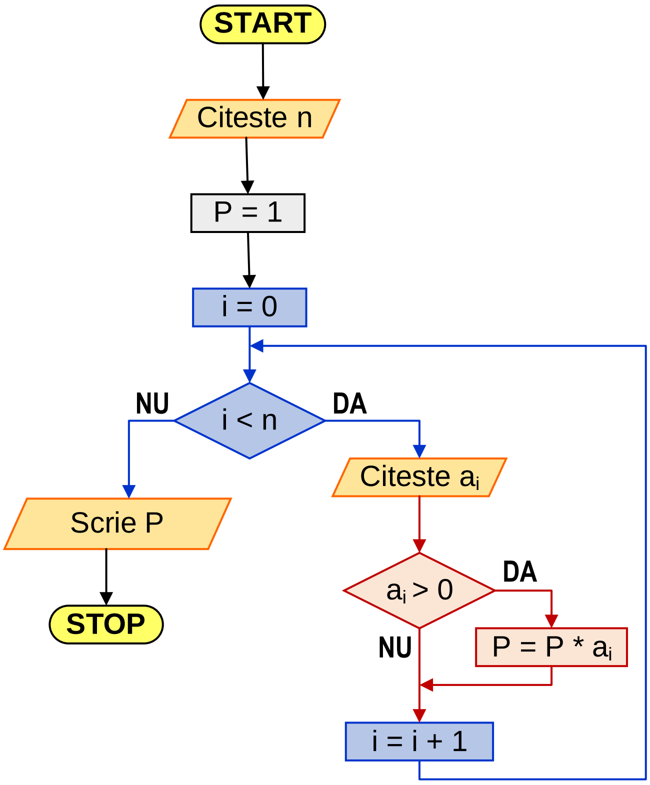
În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional ($a_1 = 2$) şi se verifică dacă este strict pozitiv. Deoarece valoarea acestuia este strict pozitivă, se continuă parcurgerea pe ramura **DA** a instrucţiunii de decizie, astfel că se modifică valoarea produsului prin înmulţirea acestuia cu valoarea elementului curent astfel: $P = P * a_i$, apoi creşte valoarea contorului **i** cu **1** (**i = 2**).

Tabelul 5.4.2. Calculul produsului

Valori obţinute pentru: **n = 8**				
i	a_i	$a_i > 0$	**P**	**Ecran**
			1	
0	-5	NU		
1	2	**DA**	P:= 1 * **2** = 2	
2	0	NU		
3	1	**DA**	P:= 2 * **1** = 2	
4	-3	NU		
5	4	**DA**	P:= 2 * **4** = 8	
6	-2	NU		
7	3	**DA**	P:= 8 * **3** = 24	**24**

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n-1**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea produsului **P**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.4a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.4b.

Schema logică	Limbajul pseudocod
	see below

Început
  Citeşte n
  P ← 1
  Pentru i ← 0, n-1
    Citeşte a_i
    Dacă a_i > 0 **atunci**
      P ← P * a_i
    Sfârşit dacă
  Sfârşit pentru
  Scrie P
Sfârşit

Figura 5.4a. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive ale unui tablou unidimensional

Programul C	Rularea programului
(see code below)	(see output below)

```
#include<stdio.h>
int main(void)
{  int n, i, P, a[15];
   printf("\n Introdu n, n = "); scanf("%d",&n);
   P = 1;
   for( i = 0 ; i < n ; i++ )
    { printf("\n a[%d]=",i); scanf("%d",&a[i]);
       if( a[i] > 0 )
          P = P * a[i];
    }
   printf("\n P = %d",P);
}
```

Introdu n, n = **8**
a[0] = **-5**
a[1] = **2**
a[2] = **0**
a[3] = **1**
a[4] = **-3**
a[5] = **4**
a[6] = **-2**
a[7] = **3**
P = 24

Figura 5.4b. Programul C şi rularea acestuia pentru calculul produsului elementelor strict pozitive unui tablou unidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.5. Numărul de elemente nule ale unei mulţimi

Se consideră şirul de valori din tabelul 5.5.1. Se doreşte calculul numărului de elemente nule din şir.

Tabelul 5.5.1. Şirul de valori

Element	1	2	3	4	5	6	7	8
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Valoare	**-5**	**2**	**0**	**1**	**0**	**4**	**-2**	**3**

Algoritmul este următorul:
- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează numărul elementelor nule (notat cu **NEN**) cu **0, NEN = 0** (zero – element neutru pentru adunare);
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i**;

- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este nul. Dacă condiţia este adevărată, se continuă pe ramura „ **DA** ”, se incrementează variabila **NEN**. Ramura „ **NU** ” corespunde valorilor negative sau pozitive ale elementelor tabloului, în consecinţă pe această ramură nu se execută nimic;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt nule, s-au numărat elementele nule) se afişează valoarea variabilei **NEN**, adică **Scrie NEN**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.5.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Variabila **NEN** primeşte iniţial valoarea **0** (zero).

La început (**i = 0**) se citeşte valoarea primului element al tabloului: **a_0 = -5**, şi se verifică dacă este nul. Primul element nefiind nul, nu se verifică condiţia, deci se trece la modificarea valorii contorului;

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (**a_1 = 2**) şi se verifică dacă este nul. Deoarece valoarea acestuia nu este nulă, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 2.** Al treilea element este nul, astfel că se continuă parcurgerea pe ramura **DA** a instrucţiunii de decizie, se modifică valoarea variabilei **NEN**, incrementând-se, apoi creşte valoarea contorului **i** cu **1** (**i = 3**). Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n - 1**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea variabilei **NEN**.

Tabelul 5.5.2. Calculul numărului de elemente

Valori obţinute pentru: **n = 8**				
i	**a_i**	**a_i nul**	**NEN**	**Ecran**
			0	
0	-5	NU		
1	2	NU		
2	0	**DA**	NEN := 0 + **1** = 1	
3	1	NU		
4	0	**DA**	NEN := 0 + **1** = 2	
5	4	NU		
6	-2	NU		
7	3	NU		**2**

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.5a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.5b.

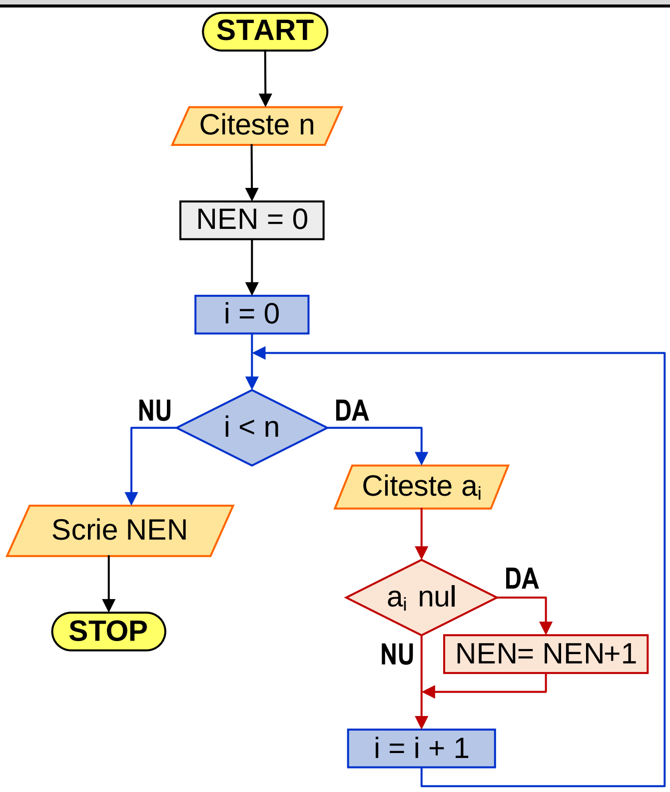
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n NEN ← 0 **Pentru** i ← 0, n - 1 **Citeşte** a_i **Dacă** a_i = 0 **atunci** NEN ← NEN + 1 **Sfârşit dacă** **Sfârşit pentru** **Scrie** NEN **Sfârşit**

Figura 5.5a. Reprezentarea algoritmului pentru calculul numărului de elemente nule ale unui tablou unidimensional

Programul C

```
#include<stdio.h>
int main(void)
{  int n, i, NEN, a[15];
   printf("\n Introdu n, n = "); scanf("%d",&n);
   NEN = 0;
   for( i = 0 ; i < n ; i++ )
    { printf("\n a[%d]=",i);  scanf("%d",&a[i]);
         if( a[i] == 0 )
            NEN++;
    }
   printf("\n NEN = %d",NEN);
}
```

Rularea programului

n = **8**
a[1] = **-5**
a[2] = **2**
a[3] = **0**
a[4] = **1**
a[5] = **0**
a[6] = **4**
a[7] = **-2**
a[8] = **3**
NEN = 2

Figura 5.5b. Programul C şi rularea acestuia pentru calculul numărului de elemente nule ale unui tablou unidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.6. Media aritmetică a elementelor strict pozitive ale unei mulţimi

Se consideră şirul de valori din tabelul 5.6.1. Se doreşte determinarea mediei aritmetice a elementelor strict pozitive ale şirului. Pentru calculul mediei aritmetice trebuie să se calculeze suma elementelor strict pozitive precum şi numărul elementelor strict pozitive, media aritmetică fiind raportul dintre suma şi numărul elementelor strict pozitive.

Tabelul 5.6.1. Şirul de valori

Element	1	2	3	4	5	6	7	8	9	10
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valoare	**15**	**-12**	**0**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma elementelor strict pozitive şi numărul de elemente strict pozitive cu **0** (zero – element neutru pentru adunare), adică **SP = 0**, respectiv **NP = 0**;
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente al tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i**;

- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este strict pozitiv, impunând condiţia ca „ **a_i > 0** ". Dacă condiţia este adevărată înseamnă că elementul curent este strict pozitiv, se continuă astfel pe ramura „**DA**”, se adaugă elementul curent la sumă şi se incrementează numărul de elemente strict pozitive. Ramura „**NU**” corespunde valorilor negative sau nule ale elementelor tabloului, astfel că pe aceasta nu se execută nimic;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt strict pozitive, iar cele strict pozitive s-au adăugat la sumă) se calculează valoarea mediei aritmetice: **Ma = SP / NP**;
- se afişează valoarea calculată a mediei aritmetice, adică **Scrie Ma**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.6.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Suma şi numărul de elemente strict pozitve primesc iniţial valoarea **0** (zero), **SP = 0**, **NP = 0**.

La început (**i = 0**) se citeşte valoarea primului element al tabloului: **a_0 = 15**, şi se verifică dacă este strict pozitiv. Primul element fiind strict pozitiv, se adună la sumă, valoarea nouă a sumei fiind obţinută prin adunarea valorii anterioare a sumei (adică **0**) cu valoarea elementului curent (**a_0 = 15**), astfel: **SP** = 0 + 15 = **15**. De asemenea, numărul de elemente strict pozitive **NP** se incrementează, adică **NP = NP + 1**;

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (**a_1 = -12**) şi se verifică dacă este strict pozitiv. Deoarece valoarea acestuia nu

Tabelul 5.6.2. Calculul mediei aritmetice

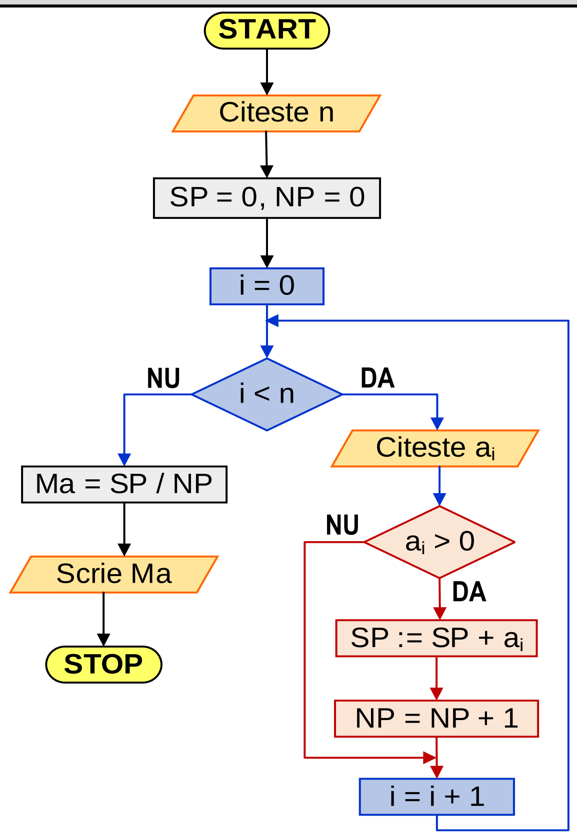
Valori obţinute pentru: **n = 10**					
i	**a_i**	**a_i > 0**	**SP**	**NP**	**Ecran**
			0	0	
0	15	**DA**	SP:= 0 + **15** = 15	1	
1	-12	NU			
2	0	NU			
3	8	**DA**	SP:= 15 + **8** = 23	2	
4	14	**DA**	SP:= 23 + **14** = 37	3	
5	-9	NU			
6	-15	NU			
7	0	NU			
8	10	**DA**	SP:= 37 + **10** = 47	4	
9	-20	NU			
			47	**4**	**11.750**

este strict pozitivă, se continuă parcurgerea pe ramura **NU** a instrucţiunii de selecţie, în continuare mărindu-se valoarea contorului **i** cu **1** (**i = 2**).

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică calculul mediei aritmetice şi afişarea valorii acesteia.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.6a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.6b.

Schema logică



Limbajul pseudocod

Început
  Citeşte n
  SP ← 0
  NP ← 0
  Pentru i ← 0, n - 1
    Citeşte a_i
    Dacă a_i>0 **atunci**
      SP ← SP + a_i
      NP ← NP + 1
    Sfârşit dacă
  Sfârşit pentru
  Ma ← SP/NP
  Scrie Ma
Sfârşit

Figura 5.6a. Reprezentarea algoritmului pentru calculul mediei aritmetice a elementelor strict pozitive ale unui tablou unidimensional

Programul C

```
#include<stdio.h>
int main(void)
{ int i, n, a[20], SP, NP; float Ma;
  printf("\n Introduceti n, n = ");  scanf("%d",&n);
  SP = 0; NP = 0;
  for( i = 0 ; i < n ; i++ )
   { printf(" a[%d] = ",i); scanf("%d",&a[i]);
     if( a[i] > 0 )
       { SP = SP + a[i]; NP++; }
   }
  Ma = SP*1.0 / NP;
  printf("Media aritmetica este Ma = %6.3f",Ma);
}
```

Rularea programului

Introduceti n, n = **10**
a[0] = **15**
a[1] = **-12**
a[2] = **0**
a[3] = **8**
a[4] = **14**
a[5] = **-9**
a[6] = **-15**
a[7] = **0**
a[8] = **10**
a[9] = **-20**
Media aritmetica este S = 11.750

Figura 5.6b. Programul C şi rularea acestuia pentru calculul mediei aritmetice a elementelor strict pozitive ale unui tablou unidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.7. Determinarea valorii şi poziţiei elementului maxim dintr-o mulţime

Se consideră şirul de valori din tabelul 5.7.1. Se doreşte determinarea elementului maxim şi a poziţiei sale din şir.

Tabelul 5.7.1. Şirul de valori

Element	1	2	3	4	5	6	7	8
Indice	0	1	2	3	4	5	6	7
Valoare	-5	2	0	1	-3	4	-2	3

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se utilizează un ciclu cu contor pentru citirea valorilor elementelor tabloului unidimensional, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „**DA**”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „**NU**”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunea care reprezintă corpul ciclului cu contor, în acest caz se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** a_i ;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- se iniţializează variabila **max** cu valoarea primului element din şir (**max =** a_0), iar variabila **pmax** se iniţializează cu **0**;
- pentru determinarea elementului maxim şi a poziţiei acestuia se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **1**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente al tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „**DA**”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „**NU**” se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după ciclul cu contor;

3. utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este mai mare decât **max**, impunând condiţia ca „ a_i **> max** ". Dacă condiţia este adevărată înseamnă că elementul curent este mai mare decât **max**, se continuă pe ramura „**DA**”, se atribuie variabilei **max** valoarea a_i a elementului curent, iar variabilei **pmax** i se atribuie valoarea indicelui **i** al elementului curent. Dacă elementul curent nu este mai mare decât variabila **max** (adică condiţia este falsă) se trece la pasul **4**;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat elementele şirului în raport cu variabila max) se afişează valorile variabilelor **max** şi **pmax**, adică **Scrie max, pmax**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.7.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Variabila **max** primeşte iniţial valoarea a_0, respectiv variabila **pmax** primeşte valoarea **0,** adică **max = -5**, **pmax = 0**.

În prima etapă (**i = 1**), se citeşte valoarea celui de al doilea element al tabloului: a_1 **= 2**, şi se verifică dacă este mai mare decât **max**. Deoarece condiţia este adevărată, variabila **max** primeşte valoarea elementului curent al tabloului, adică **max = 2**, iar **pmax** primeşte valoarea indicelui elementului curent, adică **pmax = 1**;

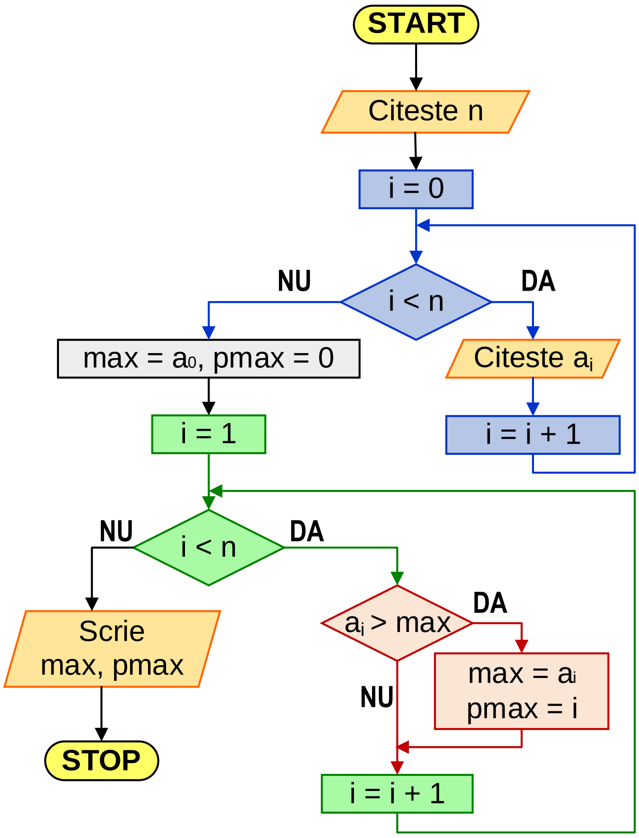
În continuare, contorul „**i**” îşi creşte valoarea cu **1**, astfel că **i = 2**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (**a_2 = 0**) şi se verifică dacă este mai mare decât **max**. Deoarece valoarea acestuia este mai mică, se continuă parcurgerea pe ramura „**NU**” a instrucţiunii de decizie, astfel că se continuă cu modificarea valorii contorului „**i**” cu **1** (**i = 3**).

Procedeul se repetă până când valoarea lui „**i**” devine mai mare decât **n - 1**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea valorilor variabilelor **max**, respectiv **pmax**.

Tabelul 5.7.2. Determinarea elementului maxim şi a poziţiei acestuia în şir

Valori obţinute pentru: **n = 8**					
i	**a_i**	**a_i > 0**	**max**	**pmax**	**Ecran**
			-5	**0**	
1	2	**DA**	2	1	
2	0	NU			
3	1	NU			
4	-3	NU			
5	4	**DA**	4	5	
6	-2	NU			
7	3	NU			
					4, 5

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.7a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.7b.

Schema logică	Limbajul pseudocod
	see below

Început
    Citeşte n
    Pentru i ← 0, n-1
        Citeşte a_i
    Sfârşit pentru
    max ← a_0
    pmax← 0
    Pentru i ← 1, n-1
        Dacă a_i > max **atunci**
            max ← a_i
            pmax ← i
        Sfârşit dacă
    Sfârşit pentru
    Scrie max, pmax
Sfârşit

Figura 5.7a. Reprezentarea algoritmului pentru determinarea valorii şi poziţiei elementului maxim dintr-o mulţime

Programul C	Rularea programului
`#include<stdio.h>` `int main(void)` `{ int n, i, max, pmax, a[15];` `  printf("\n Introdu n:");  scanf("%d",&n);` `  for( i = 0 ; i < n ; i++ )` `     { printf("\n a[%d]=",i);  scanf("%d",&a[i]); }` `  max = a[0];` `  pmax = 0;` `  for( i = 1 ; i < n ; i++ )` `     if( a[i] > max )` `         { max = a[i];` `           pmax = i;` `         }` `   printf("\n Max: a[%d] = %d",pmax, max);` `}`	 n = **8** a[0] = **-5** a[1] = **2** a[2] = **0** a[3] = **1** a[4] = **-3** a[5] = **4** a[6] = **-2** a[7] = **3** Max: a[5] = 4

Figura 5.7b. Programul C şi rularea acestuia pentru determinarea valorii şi poziţiei elementului maxim dintr-o mulţime

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.8. Ordonarea crescătoare / descrescătoare a elementelor unei mulţimi

Operaţia de ordonare a unor articole, în funcţie de diverse criterii, este foarte des întâlnită în practică, fiind dezvoltaţi o mulţime de algoritmi de sortare.

În informatică, operaţia de sortare este o operaţie fundamentală, care constă în rearanjarea elementelor unei mulţimi în ordine crescătoare sau descrescătoare.

Operaţia de sortare poate fi realizată prin ocuparea aceleiaşi zone de memorie sau prin ocuparea unei alte zone de memorie. În continuare, sunt prezentate metode de sortare cu ocuparea aceleiaşi zone de memorie.

Problema sortării poate fi formulată în cazul general astfel: se consideră o mulţime de **n** valori a_0, a_1, ... , a_{n-1} care trebuie aranjată astfel pentru ordonare crescătoare,:

$$a_0 \le a_1 \le a_2 \le \ldots \le a_{n-1} \text{, respectiv: } a_0 \ge a_1 \ge a_2 \ge \ldots \ge a_{n-1}$$

pentru ordonare descrescătoare.

5.8.1. Sortarea prin selecţie directă

Algoritmul pentru ordonarea crescătoare este următorul:

- se compară primul element din mulţime cu toate elementele care urmează după el şi dacă se găseşte un element mai mic decât primul atunci se schimbă între ele cele două elemente;
- se compară al doilea element al mulţimii cu toate elementele care urmează după el şi dacă se găseşte un element mai mic decât acesta se schimbă între ele cele două elemente;
- se procedează în mod asemănător cu al treilea, al patrulea, etc element al mulţimii, iar procesul continuă astfel până la penultimul element al mulţimii care va fi comparat cu ultimul.

Exemplul numeric:

Pentru exemplificare, se consideră o mulţime de 10 elemente şi se doreşte ordonarea acestora după valoare, în ordine crescătoare:

Element	1	2	3	4	5	6	7	8	9	10
Valoare	15	-12	5	8	14	-9	-15	0	10	-20

Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.1:

Tabelul 5.8.1. Sortarea prin selecţie directă

Element	1	2	3	4	5	6	7	8	9	10
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
Etapa 1:	15	-12	5	8	14	-9	-15	0	10	-20
Pasul 1a:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1b:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1c:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1d:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1e:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1f:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1g	-15	15	5	8	14	-9	-12	0	10	-20
Pasul 1h	-15	15	5	8	14	-9	-12	0	10	-20
Pasul 1i	-15	15	5	8	14	-9	-12	0	10	-20
Etapa 2:	-20	-15	5	8	14	-9	-12	0	10	15
Etapa 3:	-20	-15	-12	15	14	8	5	0	10	-9
...										
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**

Etapa 1: se compară primul element cu elementele care urmează după el şi dacă se găseşte un element mai mic decât primul atunci se schimbă între ele cele două elemente, astfel:

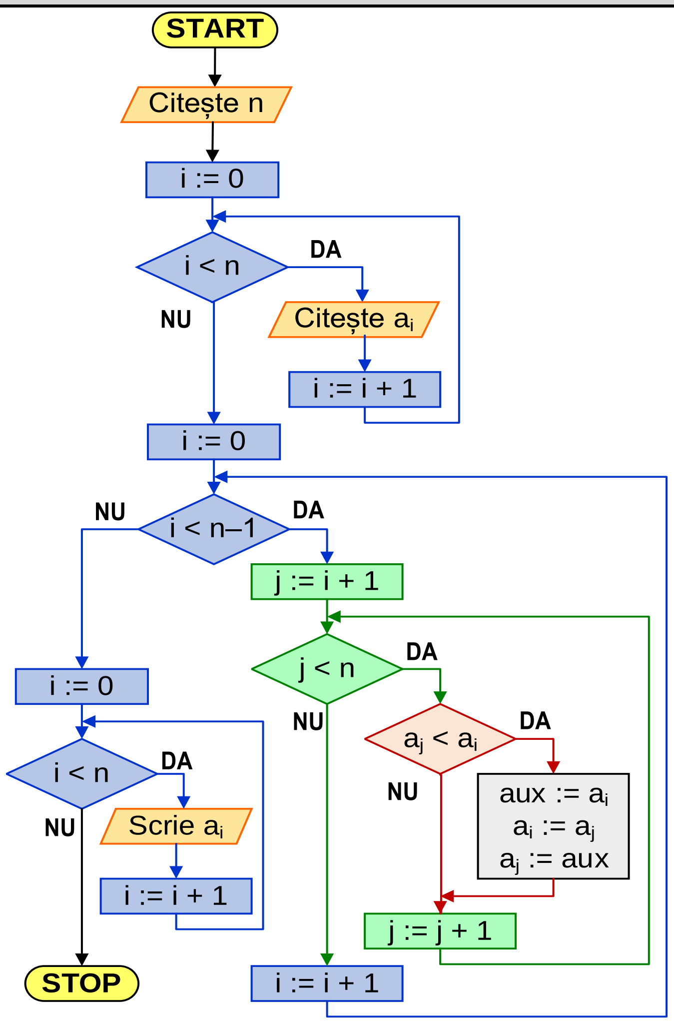
- Pasul 1a: se compară primul element (cu valoarea **15**) cu al doilea (cu valoarea **-12**) şi se constată că al doilea element este mai mic decât primul, deci **se schimbă între ele**;
- Pasul 1b: se compară primul element (cu valoarea **-12**) cu al treilea (cu valoarea **5**) şi se constată că primul element este mai mic decât al treilea, situaţie în care **se trece la pasul următor**;
- Pasul 1c: se compară primul element (cu valoarea **-12**) cu al patrulea (cu valoarea **8**) şi se constată că primul element este mai mic decât al patrulea, situaţie în care **se trece la pasul următor**;
- Pasul 1d: se compară primul element (cu valoarea **-12**) cu al cincilea (cu valoarea **14**) şi se constată că primul element este mai mic decât al cincilea, situaţie în care **se trece la pasul următor**;
- Pasul 1e: se compară primul element (cu valoarea **-12**) cu al şaselea (cu valoarea **-9**) şi se constată că primul element este mai mic decât al şaselea, situaţie în care **se trece la pasul următor**;
- Pasul 1f: se compară primul element (cu valoarea **-12**) cu al şaptelea (cu valoarea **-15**) şi se constată că primul element este mai mare decât al şaptelea, situaţie în care **se schimbă între ele**;
- Pasul 1g: se compară primul element (cu valoarea **-15**) cu al optulea (cu valoarea **0**) şi se constată că primul element este mai mic decât al optulea, situaţie în care **se trece la pasul următor**;
- Pasul 1h: se compară primul element (cu valoarea **-15**) cu al nouălea (cu valoarea **10**) şi se constată că primul element este mai mic decât al nouălea, situaţie în care **se trece la pasul următor**;
- Pasul 1i: se compară primul element (cu valoarea **-15**) cu al zecelea (cu valoarea **-20**) şi se constată că primul element este mai mare decât al zecelea, situaţie în care **se schimbă între ele**;

Etapa 2 - 9: se compară al doilea, al treilea, etc element cu elementele care urmează după el şi dacă se găseşte un element mai mic decât al doilea atunci se schimbă între ele cele două elemente;

În final, cu ajutorul unui ciclu cu contor se afişează mulţimea ordonată.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.8b.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  Pentru i ← 0, n - 1
    Citeşte a_i
  Sfârşit pentru
  Pentru i ← 0,n - 2
    Pentru j ← i + 1, n - 1
      Dacă $a_j < a_i$ **atunci**
        aux ← a_i
        $a_i \leftarrow a_j$
        a_j ← aux
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, n - 1
    Scrie a_i
  Sfârşit pentru
Sfârşit

Figura 5.8a. Reprezentarea algoritmului pentru sortarea (crescătoare) prin selecţie directă a elementelor unei mulţimi

Programul C

```c
#include<stdio.h>
int main(void)
{ int i,j,n,aux,a[50];
  printf("\n Introdu n, n = ");  scanf("%d",&n);
  printf("\n Introduceti elementele: \n");
  for( i = 0 ; i < n ; i++ )
    {  printf(" a[%2d] = ",i);  scanf("%d",&a[i]);  }
  for( i = 0 ; i < n - 1 ; i++ )
    for( j = i + 1 ; j < n ; j++ )
      if( a[j] < a[i] )
        { aux = a[i]; a[i] = a[j]; a[j] = aux; }
  printf("\n Sirul sortat este: \n");
  for( i = 0 ; i < n ; i++ )
    printf("%4d",a[i]);
}
```

Rularea programului

Introduceti n, n= **5**
a[0] = **11**
a[1] = **-2**
a[2] = **5**
a[3] = **-1**
a[4] = **0**
Sirul sortat este:
 -2 -1   0   5 11

Figura 5.8b. Programul C şi rularea acestuia pentru sortarea (crescătoare) prin selecţie directă a elementelor unei mulţimi

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.8.2. Sortarea prin interschimbare – Bubble Sort (metoda bulelor)

În continuare este prezentat algoritmul pentru ordonarea descrescătoare. Elementele mulţimii sunt ordonate descrescător dacă între oricare două elemente alăturate ale mulţimii există relaţia:

$$a_i \geq a_{i+1}$$

În cadrul metodei bulelor, se parcurge mulţimea, de la primul element până la penultimul şi se verifică dacă două elemente alăturate sunt în relaţia de mai sus. Condiţia care se pune în acest caz este:

$$a_i < a_{i+1}$$

Dacă condiţia este adevărată, înseamnă că elementele nu sunt ordonate corespunzător, caz în care se realizează o interschimbare între ele a celor două elemente. Se utilizează o variabilă (**kod**) cu ajutorul căreia se numără de câte ori s-a realizat interschimbarea elementelor alăturate. La început, înaintea începerii parcurgerii mulţimii, această variabila primeşte valoarea **0** (**kod = 0**).

Dacă condiţia este falsă, înseamnă că cele două elemente sunt ordonate corespunzător, situaţie în care se trece la verificarea următoarei perechi de elemente alăturate.

La final, după parcurgerea întregii mulţimi, se verifică valoarea variabilei cu ajutorul căreia s-au numărat interschimbările. Dacă valoarea variabilei **kod** este diferită de zero, adică în timpul parcurgerii s-au realizat interschimbări, se reia parcurgerea mulţimii. Dacă valoarea variabilei **kod** este zero înseamnă că în timpul parcurgerii nu s-a realizat nicio interschimbare, deci şirul este ordonat.

Exemplul numeric: Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.2:

Tabelul 5.8.2. Sortarea prin interschimbare (metoda bulelor)

Element	1	2	3	4	5	6	7	8	9	10	
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**	**kod**
Etapa 1:	15	-12	5	8	14	-9	-15	0	10	-20	0
Pasul 1.1:	15	-12	5	8	14	-9	-15	0	10	-20	0
Pasul 1.2:	15	5	-12	8	14	-9	-15	0	10	-20	1
Pasul 1.3:	15	5	8	-12	14	-9	-15	0	10	-20	2
Pasul 1.4:	15	5	8	14	-12	-9	-15	0	10	-20	3
Pasul 1.5:	15	5	8	14	-9	-12	-15	0	10	-20	4
Pasul 1.6:	15	5	8	14	-9	-12	-15	0	10	-20	4
Pasul 1.7:	15	5	8	14	-9	-12	0	-15	10	-20	5
Pasul 1.8:	15	5	8	14	-9	-12	0	10	-15	-20	6
Pasul 1.9:	15	5	8	14	-9	-12	0	10	-15	-20	**6**
Etapa 2:	15	8	14	5	-9	0	10	-12	-15	-20	
Etapa 3:	15	14	8	5	0	10	-9	-12	-15	-20	
Etapa 4:	15	14	8	5	10	0	-9	-12	-15	-20	
Etapa 5:	15	14	8	10	5	0	-9	-12	-15	-20	
Etapa 6:	15	14	10	8	5	0	-9	-12	-15	-20	
Final:	**15**	**14**	**10**	**8**	**5**	**0**	**-9**	**-12**	**-15**	**-20**	

Etapa 1:

Pasul 1.1: se verifică elementele $\mathbf{a_0}$ şi $\mathbf{a_1}$, deoarece $\mathbf{a_0 > a_1}$ se trece la următoarea pereche de elemente, valoarea variabilei **kod** nu se schimbă, adică **kod = 0**;

Pasul 1.2: se verifică elementele $\mathbf{a_1}$ şi $\mathbf{a_2}$, deoarece $\mathbf{a_1 < a_2}$ se interschimbă între ele cele două elemente şi se creşte valoarea variabilei **kod** cu **1**, adică **kod = 0 + 1 = 1**;

Pasul 1.3: se verifică elementele $\mathbf{a_2}$ şi $\mathbf{a_3}$, deoarece $\mathbf{a_2 < a_3}$ se interschimbă între ele cele două elemente şi se creşte valoarea variabilei **kod** cu **1**, adică **kod = 1 + 1 = 2**;

Pasul 1.4: se verifică elementele $\mathbf{a_3}$ şi $\mathbf{a_4}$, deoarece $\mathbf{a_3 < a_4}$ se interschimbă între ele cele două elemente şi se creşte valoarea variabilei **kod** cu **1**, adică **kod = 2 + 1 = 3**;

Pasul 1.5: se verifică elementele $\mathbf{a_4}$ şi $\mathbf{a_5}$, deoarece $\mathbf{a_4 < a_5}$ se interschimbă între ele cele două elemente şi se creşte valoarea variabilei **kod** cu **1**, adică **kod = 3 + 1 = 4**;

Pasul 1.6: se verifică elementele $\mathbf{a_5}$ şi $\mathbf{a_6}$, deoarece $\mathbf{a_5 > a_6}$ se trece la următoarea pereche de elemente, valoarea variabilei **kod** nu se schimbă, adică **kod = 4**;

Pasul 1.7: se verifică elementele $\mathbf{a_6}$ şi $\mathbf{a_7}$, deoarece $\mathbf{a_6 < a_7}$ se interschimbă între ele cele două elemente şi se creşte valoarea variabilei **kod** cu **1**, adică **kod = 4 + 1 = 5**;

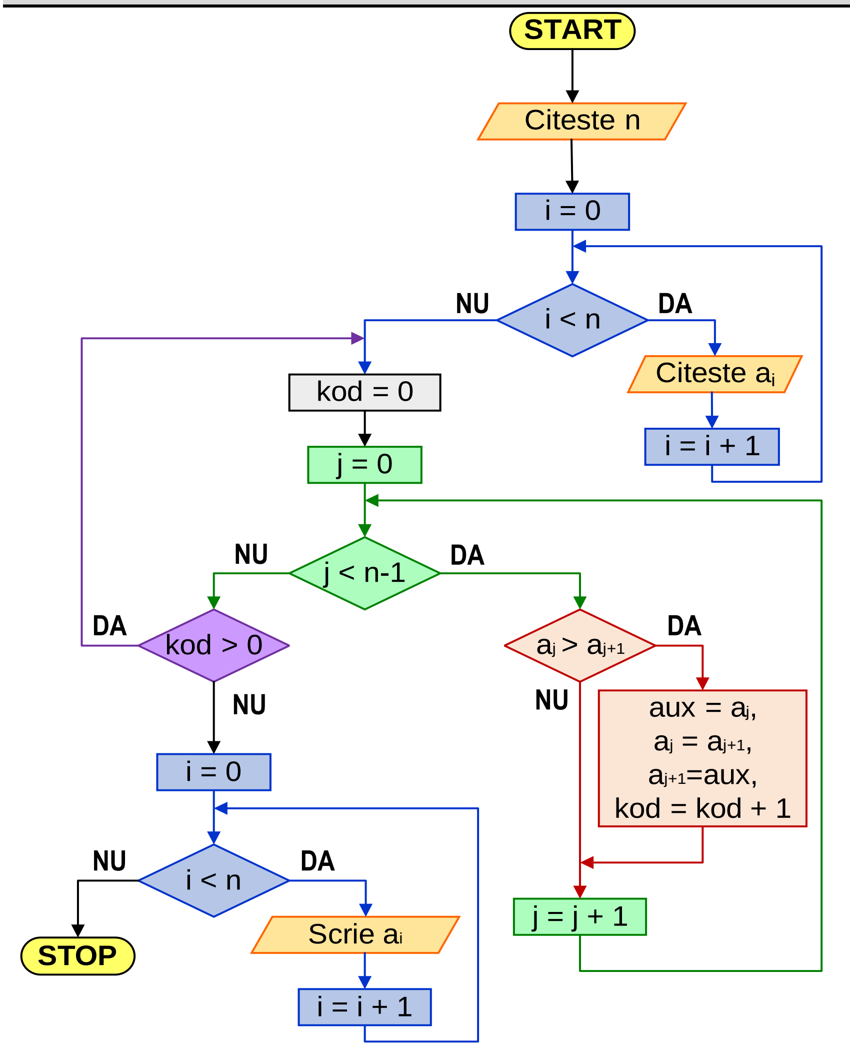
Pasul 1.8: se verifică elementele $\mathbf{a_7}$ şi $\mathbf{a_8}$, deoarece $\mathbf{a_7 < a_8}$ se interschimbă între ele cele două elemente şi se creşte valoarea variabilei **kod** cu **1**, adică **kod = 5 + 1 = 6**;

Pasul 1.9: se verifică elementele $\mathbf{a_8}$ şi $\mathbf{a_9}$, deoarece $\mathbf{a_8 > a_9}$ se trece la următoarea pereche de elemente, valoarea variabilei **kod** nu se schimbă, adică **kod = 6**;

Deoarece după parcurgerea şirului de valori, variabila **kod** are o valoare strict pozitivă (**kod > 0**) se reia parcurgerea şirului de valori, variabila **kod** primind din nou valoarea **0**. Procedeul se repetă în mod asemănător până când după o parcurgere a şirului variabila **kod** nu-şi mai modifică valoarea, adică **kod = 0**. În acest moment, şirul este ordonat descrescător.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8c, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.8d.

Schema logică	Limbajul pseudocod



Început
Citeşte n
Pentru i ← 0, n-1
Citeşte a_i
Sfârşit pentru
Execută
kod ← 0
Pentru j ← 0, n - 2
Dacă $a_j < a_{j+1}$ atunci
aux ← a_j
a_j ← a_{j+1}
a_{j+1} ← aux
kod ← kod + 1
Sfârşit dacă
Sfârşit pentru
Cât timp kod > 0
Pentru i ← 0, n - 1
Scrie a_i
Sfârşit pentru
Sfârşit

Figura 5.8c. Reprezentarea algoritmului pentru sortarea (descrescătoare) prin interschimbare (metoda bulelor)

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { int i,j,kod,n,a[25],aux; printf("\n Introdu n, n = "); scanf("%d",&n); for(i = 0 ; i < n ; i++) { printf(" a[%d] = ",i); scanf("%d",&a[i]); } do{ kod = 0; for(j = 0 ; j < n-1 ; j++) if(a[j] < a[j+1]) { aux = a[j]; a[j] = a[j+1]; a[j+1] = aux; kod++; } }while(kod > 0); printf("\n Sirul ordonat: \n"); for(i = 0 ; i < n ; i++) printf("%3d ",a[i]); }</pre>	Introduceti n, n= **5** a[0] = **21** a[1] = **-5** a[2] = **6** a[3] = **-1** a[4] = **0** Sirul ordonat: 21 6 0 -1 -5

Figura 5.8d. Programul C şi rularea acestuia pentru sortarea (descrescătoare) prin interschimbare (metoda bulelor)

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.8.3. Sortarea prin metoda selecţiei naive (naive sort)

În acest caz, algoritmul de ordonare crescătoare poate fi descris astfel:

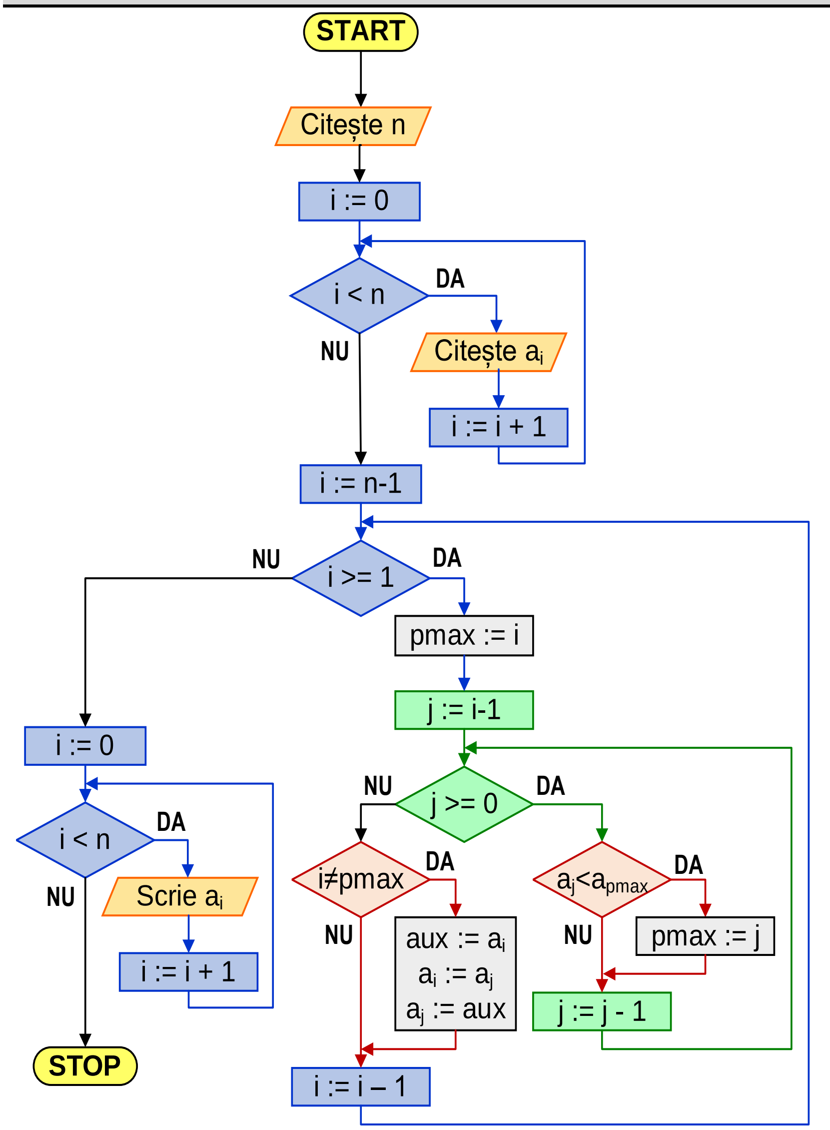
- în prima etapă, se caută în mulţimea neordonată cu **n** elemente, valoarea maximă şi poziţia acestei valori şi se realizează interschimbarea între valoarea maximă şi ultima valoare din mulţime (de indice **n - 1**);
- în a doua etapă, se caută în mulţimea cu **n - 1** elemente (se omite ultimul element) valoarea maximă şi poziţia acesteia şi se realizează interschimbarea între valoarea maximă găsită şi valoarea de indice **n - 2**;
- se repetă operaţiile până când se obţine o mulţime cu un singur element.

Exemplul numeric: Se consideră şirul prezentat în exemplul anterior, adică: **15**, **-12**, **5**, **8**, **14**, **-9**, **-15**, **0**, **10**, **-20**.
Etapa 1a: se parcurge mulţimea neordonată (cu **n** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 0** (valoarea **15**);
Etapa 1b: se schimbă între ele elementul de pe ultima poziţie (poziţia cu indicele **n - 1**) şi elementul de pe poziţia **pmax**, adică primul (valoarea **15**) şi ultimul element (valoarea **-20**);
Etapa 2a: se parcurge mulţimea neordonată (cu **n - 1** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 4** (valoarea **14**);
Etapa 2b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n - 2**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **14** şi **10**;
Etapa 3a: se parcurge mulţimea neordonată (cu **n - 2** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 4** (valoarea **10**);
Etapa 3b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n - 3**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **10** şi **0**;
Etapa 4a: se parcurge mulţimea neordonată (cu **n - 3** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 3** (valoarea **8**);
Etapa 4b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n - 4**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **8** şi **-15**;
Etapa 5a: se parcurge mulţimea neordonată (cu **n - 4** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 2** (valoarea **5**);
Etapa 5b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n - 5**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **5** şi **-9**;

Etapa 6a: se parcurge mulţimea neordonată (cu **n - 5** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 4** (valoarea **0**);
Etapa 6b: deoarece valoarea maximă a elementului din subşir este chiar pe ultima poziţie, nu se schimbă nimic;
Etapa 7a: se parcurge mulţimea neordonată (cu **n - 6** elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 2** (valoarea **-9**);
Etapa 7b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n - 7**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **-9** şi **-15**;
Etapa 8a: se parcurge mulţimea neordonată (cu 3 elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 1** (valoarea **-12**);
Etapa 8b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **2**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **-12** şi **-15**;
Etapa 9a: se parcurge mulţimea neordonată (cu 2 elemente) şi se determină indicele şi valoarea elementului maxim, în acest caz **pmax = 1** (valoarea **-15**);
Etapa 9b: deoarece valoarea maximă a elementului din subşir este chiar pe ultima poziţie, nu se schimbă nimic;

După parcurgerea acestor etape, mulţimea este ordonată crescător.

Schema logică	Limbajul pseudocod



Început
Citeşte n
Pentru i ← 0, n - 1
Citeşte a_i
Sfârşit pentru
Pentru i ← n - 1, 1, -1
pmax ← i
Pentru j ← i - 1,0,-1
Dacă $a_j < a_{pmax}$ **atunci**
pmax ← j
Sfârşit dacă
Sfârşit pentru
Dacă i ≠ pmax **atunci**
aux ← a_i
a_i ← a_{pmax}
a_{pmax} ← aux
Sfârşit dacă
Sfârşit pentru
Pentru i ← 0, n - 1
Scrie a_i
Sfârşit pentru
Sfârşit

Figura 5.8e. Reprezentarea algoritmului pentru sortarea (crescătoare) prin metoda selecţiei naive (naive sort)

Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.3:

Tabelul 5.8.3. Sortarea prin metoda selecţiei naive

Element	1	2	3	4	5	6	7	8	9	10
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
Etapa 1a:	15	-12	5	8	14	-9	-15	0	10	-20
Etapa 1b:	-20	-12	5	8	14	-9	-15	0	10	**15**
Etapa 2a:	-20	-12	5	8	14	-9	-15	0	10	**15**
Etapa 2b:	-20	-12	5	8	10	-9	-15	0	**14**	**15**
Etapa 3a:	-20	-12	5	8	10	-9	-15	0	**14**	**15**
Etapa 3b:	-20	-12	5	8	0	-9	-15	**10**	**14**	**15**
Etapa 4a:	-20	-12	5	8	0	-9	-15	**10**	**14**	**15**
Etapa 4b:	-20	-12	5	-15	0	-9	**8**	**10**	**14**	**15**
Etapa 5a:	-20	-12	5	-15	0	-9	**8**	**10**	**14**	**15**
Etapa 5b:	-20	-12	-9	-15	0	**5**	**8**	**10**	**14**	**15**
Etapa 6a:	-20	-12	-9	-15	0	**5**	**8**	**10**	**14**	**15**
Etapa 6b:	-20	-12	-9	-15	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 7a:	-20	-12	-9	-15	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 7b:	-20	-12	-15	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 8a:	-20	-12	-15	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 8b:	-20	-15	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 9a:	-20	-15	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 9b:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8e, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.8f.

Programul C

```
#include<stdio.h>
int main(void)
{ int i,j,pmax,n,a[25],aux;
  printf("\n Introdu n, n = ");  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
     { printf(" a[%d] = ",i); scanf("%d",&a[i]);   }
  for( i = n-1 ; i >= 1 ; i-- )
     { pmax = i;
     for( j = i - 1 ; j >= 0 ; j-- )
           if( a[j] < a[pmax] )  pmax = j;
     if( i != pmax )
           { aux = a[i]; a[i] = a[pmax]; a[pmax] = aux; }
     }
  printf("\n Sirul ordonat: \n");
  for( i = 0 ; i < n ; i++ )
       printf("%3d ",a[i]);
}
```

Rularea programului

Introduceti n, n= **7**
a[0] = **21**
a[1] = **-5**
a[2] = **6**
a[3] = **-1**
a[4] = **0**
a[5] = **34**
a[6] = **100**

Sirul ordonat:
-5 -1 0 6 21 34 100

Figura 5.8f. Programul C şi rularea acestuia pentru sortarea (crescătoare) prin metoda selecţiei naive (naive sort)

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

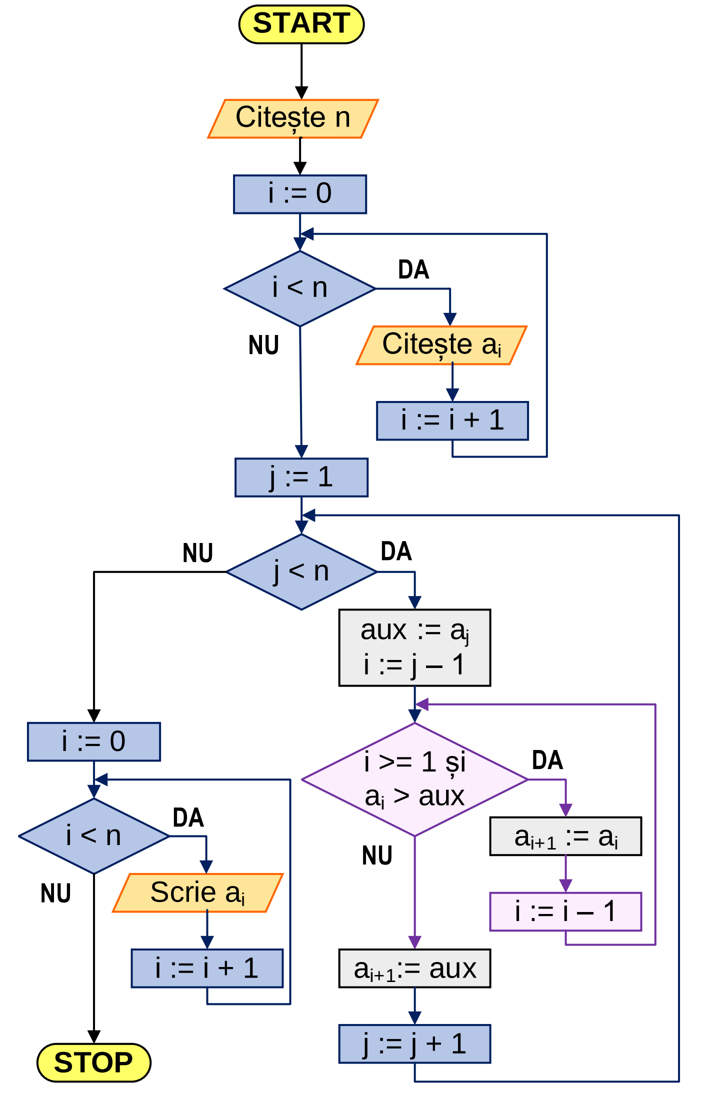
5.8.4. Sortarea prin metoda inserţiei (Insertion Sort)

Pentru ordonarea crescătoare cu ajutorul acestei metode se consideră că primele **i** elemente ale mulţimii sunt ordonate iar elementul de pe poziţia **i** va fi inserat în subşirul **[0, i - 1]**, astfel încât subşirul **[0, i]** să fie ordonat. Inserarea se realizează astfel:

- se memorează într-o variabilă ajutătoare valoarea elementului de pe poziţia **i**;
- de la poziţia **i - 1** până la poziţia **0**, se deplasează cu o poziţie la dreapta toate elementele mai mari decât valoarea memorată în variabila ajutătoare;
- se inserează valoarea variabilei ajutătoare în locul ultimului element deplasat în etapa anterioară.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8g, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.8h.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  Pentru i ← 0, n - 1
    Citeşte a_i
  Sfârşit pentru
  Pentru j ← 1, n - 1
    aux ← a_j
    i ← j - 1
    Cât timp i ≥ 0 **ŞI** a_i > aux **execută**
      a_{i+1} ← a_i
      i ← i - 1
    Sfârşit cât timp
    a_{i+1} ← aux
  Sfârşit pentru
  Pentru i ← 0, n - 1
    Scrie a_i
  Sfârşit pentru
Sfârşit

Figura 5.8g. Reprezentarea algoritmului pentru sortarea (crescătoare) prin metoda inserţiei (Insertion Sort)

Programul C	Rularea programului

Programul C

```
#include<stdio.h>
int main(void)
{ int i,j,n,a[25],aux;
  printf("\n Introdu n, n = ");  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
     { printf(" a[%d] = ",i); scanf("%d",&a[i]);   }
  for( j = 1 ; j <= n-1 ; j++ )
     { aux = a[j];  i = j - 1;
       while( i >= 0 && a[i] > aux )
            { a[i+1] = a[i]; i--;  }
       a[i+1] = aux;
     }
    printf("\n Sirul ordonat: \n");
    for( i = 0 ; i < n ; i++ )
         printf("%3d ",a[i]);
}
```

Rularea programului

```
Introduceti n, n= 6
 a[ 0] = 11
 a[ 1] = -9
 a[ 2] = 8
 a[ 3] = -3
 a[ 4] = 0
 a[ 5] = 34

Sirul ordonat:
  -9  -3   0   8  11   34
```

Figura 5.8h. Programul C şi rularea acestuia pentru sortarea (crescătoare) prin metoda inserţiei (Insertion Sort)

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric: Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.4:

Tabelul 5.8.4. Sortarea prin metoda inserţiei (Insertion Sort)

Element	1	2	3	4	5	6	7	8	9	10	**aux**
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**	
Etapa 1:	15	**-12**	5	8	14	-9	-15	0	10	-20	**-12**
	-12	15	5	8	14	-9	-15	0	10	-20	
Etapa 2:	-12	15	**5**	8	14	-9	-15	0	10	-20	**5**
	-12	5	15	8	14	-9	-15	0	10	-20	
Etapa 3:	-12	5	15	**8**	14	-9	-15	0	10	-20	**8**
	-12	5	8	15	14	-9	-15	0	10	-20	
Etapa 4:	-12	5	8	15	**14**	-9	-15	0	10	-20	**14**
	-12	5	8	14	15	-9	-15	0	10	-20	
Etapa 5:	-12	5	8	14	15	**-9**	-15	0	10	-20	**-9**
	-12	-9	5	8	14	15	-15	0	10	-20	
Etapa 6:	-12	-9	5	8	14	15	**-15**	0	10	-20	**-15**
	-15	-12	-9	5	8	14	15	0	10	-20	
Etapa 7:	-15	-12	-9	5	8	14	15	**0**	10	-20	**0**
	-15	-12	-9	0	5	8	14	15	10	-20	
Etapa 8:	-15	-12	-9	0	5	8	14	15	**10**	-20	**10**
	-15	-12	-9	0	5	8	10	14	15	-20	
Etapa 9:	-15	-12	-9	0	5	8	10	14	15	**-20**	**-20**
	-20	-15	-12	-9	0	5	8	10	14	15	
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**	

Etapa 1: Variabila **j** are valoarea **1**, deci variabila **aux** primeşte valoarea **-12** (**aux = -12**). Se verifică în subşirul format din primul element dacă elementele acestuia sunt mai mari decât **-12**. Primul element al şirului având valoarea **15** este mai mare decât **-12**, deci se deplasează la dreapta cu o unitate, iar pe prima poziţie se aşază valoarea **-12**;

Etapa 2: Variabila **j** are valoarea **2**, deci variabila **aux** primeşte valoarea **5** (**aux = 5**). Se verifică în subşirul format din primele două elemente dacă elementele acestuia sunt mai mari decât **5**. Al doilea element al şirului având valoarea **15** este mai mare decât **5**, deci se deplasează la dreapta cu o unitate, iar pe poziţia acestuia se aşază valoarea **5**;
Etapa 3: Variabila **j** are valoarea **3**, deci variabila **aux** primeşte valoarea **8** (**aux = 8**). Se verifică în subşirul format din primele trei elemente dacă elementele acestuia sunt mai mari decât **8**. Al treilea element al şirului având valoarea **15** este mai mare decât **8**, deci se deplasează la dreapta cu o unitate, iar pe poziţia acestuia se aşază valoarea **8**;
Etapa 4: Variabila **j** are valoarea **4**, deci variabila **aux** primeşte valoarea **14** (**aux = 14**). Se verifică în subşirul format din primele patru elemente dacă elementele acestuia sunt mai mari decât **14**. Al patrulea element al şirului având valoarea **15** este mai mare decât **14**, deci se deplasează la dreapta cu o unitate, iar pe poziţia acestuia se aşază valoarea **14**;
Etapa 5: Variabila **j** are valoarea **5**, deci variabila **aux** primeşte valoarea **-9** (**aux = -9**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **-9**. Elementele cu valorile **5**, **8**, **14**, **15** având valori mai mari decât **-9**, se deplasează la dreapta cu o unitate, iar pe poziţia a doua se aşază valoarea **-9**;
Etapa 6: Variabila **j** are valoarea **6**, deci variabila **aux** primeşte valoarea **-15** (**aux = -15**). Se verifică în subşirul format din elementele din stânga dacă sunt mai mari decât **-15**. Toate elementele din stânga au având valori mai mari decât **-15**, deci se deplasează la dreapta cu o unitate, iar pe prima poziţie se aşază valoarea **-15**;
Etapa 7: Variabila **j** are valoarea **7**, deci variabila **aux** primeşte valoarea **0** (**aux = 0**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **0**. Elementele cu valorile **5**, **8**, **14**, **15** având valori mai mari decât **0**, se deplasează la dreapta cu o unitate, iar pe poziţia a patra se aşază valoarea **0**;
Etapa 8: Variabila **j** are valoarea **8**, deci variabila **aux** primeşte valoarea **10** (**aux = 10**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **10**. Elementele cu valorile **14**, **15** având valori mai mari decât **10**, se deplasează la dreapta cu o unitate, iar pe poziţia şapte se aşază valoarea **10**;
Etapa 9: Variabila **j** are valoarea **9**, deci variabila **aux** primeşte valoarea **-20** (**aux = -20**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **-20**. Se observă că toate elementele din stânga au valori mai mari decât **-20**, astfel că se deplasează la dreapta cu o unitate, iar pe prima poziţie şapte se aşază valoarea **-20**.

Se observă că şirul de valori este ordonat crescător.

5.8.5. Sortarea prin numărare (Counting Sort)

În cazul acestei metode se utilizează spaţiu suplimentar de memorie, utilizându-se următorii vectori:
- vectorul sursă (nesortat), notat cu **a**;
- vectorul destinaţie (sortat), notat cu **b**;
- vector numărător (care conţine contoarele), notat **c**;

Algoritmul pentru ordonarea crescătoare utilizând această metodă constă în parcurgerea vectorului sursă şi pentru fiecare element al acestuia se numără câte elemente mai mici decât elementul curent sunt, valoarea astfel obţinută reprezintă poziţia pe care o va avea elementul curent al vectorului sursă în vectorul destinaţie, valoarea memorându-se în vectorul numărător **c**, pe poziţia corespunzătoare elementului curent al vectorului sursă;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8i, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.8j.

Exemplul numeric: Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.5:

Tabelul 5.8.5. Sortarea prin metoda inserţiei (Insertion Sort)

Element	1	2	3	4	5	6	7	8	9	10
Indice	0	1	2	3	4	5	6	7	8	9
a [i]	15	-12	5	8	14	-9	-15	0	10	-20
c [i]	9	2	5	6	8	3	1	4	7	0
b [i]	-20	-15	-12	-9	0	5	8	10	14	15
Final:	-20	-15	-12	-9	0	5	8	10	14	15

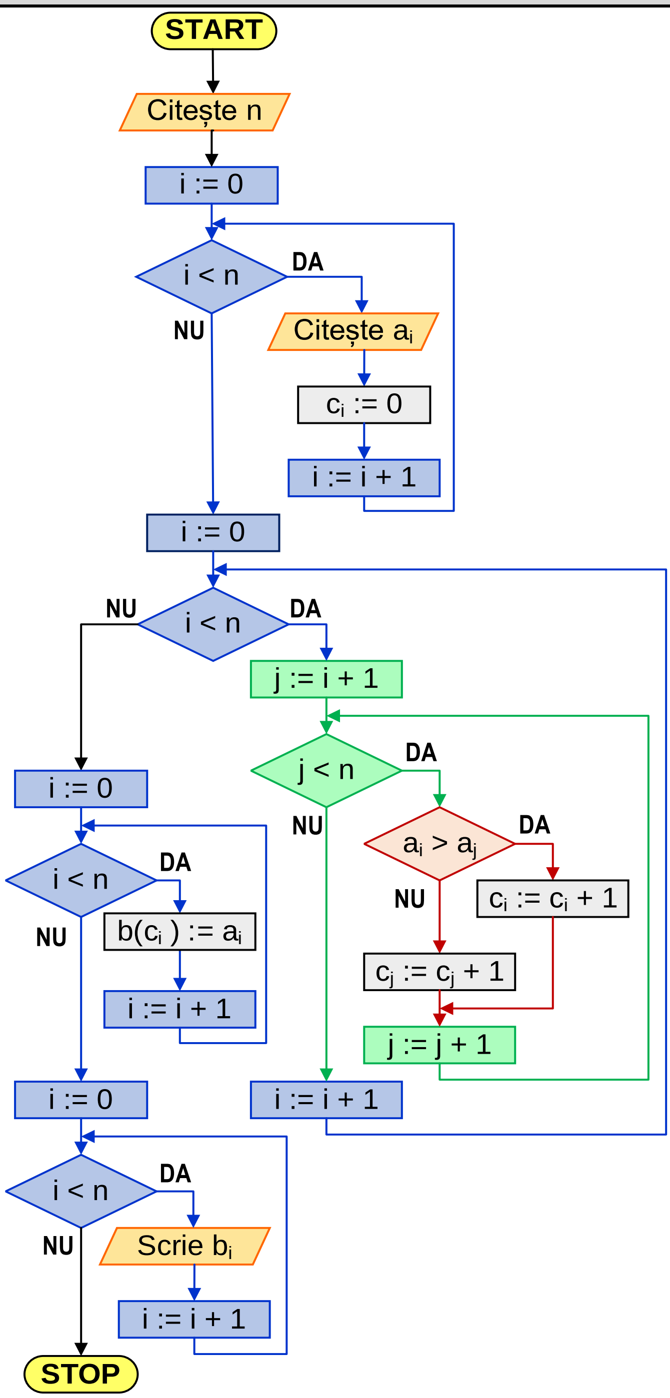
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n **Pentru** $i \leftarrow 0, n - 1$ **Citeşte** a_i $c_i \leftarrow 0$ **Sfârşit pentru** **Pentru** $i \leftarrow 0, n - 1$ **Pentru** $j \leftarrow i + 1, n-1$ **Dacă** $a_i > a_j$ **atunci** $c_i \leftarrow c_i+1$ **altfel** $c_j \leftarrow c_j+1$ **Sfârşit dacă** **Sfârşit pentru** **Sfârşit pentru** **Pentru** $i \leftarrow 0, n - 1$ $b_{c_i} \leftarrow a_i$ **Sfârşit pentru** **Pentru** $i \leftarrow 0, n - 1$ **Scrie** b_i **Sfârşit pentru** **Sfârşit**

Figura 5.8i. Reprezentarea algoritmului pentru sortarea (crescătoare) prin numărare (Counting Sort)

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { int i,j,n,a[25],b[25],c[25]; printf("\n Introdu n, n = "); scanf("%d",&n); for(i = 0 ; i < n ; i++) { printf(" a[%d] = ",i); scanf("%d",&a[i]); c[i] = 0; } for(i = 0 ; i < n ; i++) for(j = i + 1 ; j < n ; j++) if(a[i] > a[j]) c[i]++; else c[j]++; printf("\n Sirul ordonat: \n"); for(i=0 ; i < n ; i++) b[c[i]] = a[i]; for(i=0 ; i < n ; i++) printf("%3d ",b[i]); }</pre>	Introduceti n, n= **10** a[0] = **15** a[1] = **-12** a[2] = **5** a[3] = **8** a[4] = **14** a[5] = **-9** a[6] = **-15** a[7] = **0** a[8] = **10** a[9] = **-20** Sirul ordonat: -20 -15 -12 -9 0 5 8 10 14 15

Figura 5.8j. Programul C şi rularea acestuia pentru sortarea (crescătoare) prin numărare (Counting Sort)

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.9. Căutarea unui element într-o mulţime neordonată (căutare secvenţială)

Algoritmul de căutare secvenţială este unul dintre cei mai simpli algoritmi de calcul studiaţi. Cu ajutorul acestuia se urmăreşte să se verifice dacă o valoarea (notată cu **v**) se află printre elementele unui vector, notat cu **a**.

Se parcurge vectorul, de la primul element până la ultimul şi se compară valoarea **v** cu fiecare element al vectorului, utilizând operatorul de egalitate.

Dacă există egalitate între valoarea **v** şi valoarea unui element al vectorului, o variabilă ajutătoare notată **găsit** primeşte valoarea **1**. Iniţial aceasta a primit valoarea 0 (**găsit = 0**).

În final, după parcurgerea vectorului, se verifică valoarea variabilei **găsit**, dacă aceasta este **1** se afişează pe ecran un mesaj prin care se specifică faptul că variabila **v** se află printre elementele vectorului, iar în cazul în care aceasta este **0** atunci se afişează pe ecran un mesaj prin care se specifică că variabila **v** nu se află printre elementele vectorului.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.9a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.9b.

Exemplul numeric nr. 1: Se caută valoarea **5** în şirul de valori **15**, **-12**, **5**, **8**, **14**, **-9**, **-15**, **0**, **10**, **-20**.
Modul de funcţionare al algoritmului este prezentat în tabelul V.9.1:

Tabelul V.9.1. Căutarea unui element într-un vector (căutare secvenţială) – exemplul 1

Element	1	2	3	4	5	6	7	8	9	10
Indice i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
a [i]	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
v == a[i]	**NU**	**NU**	**DA**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**
gasit:	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Observaţie: valoarea variabilei ajutătoare **gasit** rămâne **1** după ce s-a găsit corespondenţa între valoarea **v** şi unul dintre elementele vectorului.

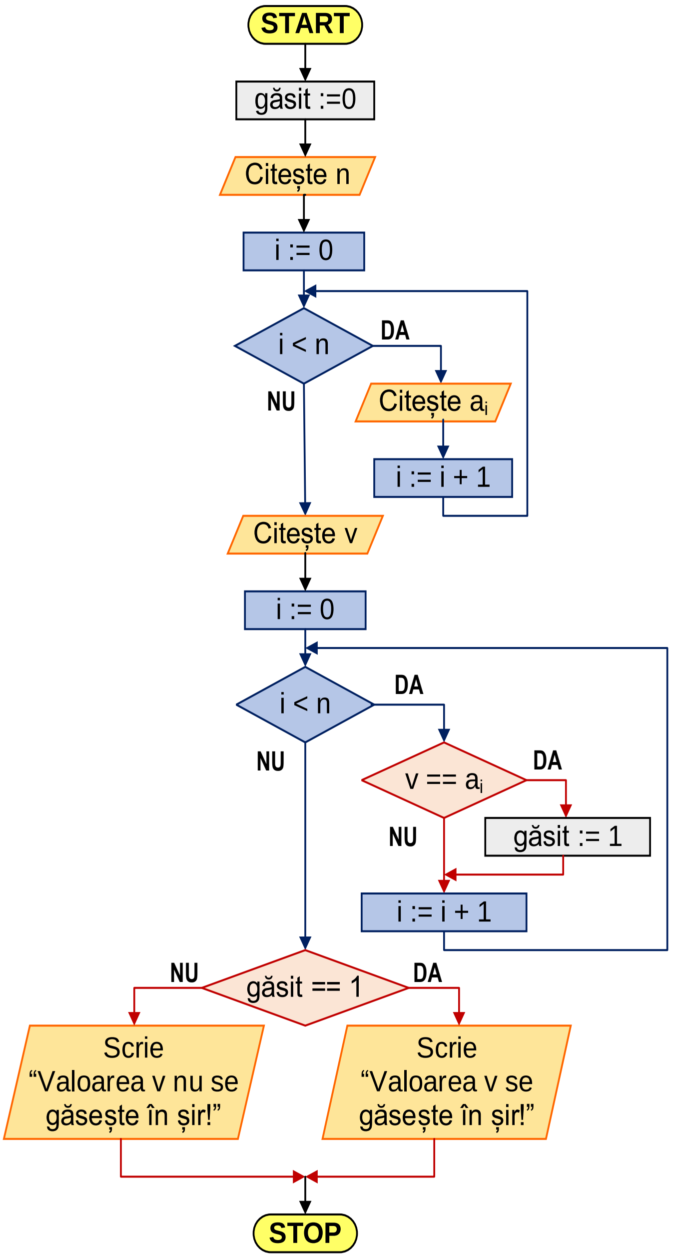
Exemplul numeric nr. 2: Se caută valoarea **3** în şirul de valori **15**, **-12**, **5**, **8**, **14**, **-9**, **-15**, **0**, **10**, **-20**.
Modul de funcţionare al algoritmului este prezentat în tabelul V.9.2:

Tabelul V.9.2. Căutarea unui element într-un vector (căutare secvenţială) – exemplul 2

Element	1	2	3	4	5	6	7	8	9	10
Indice i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
a [i]	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
v == a[i]	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**
gasit:	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Observaţie: valoarea variabilei ajutătoare **gasit** rămâne **0** deoarece nu s-a găsit corespondenţă între valoarea **v** şi elementele vectorului.

Schema logică	Limbajul pseudocod




```
Început
   Citeşte n
   Pentru i ← 1, n
       Citeşte a_i
   Sfârşit pentru
   Citeşte v
   gasit ← 0
   Pentru i ← 1, n
       Dacă v == a_i atunci
           gasit ← 1
       Sfârşit dacă
   Sfârşit pentru
   Dacă gasit == 1 atunci
       Scrie ”Valoarea se gaseste in sir!”
   altfel
       Scrie ”Valoarea nu se gaseste in sir!”
   Sfârşit dacă
Sfârşit
```

Figura 5.9a. Reprezentarea algoritmului pentru căutarea unui element într-un vector (căutare secvenţială)

Programul C	Rularea programului
`#include<stdio.h>` `int main(void)` `{ int a[50],n,v,i,gasit=0;` `printf("\n Introdu n = "); scanf("%d",&n);` `for( i = 0 ; i < n ; i++ )` `    { printf("\n a[%d] = ",i ); scanf("%d",&a[i]);` `    }` `printf("\n Introdu valoarea v = "); scanf("%d",&v);` `for( i = 0 ; i < n ; i++ )` `      if( v == a[i] ) gasit=1;` `if( gasit == 1 )` `     printf("\n Valoarea %d se gaseste in sir!", v);` `else` `     printf("\n Valoarea %d NU se gaseste in sir!", v);` `}`	**Exemplul rulare 1:** Introduceti n = **7** a[0] = **15** a[1] = **-12** a[2] = **5** a[3] = **8** a[4] = **14** a[5] = **-9** a[6] = **-15** Introduceti valoarea v = **5** Valoarea 5 se gaseste in sir! **Exemplul rulare 2:** Introduceti n= **5** a[0] = **11** a[1] = **0** a[2] = **-2** a[3] = **9** a[4] = **6** Introduceti valoarea v= **10** Valoarea 10 nu se gaseste in sir!

Figura 5.9b. Programul C şi rularea acestuia pentru căutarea unui element într-un vector (căutare secvenţială)

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.10. Căutarea unui element într-o mulţime ordonată (căutare binară)

Algoritmul de căutare binară este utilizat pentru găsirea unui element într-o listă ordonată de elemente (crescător sau descrescător). Algoritmul funcţionează pe baza tehnicii „divide et impera” (împarte şi cucereşte).

În cadrul algoritmului, valoarea căutată este comparată cu cea a elementului din mijlocul listei, existând trei situaţii posibile:

- dacă valoarea căutată este egală cu cea de la mijlocul listei, algoritmul se finalizează;
- dacă valoarea căutată este mai mare decât valoarea de la mijlocul listei, algoritmul se reia de la mijlocul listei până la sfârşit;
- dacă valoarea căutată este mai mică decât valoarea de la mijlocul listei, algoritmul se reia de la începutul listei până la mijlocul acesteia.

Se consideră un şir de numere ordonat crescător, notat cu **a** şi trei variabile **st**, **dr**, şi **mj** cu ajutorul cărora se memorează indicii extremităţilor, respectiv a mijlocului şirului considerat. Se utilizează o variabilă ajutătoare, cu numele **găsit**, cu valoarea iniţială **0**. Când se găseşte egalitate între valoarea căutată şi un element din şir, variabila **gasit** primeşte valoarea **1**.

Algoritmul de căutare presupune parcurgerea următorilor paşi:

- se citeşte numărul de elemente din şir, notat cu **n**;
- se citesc în ordine crescătoare, elementele şirului;
- se citeşte valoarea căutată, notată cu **v**;
- se iniţializează variabilele **st**, **dr** şi **gasit** cu valorile **0**, **n - 1**, respectiv **0**;
- cu ajutorul unui ciclu cu test iniţial de realizează restrângerea şirului în care se caută valoarea **v**, după metoda prezentată anterior, atât timp cât se îndeplinesc simultan condiţiile **st <= dr**, respectiv **gasit != 1**. În cadrul ciclului, se determină valoarea variabilei **mj** şi se compară valoarea căutată **v** cu valoarea elementului din şir a cărui indice este **mj**, astfel:

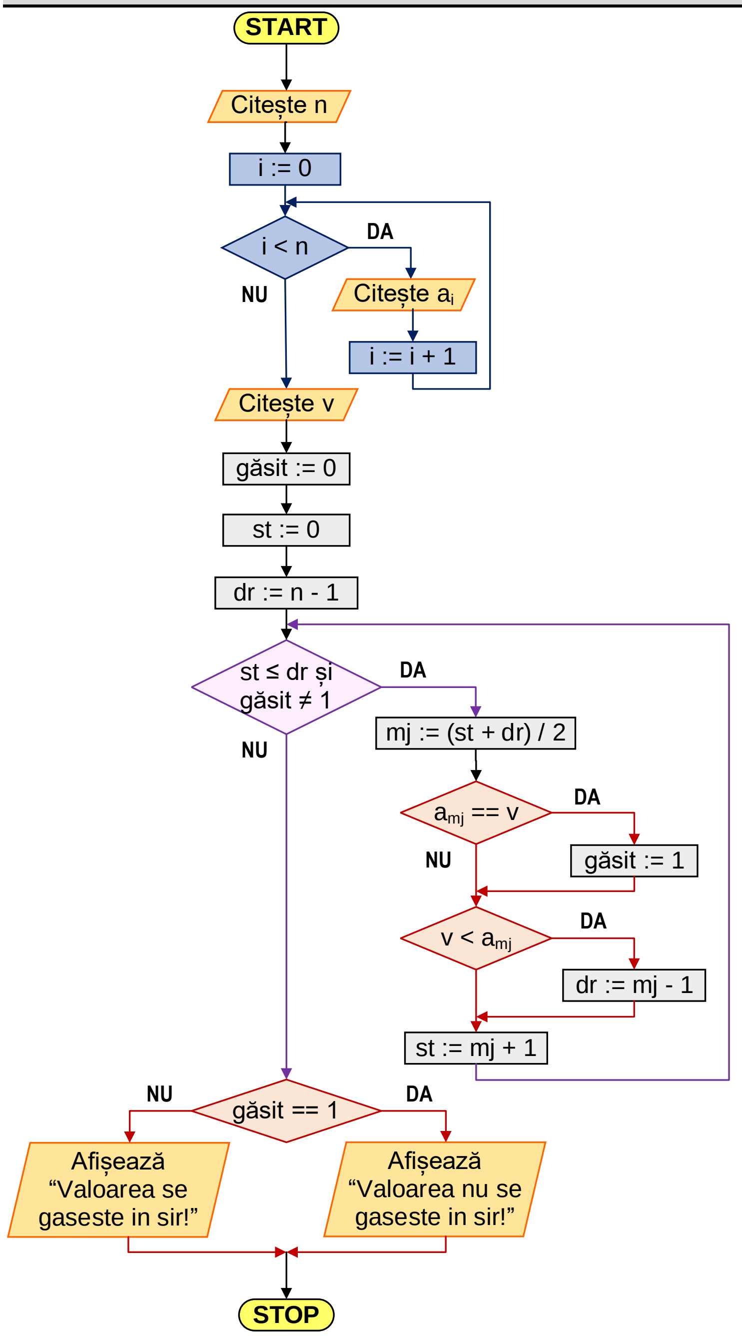
- dacă cele două valori sunt egale înseamnă că s-a găsit valoarea **v** în şir, variabila **gasit** primind valoarea **1**, astfel că nu se mai îndeplineşte condiţia de execuţie a corpului ciclului şi se părăseşte ciclul, rularea continuând cu instrucţiunea următoare ciclului;

- dacă cele două valori nu sunt egale, se verifică dacă valoarea variabilei **v** este mai mică decât cea a elementului cu indicele **mj**, existând două posibilităţi: **v < a[mj]** situaţie în care se restrânge şirul de elemente, variabila **dr** primind valoarea **mj - 1**, respectiv dacă **v > a[mj]** se restrânge şirul de elemente prin atribuirea variabilei **st** a valorii **mj + 1**;

Cu noul şir de valori se revine la căutarea valorii **v**.

- după părăsirea ciclului cu test iniţial se verifică valoarea variabilei **gasit**. Dacă aceasta este egală cu **1** înseamnă că valoarea căutată **v** se află în şir, în caz contrar valoarea nu se află în şir.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.10a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.10b.

Schema logică	Limbajul pseudocod
	see below

Început
  Citeşte n
  Pentru i ← 0, n - 1
    Citeşte a_i
  Sfârşit pentru
  Citeşte v
  gasit ← 0
  st ← 0
  dr ← n - 1
  Cât timp st ≤ dr **ŞI** gasit ≠ 1 **execută**
    mj ← (st + dr) DIV 2
    Dacă v == a_{mj} **atunci**
      gasit ← 1
    altfel
      Dacă v < a_{mj} **atunci**
        dr ← mj - 1
      altfel
        st ← mj + 1
      Sfârşit dacă
    Sfârşit dacă
  Sfârşit cât timp
  Dacă gasit == 1 **atunci**
    Scrie "Valoarea se gaseste in sir!"
  altfel
    Scrie "Valoarea nu se gaseste in sir!"
  Sfârşit dacă
Sfârşit

Figura 5.10a. Reprezentarea algoritmului pentru căutarea unui element într-un vector (căutare binară)

Exemplul numeric 1: Se caută valoarea **5** în şirul de valori: **-20, -15, -12, -9, 0, 5, 8, 10, 14, 15**.

Modul de funcţionare al algoritmului este prezentat în tabelul 5.10.1:

Tabelul 5.10.1. Căutarea unui element într-un vector (căutare secvenţială) – exemplul 1

	Element	1	2	3	4	5	6	7	8	9	10
	Indice i	0	1	2	3	4	5	6	7	8	9
Iniţial	a [i]	-20	-15	-12	-9	0	5	8	10	14	15
Etapa 1	st =	0		dr =		9		mj =		4	
	şir obţinut						5	8	10	14	15
Etapa 2	st =	5		dr =		9		mj =		7	
	şir obţinut						5	8			
Etapa 3	st =	5		dr =		6		mj =		5	
	şir obţinut						5				
Valoarea **5** apartine sirului!											

Programul C

```c
#include<stdio.h>
int main(void)
{ int i,n,v,st,dr,mj,gasit,a[50];
  printf("\n Introdu n = ");  scanf( "%d",&n );
  for( i = 0 ; i < n ; i++ )
    {  printf("\n a[%d] = ",i);  scanf("%d",&a[i]);   }
  printf("\n Introdu valoarea v = "); scanf("%d",&v);
  gasit = 0; st = 0; dr = n - 1;
  while( st <= dr && gasit != 1 )
    { mj = (st+dr) / 2;
      if ( a[mj] == v )  gasit=1;
      else
          if ( v < a[mj] ) dr = mj - 1;
          else st = mj + 1;
    }
  if( gasit == 1 )
        printf("\n Valoarea %d se gaseste in sir!",v);
  else
        printf("\n Valoarea %d NU se gaseste in sir!",v);
}
```

Rularea programului

Exemplul rulare 1:
Introduceti n= **6**
a[0] = **1**
a[1] = **2**
a[2] = **3**
a[3] = **5**
a[4] = **8**
a[5] = **11**
Introduceti valoarea v= **8**
Valoarea 8 se gaseste in sir!

Exemplul rulare 2:
Introduceti n= **6**
a[0] = **1**
a[1] = **2**
a[2] = **3**
a[3] = **5**
a[4] = **8**
a[5] = **11**
Introduceti valoarea v= **20**
Valoarea 20 NU se gaseste in sir!

Figura 5.10b. Programul C şi rularea acestuia pentru căutarea unui element într-un vector (căutare binară)

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.11. Inserarea unui element într-o mulţime, pe o anumită poziţie

Se consideră o mulţime notată cu **a**, cu **n** elemente şi se doreşte inserarea unui element în mulţime, pe o anumită poziţie – notată cu **p**, mulţimea astfel obţinută având **n + 1** elemente. Mulţimea nouă se va forma astfel:
- până la poziţia **p - 1**, elementele mulţimii rămân la fel, ca la început;
- pe poziţia **p** se inserează valoarea **v**;
- de la poziţia **p + 1** la **n + 1** se inserează elementele mulţimii iniţiale de la poziţia **p** la **n**;

Astfel, pentru crearea noii mulţimi este necesar şi suficient să se parcurgă mulţimea de la ultimul element al său (de indice **n + 1**) până la poziţia **p + 1**, atribuindu-se elementului de pe poziţia **i**, valoarea elementului de pe poziţia **i - 1** din mulţime, adică se realizează o deplasare la dreapta a elementelor mulţimii de la poziţia **p** şi până la sfârşit, lăsând astfel liberă poziţia **p**, pe care se inserează valoarea **v**.

Algoritmul de inserare constă în parcurgerea următorilor paşi:
- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se citeşte valoarea **v** care se doreşte a fi inserată;
- se citeşte valoarea **p** a poziţiei în care se doreşte inserarea variabilei **v**;
- cu ajutorul unui ciclu cu contor se realizează deplasarea la dreapta cu o poziţie a elementelor mulţimii situate de la poziţia **p** şi până la sfârşit, operaţie care presupune parcurgerea noii mulţimi în ordine inversă de la poziţia **n + 1** până la poziţia **p + 1**;
- pe poziţia **p** se inserează valoarea **v**, restul mulţimii rămâne la fel;
- în final se afişează mulţimea astfel obţinută.

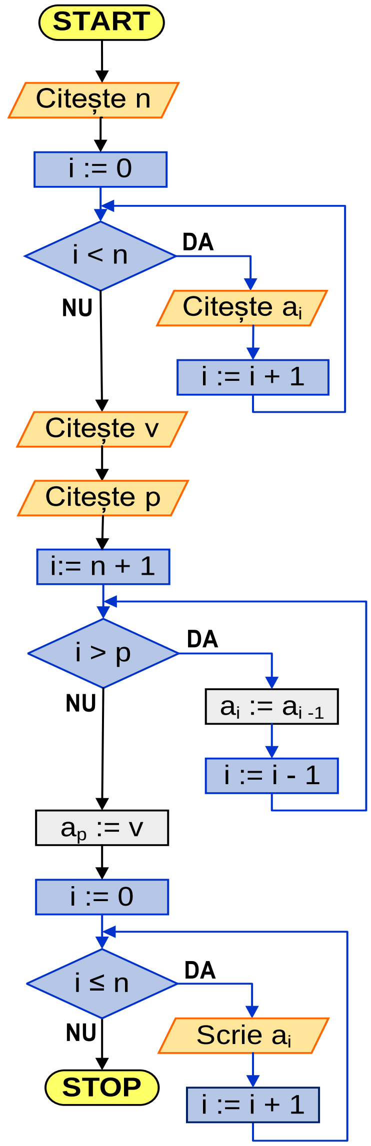
Exemplul numeric: Se inserează valoarea **0** pe poziţia **4** în şirul de valori: **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**.
Modul de funcţionare al algoritmului este prezentat în tabelul 5.11.1:

Tabelul 5.11.1. Inserarea unui element într-o mulţime pe poziţia p

	Element	1	2	3	4	5	6	7	8	9	
Iniţial	**Indice i**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
	a [i]	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Intermediar	**Indice i**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	a [i]	**1**	**2**	**3**		**4**	**5**	**6**	**7**	**8**	**9**
Final	**Indice i**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	a[i]	**1**	**2**	**3**	**0**	**4**	**5**	**6**	**7**	**8**	**9**

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.11a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.11b.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  Pentru i ← 0, n - 1
    Citeşte a_i
  Sfârşit pentru
  Citeşte v
  Citeşte p
  Pentru i ← n+1, p+1, -1
    $a_i \leftarrow a_{i-1}$
  Sfârşit pentru
  $a_p \leftarrow v$
  Pentru i ← 0, n
    Scrie a_i
  Sfârşit pentru
Sfârşit

Figura 5.11a. Reprezentarea algoritmului pentru inserarea unui element într-o mulţime pe poziţia **p**

Programul C	Rularea programului

```c
#include<stdio.h>
int main(void)
{ int i,n,v,p,a[50];
  printf( "\n Introdu n = "); scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
    {  printf( " a[%d] = ",i );  scanf("%d",&a[i]);  }
  printf( "\n Introdu valoarea v = " );   scanf( "%d",&v );
  printf( "\n Introdu pozitia p = " );   scanf( "%d",&p );
  for( i = n + 1 ; i > p ; i-- )  a[i] = a[i-1];
  a[p] = v;
  for( i = 0 ; i <= n ; i++ )  printf( " %4d ",a[i] );
}
```

Introdu n= **5**
 a[0] = **1**
 a[1] = **2**
 a[2] = **4**
 a[3] = **5**
 a[4] = **6**
Introdu valoarea v= **3**
Introdu pozitia p= **3**
Sirul nou este:
   1   2   3   4   5   6

Figura 5.11b. Programul C şi rularea acestuia pentru inserarea unui element într-o mulţime pe poziţia **p**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.12. Determinarea numărului de apariţii a unei valori într-o mulţime

Se consideră o mulţime cu **n** elemente şi se doreşte determinarea numărului de apariţii a unei valori **v** în cadrul mulţimii. Algoritmul constă în parcurgerea următorilor paşi:

- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se afişează elementele mulţimii citite anterior;
- se citeşte valoarea **v** căutată;

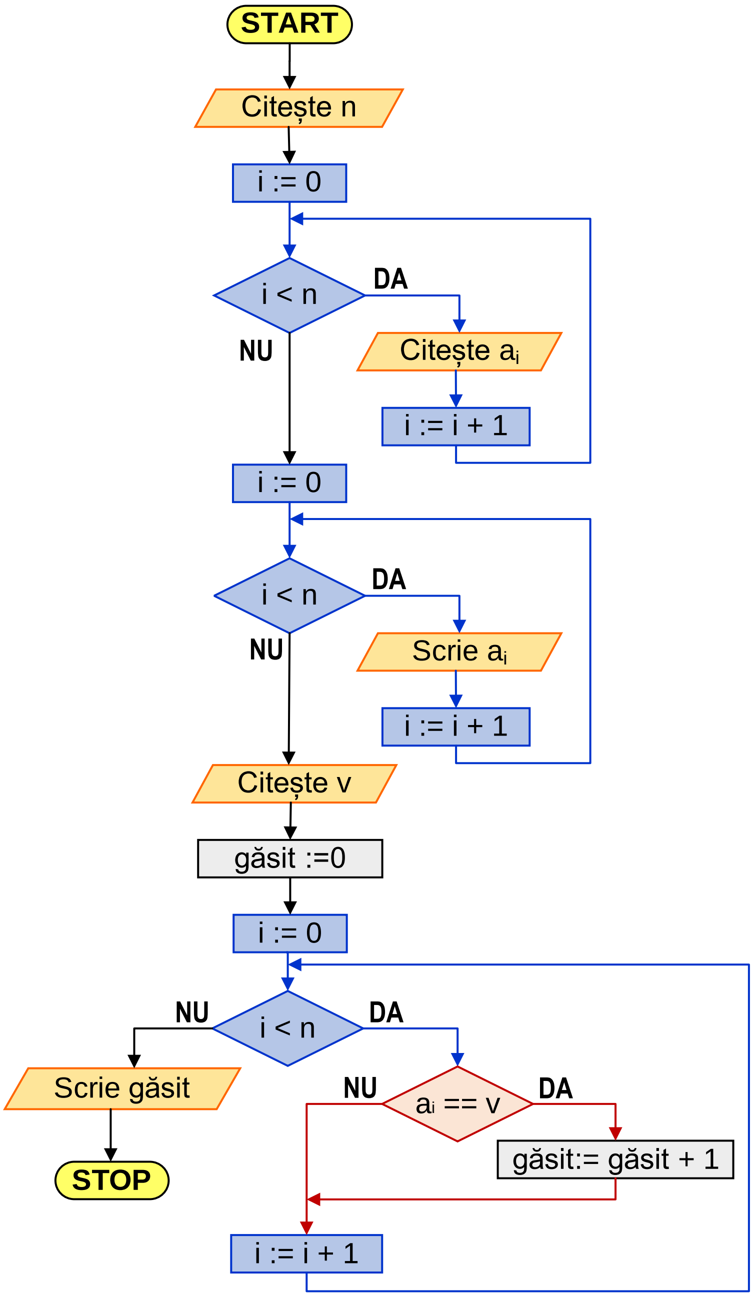
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n **Pentru** i ← 0, n - 1 **Citeşte** a_i **Sfârşit pentru** **Pentru** i ← 0, n - 1 **Scrie** a_i **Sfârşit pentru** **Citeşte** v gasit ← 0 **Pentru** i ← 0, n - 1 **Dacă** v == a_i **atunci** gasit ← gasit + 1 **Sfârşit dacă** **Sfârşit pentru** **Scrie** gasit **Sfârşit**

Figura 5.12a. Reprezentarea algoritmului pentru determinarea numărului de apariţii a unei valori **v** într-o mulţime

- se iniţializează o variabilă ajutătoare, cu numele **gasit**, cu valoarea **0** (zero). Cu ajutorul acesteia se va calcula numărul de apariţii a variabilei **v** în mulţime;
- cu ajutorul unui ciclu cu contor se realizează parcurgerea mulţimii element cu element, pornind de la primul element spre ultimul. Cu ajutorul unei instrucţiuni de decizie se verifică pentru fiecare element în parte dacă este egal cu valoarea **v**. Dacă se constată egalitatea se incrementează valoarea variabilei **gasit**. Dacă nu se constată egalitatea se trece la următorul element din mulţime;
- în final se afişează numărul de apariţii a variabilei **v** în mulţimea considerată prin afişarea valorii variabilei **gasit**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.12a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.12b.

<table>
<tr><th>Programul C</th><th>Rularea programului</th></tr>
<tr><td>
<pre>
#include<stdio.h>
int main(void)
{ int n,i, gasit, a[20], v;
  printf("\n Introduceti n, n = "); scanf("%d",&n);
  for(i = 0 ; i < n ; i++)
   {   printf(" Introduceti a[%d] = ",i); scanf("%d",&a[i]); }
  for(i = 0 ; i < n ; i++)
      printf(" %d",a[i]);
  printf("\n Introduceti v, v = "); scanf("%d",&v);
  gasit = 0;
  for(int i = 0 ; i < n ; i++)
     if(a[i] == v)
              gasit++;
  printf("\n S-au gasit %d elemente!\n", gasit);
}
</pre>
</td><td>
Cazul 1:
Introduceti n, n = 6
Introduceti a[0] = 1
Introduceti a[1] = 2
Introduceti a[2] = 5
Introduceti a[3] = 2
Introduceti a[4] = 3
Introduceti a[5] = 4
 1 2 5 2 3 4
Introduceti v, v = 2

S-au gasit 2 elemente!

Cazul 2:
Introduceti n, n = 6
Introduceti a[0] = 1
Introduceti a[1] = 2
Introduceti a[2] = 3
Introduceti a[3] = 4
Introduceti a[4] = 5
Introduceti a[5] = 6
 1 2 3 4 5 6
Introduceti v, v = 8

S-au gasit 0 elemente!
</td></tr>
</table>

Figura 5.12b. Programul C şi rularea acestuia pentru determinarea numărului de apariţii a unei valori **v** într-o mulţime

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.13. Eliminarea unui element dintr-o mulţime

Se consideră o mulţime (ordonată sau nu) cu **n** elemente şi se doreşte eliminarea unui element din mulţime. Se va obţine o mulţime cu mai puţine elemente, numărul acestora fiind mai mic cu numărul elementelor găsite şi eliminate.

Schema logică

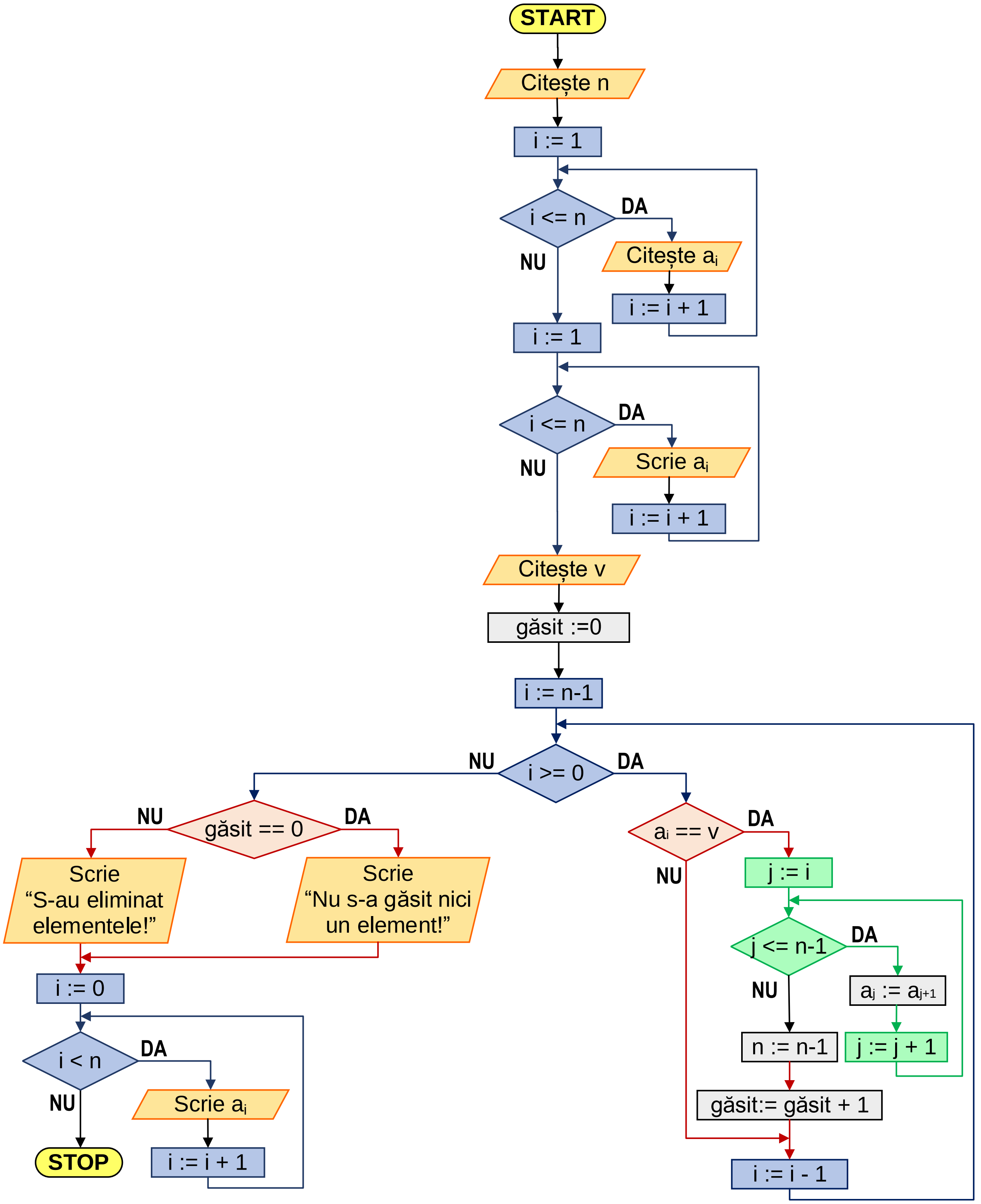


Figura 5.13a. Reprezentarea algoritmului pentru eliminarea unui element dintr-o mulţime

Algoritmul de eliminare constă în parcurgerea următorilor paşi:
- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se afişează elementele mulţimii citite anterior;
- se citeşte valoarea **v** care se doreşte a fi eliminată;
- se iniţializează o variabilă ajutătoare, cu numele **gasit**, cu valoarea **0** (zero). Cu ajutorul acesteia se va calcula numărul de elemente eliminate;
- cu ajutorul unui ciclu cu contor se realizează parcurgerea mulţimii element cu element, pornind de la ultimul element spre primul. Pentru fiecare element al mulţimii se verifică, cu ajutorul unei decizii cu o ramură, dacă valoarea acestuia este egală cu valoarea variabilei ajutătoare **v**. În cazul în care valoarea elementului curent al mulţimii este egală cu valoarea variabilei **v**, se realizează următoarele operaţii:

- deoarece s-a găsit un element care trebuie eliminat, eliminarea acestuia se realizează prin mutarea cu o poziţie spre primul element al elementelor care au fost verificate până acum, adică a elementelor de la poziţia **i + 1** până la ultimul, astfel că elementul de pe poziţia **i + 1** va „sări” pe poziţia **i**, cel de pe poziţia **i + 2** va „sări” pe poziţia **i + 1**, ş.a.m.d;

- deoarece numărul de elemente a scăzut cu **1**, se decrementează variabila **n**;

- deoarece s-a găsit un element care trebuie eliminat, se incrementează variabila **gasit**.

- în final cu ajutorul unei decizii se verifică valoarea variabilei **gasit**. Dacă aceasta este **0**, înseamnă că şirul nu conţine elemente egale cu valoarea căutată şi se afişează un mesaj corespunzător. Dacă valoarea variabilei **gasit** este diferită de zero (pozitivă) atunci se afişează numărul de elemente eliminate şi utilizând un ciclu cu contor se afişează elementele mulţimii.

Limbajul pseudocod

```
Început
   Citeşte n
   Pentru i ← 0, n - 1
      Citeşte a_i
   Sfârşit pentru
   Pentru i ← 0, n - 1
      Scrie a_i
   Sfârşit pentru
   Citeşte v
   gasit ← 0
   Pentru i ← n - 1, 0, -1
      Dacă v == a_i atunci
         Pentru j ← i, n-1
            a_j ← a_j+1
         Sfârşit pentru
         n ← n - 1, gasit ← gasit + 1
      Sfârşit dacă
   Sfârşit pentru
   Dacă gasit == 0 atunci
      Scrie „Nu s-au gasit elemente”
   altfel
      Scrie gasit
   Sfârşit dacă
   Pentru i ← 0, n - 1
      Scrie a_i
   Sfârşit pentru
Sfârşit
```

Figura 5.13a. Reprezentarea algoritmului pentru eliminarea unui element dintr-o mulţime

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 5.13a, iar programul C şi rularea acestuia sunt prezentate în figura 5.13b.

<table>
<tr><th>Programul C</th><th>Rularea programului</th></tr>
<tr><td>

```
#include<stdio.h>
int main(void)
{ int n, i, gasit, a[20], v;
  printf("\n Introduceti n, n = " );  scanf( "%d",&n );
  for( i = 0 ; i < n ; i++ )
  { printf( " Introduceti a[%d] = ",i );  scanf( "%d",&a[i] );  }
  for( i = 0 ; i < n ; i++ )
    printf( " %d",a[i] );
  printf( "\n Introduceti v, v = " ); scanf( "%d",&v );
  gasit = 0;
  for( i = n - 1 ; i >= 0 ; i-- )
    if( a[i] == v )
      { for( int j = i ; j < n - 1; j++ )
            a[j] = a[j+1];
        n--;        gasit++;
      }
  if( gasit == 0 )
        printf( "\n Nu s-a gasit niciun element!" );
  else
        printf( "\n S-au eliminat %d elemente!\n", gasit );
  for( i = 0 ; i < n - 1 ; i++ )
        printf( "  %d",a[i] );
}
```

</td><td>

Cazul 1:
Introduceti n, n = **8**
Introduceti a[0] = **2**
Introduceti a[1] = **3**
Introduceti a[2] = **1**
Introduceti a[3] = **5**
Introduceti a[4] = **4**
Introduceti a[5] = **6**
Introduceti a[6] = **5**
Introduceti a[7] = **9**
2 3 1 5 4 6 5 9
Introduceti v, v = **5**
S-au eliminat 2 elemente!
2 3 1 4 6 9

Cazul 2:
Introduceti n, n = **4**
Introduceti a[0] = **2**
Introduceti a[1] = **3**
Introduceti a[2] = **1**
Introduceti a[3] = **4**
2 3 1 4
Introduceti v, v = **5**
Nu s-a gasit niciun element!

</td></tr>
</table>

Figura 5.13b. Programul C şi rularea acestuia pentru eliminarea unui element dintr-o mulţime

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.14. Determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător

Se consideră o mulţime cu **n** elemente şi se doreşte determinarea celui mai mic număr din mulţime, mai mare decât o valoare impusă **v**. Algoritmul constă în parcurgerea următorilor paşi:
- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se afişează elementele mulţimii citite anterior;
- se citeşte valoarea variabilei **v**;
- cu ajutorul unei instrucţiuni de decizie se verifică dacă variabila **v** este mai mică decât ultimul element al mulţimii **a[n-1]**, existând două posibilităţi:

- dacă variabila **v** este mai mică înseamnă că există un element al mulţimii mai mare sau egal decât **v**. În acest caz, se iniţializează contorul **i** cu valoarea **0**, iar cu ajutorul unei instrucţiuni de ciclare cu test iniţial se verifică pe rând dacă valoarea **v** este mai mare decât valoarea elementului curent al mulţimii. Atunci când valoarea variabilei **v** este mai mică decât elementul curent al mulţimii, se părăseşte ciclul cu test iniţial şi se afişează valoarea variabilei **v** precum şi cea a elementului curent al mulţimii **a[i]**;

- dacă variabila **v** nu este mai mică decât ultimul element al mulţimii, căutarea nu se mai efectuează şi se afişează un mesaj corespunzător.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 5.14a, iar programul C şi rularea acestuia sunt prezentate în figura 5.14b.

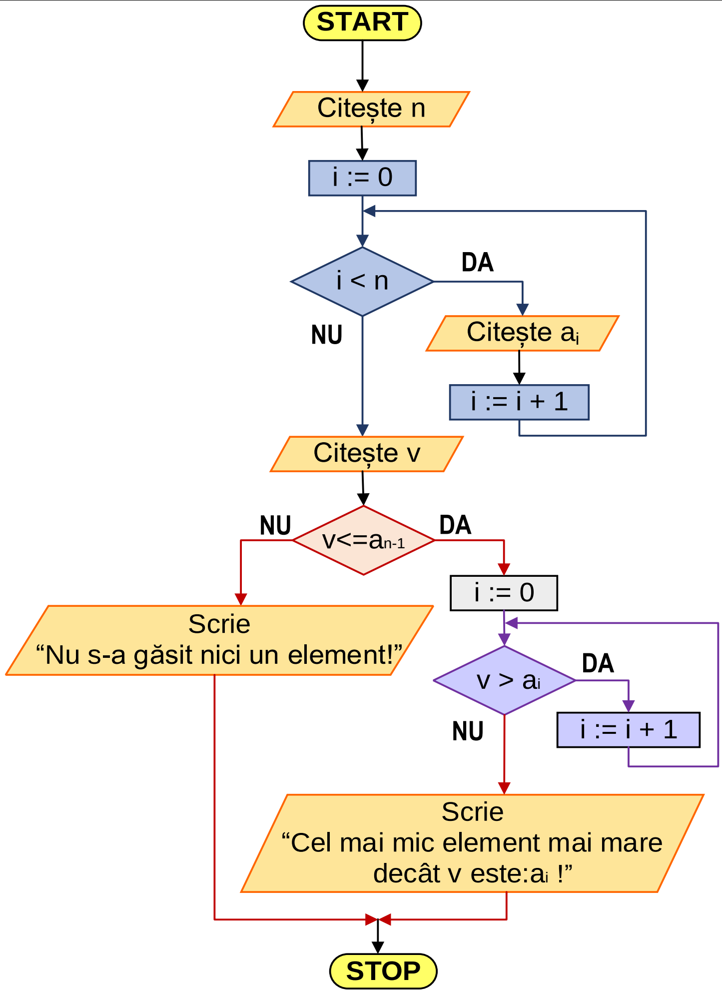
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n **Pentru** i ← 0, n - 1 **Citeşte** a_i **Sfârşit pentru** **Citeşte** v **Dacă** v ≤ a_{n-1} **atunci** i ← 0 **Cât timp** v > a_i **execută** i ← i + 1 **Sfârşit cât timp** **Scrie** a_i **altfel** **Scrie** „Nu s-a gasit” **Sfârşit dacă** **Sfârşit**

Figura 5.14a. Reprezentarea algoritmului pentru determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător

Programul C

```
#include<stdio.h>
int main(void)
{ int n,i; float a[20], v;
  printf("\n Introduceti n, n = ");  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
  { printf(" Introduceti a[%d] = ",i );  scanf("%f",&a[i]);  }
  printf( "\n Introduceti v, v = ");  scanf("%f",&v);
  if( v <= a[n-1] )
   {  i = 0;
      while( v > a[i] )  i++;
      printf( "\n Cel mai mic element mai mare decat %6.3f este %6.3f",v,a[i] );
   }
  else  printf("\n Nu s-a gasit niciun element!");
}
```

Figura 5.14b. Programul C şi rularea acestuia pentru determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător

Rularea programului	
Cazul 1: Introduceti n, n = **6** Introduceti a[0] = **1** Introduceti a[1] = **2** Introduceti a[2] = **3** Introduceti a[3] = **4** Introduceti a[4] = **5** Introduceti a[5] = **6** Introduceti v, v = **3.14159** Cel mai mic element mai mare decat 3.142 este 4.000	**Cazul 2:** Introduceti n, n = **6** Introduceti a[0] = **1** Introduceti a[1] = **2** Introduceti a[2] = **3** Introduceti a[3] = **4** Introduceti a[4] = **5** Introduceti a[5] = **6** Introduceti v, v = **7** Nu s-a gasit niciun element!

Figura 5.14b. Programul C şi rularea acestuia pentru determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător - continuare

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.15. Conversia unui număr din baza 10 într-o bază de la 2 la 9

Într-un sistem de numeraţie poziţional un număr oarecare **N** (cu parte întreagă şi parte fracţionară, separate prin virgulă) se poate scrie sub una din următoarele forme:

$$N = a_{n-1}a_{n-2}...a_1a_0, a_{-1}a_{-2}...a_{(m-1)}a_{-m}$$

$$N = a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + ... + a_1q^1 + a_0q^0 + a_{-1}q^{-1} + a_{-2}q^{-2} + ... + a_{-m}q^{-m}$$

$$N = \sum_{i=-m}^{n-1} a_i q^i$$

unde: **q** reprezintă baza sistemului de numeraţie, $\mathbf{a_i}$ reprezintă cifrele utilizate în sistemul de numeraţie, **n** reprezintă numărul de cifre întregi ale numărului, **m** fiind numărul de cifre fracţionare ale numărului.

În relaţiile de mai sus, cifrele $\mathbf{a_i}$ reprezintă coeficienţii cu care se înmulţesc puterile $\mathbf{q^i}$ ale bazei **q**. Cifra $\mathbf{a_{n-1}}$ este cifra cea mai semnificativă în timp ce cifra $\mathbf{a_{-m}}$ este cifra cea mai puţin semnificativă.

Pentru înţelegerea modului de conversie a părţii întregi a unui număr dintr-o bază de numeraţie în alta se pleacă de la faptul cunoscut că dacă **N** şi **q** sunt numere întregi, există întotdeauna un singur întreg **r** – numit rest (pozitiv) mai mic decât **q** şi un singur întreg **C**, astfel încât:

$$\frac{N}{q} = C + \frac{r}{q},\ 0 \le r < q$$

Pe baza regulii prezentate anterior pentru determinarea algoritmului de conversie a unui număr întreg **N** din baza **p** în baza **q** se porneşte de la expresia numărului **N** scris în baza **q** :

$$N_p = a_{n-1} \cdot q^{n-1} + ... + a_2 \cdot q^2 + a_1 \cdot q^1 + a_0 \cdot q^0$$

Prin împărţire cu baza **q** se obţine : $\frac{N_p}{q} = \underbrace{a_{n-1} \cdot q^{n-2} + ... + a_2 \cdot q^1 + a_1 \cdot q^0}_{Cat} + \underbrace{\frac{a_0}{q}}_{Rest} = C_0 + \frac{r_0}{q}$

astfel : $a_0 = r_0$, respectiv $C_0 = a_{n-1} \cdot q^{n-2} + ... + a_2 \cdot q^1 + a_1 \cdot q^0$.

Se împarte câtul obţinut la **q**, obţinându-se :

$$\frac{C_0}{q} = \underbrace{a_{n-1} \cdot q^{n-3} + ... + a_2 \cdot q^0}_{Cat} + \underbrace{\frac{a_1}{q}}_{Rest} = C_1 + \frac{r_1}{q}$$

adică : $a_1 = r_1$, respectiv $C_1 = a_{n-1} \cdot q^{n-3} + ... + a_2 \cdot q^0$.

Operaţia se continuă până când se obţine un cât egal cu zero.

Algoritmul de conversie a unui număr întreg dintr-o bază în alta poate fi enunţat astfel :
Pentru conversia părţii întregi se realizează împărţirea succesivă a numărului scris în baza p la q (baza în care se doreşte conversia). Astfel, se împarte numărul la bază obţinându-se un cât şi un rest. Se împarte câtul din nou la bază obţinându-se un nou cât şi un nou rest, ş.a.m.d. până când câtul devine 0. Resturile obţinute aşezate în ordine inversă reprezintă cifrele numărului în baza cerută.

Pe baza aspectelor teoretice prezentate anterior, se consideră un număr natural **N** scris în baza **10**. În continuare sunt prezentate algoritmul, schema logică si programul pentru conversia numărului natural din baza **10** în baza **2 - 9**.

- se citeşte numărul natural **N** în baza **10**;
- într-un ciclu cu test final se citeşte baza **b** în care se doreşte conversia (de la **2** la **9**), repetându-se operaţia de citire a valorii variabilei **b** atât timp cât se introduce un număr mai mic ca **2** sau mai mare ca **9**;
- se iniţializează variabila **Nb** cu **0**, respectiv **fact** cu **1**, adică **Nb = 0, fact = 1**; Variabila **Nb** reprezintă numărul natural **N** scris în baza **b**, iar variabila **fact** se utilizează pentru a plasa fiecare cifră a numărului **Nb** în poziţia corespunzătoare;
- cu ajutorul unui ciclu cu test final, se execută următoarele operaţii, atât timp cât **N > 0**:
  - se determină restul împărţirii numărului **N** la baza **b** (notat **rest**), valoarea astfel obţinută reprezintă ultima cifră din numărul **N** scris în baza **b**;
  - se modifică valoarea numărului **N** scris în baza **10** prin atribuirea câtului obţinut prin împărţirea numărului **N** la baza **b**;
  - se modifică valoarea numărului **N** scris în baza **b**, notat **Nb**, cu ajutorul relaţiei **Nb = Nb + fact * rest**;
  - se modifică valoarea variabilei **fact** cu ajutorul relaţiei **fact = fact * 10**. Valoarea variabilei **rest**, obţinută în fiecare etapă se înmulţeşte cu valoarea curentă a variabilei **fact** astfel se poziţionează cifra numărului în baza **b** pe poziţia corespunzătoare în cadrul numărului scris în baza **b**;
- când valoarea variabilei **N** nu mai este strict pozitivă (devine zero), se părăseşte ciclul şi se continuă cu următoarea secvenţă din schema logică, adică se afişează valoarea numărului natural **N** în baza **b**, adică **Scrie Nb**.

Exemplu numeric (tabelul 5.15):

Se consideră **N = 123** şi **b = 2** şi se aplică algoritmul prezentat anterior. Se iniţializează **Nb = 0** şi **fact = 1**.

În prima etapă, se împarte numărul **123** la baza **2**, obţinându-se câtul **61** şi restul **1**. Restul obţinut este ultima cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 0 + 1 * 1 = 1**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 1 * 10 = 10**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **123** la baza **2**, adică **61**;

Tabelul 5.15. Conversia unui număr natural din baza 10 în baza 2 - 9

Valori obţinute pentru: **N = 123, b = 2**						
Etapa	**N**	**[N/b]**	**rest**	**Nb = Nb + fact * rest**	**fact**	**Ecran**
0	123			**0**	**1**	
1	123	61	**1**	Nb = 0 + 1 * 1 = 1	10	
2	61	30	**1**	Nb = 1 + 10 * 1 = 11	100	
3	30	15	**0**	Nb = 11 + 100 * 0 = 11	1000	
4	15	7	**1**	Nb = 11 + 1000 * 1 = 1011	10000	
5	7	3	**1**	Nb = 1011 + 10000 * 1 = 11011	100000	
6	3	1	**1**	Nb = 11011 + 100000 * 1 = 111011	1000000	
7	1	0	**1**	Nb = 111011 + 1000000 * 1 = 1111011	10000000	
						1111011

În următoarea etapă, se împarte numărul **61** la baza **2**, obţinându-se câtul **30** şi restul **1**. Restul obţinut este penultima cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 1 + 10 * 1 = 11**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 10 * 10 = 100**.

Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **61** la baza **2**, adică **30**;

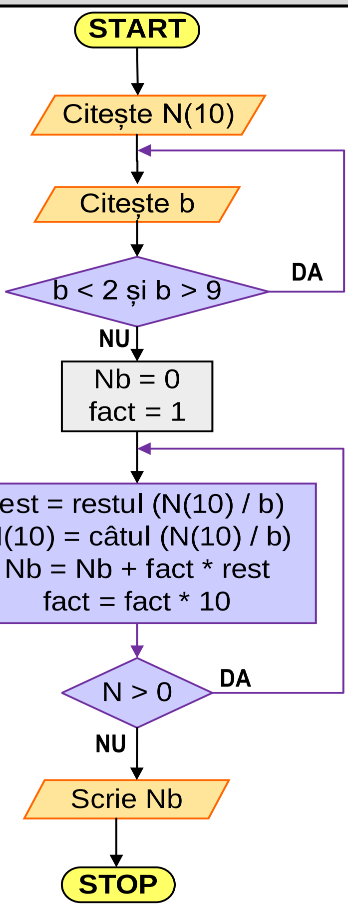
În următoarea etapă, se împarte numărul **30** la baza **2**, obţinându-se câtul **15** şi restul **0**. Restul obţinut este antepenultima cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 11 + 10 * 0 = 11**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 100 * 10 = 1000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **30** la baza **2**, adică **15**;

În următoarea etapă, se împarte numărul **15** la baza **2**, obţinându-se câtul **7** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 11 + 1000 * 1 = 1011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 1000 * 10 = 10000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **15** la baza **2**, adică **7**;

În următoarea etapă, se împarte numărul **7** la baza **2**, obţinându-se câtul **3** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 1011 + 10000 * 1 = 11011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 10000 * 10 = 100000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **7** la baza **2**, adică **3**;

În următoarea etapă, se împarte numărul **3** la baza **2**, obţinându-se câtul **1** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 11011 + 100000 * 1 = 111011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 100000 * 10 = 1000000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **3** la baza **2**, adică **1**;

Schema logică	Limbajul pseudocod




```
Început
  Citeşte N
  NC←N
  Citeşte b
  Cât timp b < 2 SAU b > 9 execută
        Citeşte b
  Sfârşit cât timp
  NB ← 0
  fact ← 1
  Cât timp N > 0 execută
      rest ← N MOD b
      N ← N DIV b
      Nb ← Nb+fract*rest
      fact ← fact*10
  Sfârşit cât timp
  Scrie Nb
Sfârşit
```

Figura 5.15a. Reprezentarea algoritmului pentru calculul conversia unui număr natural din baza 10 în baza 2 - 9

În următoarea etapă, se împarte numărul **1** la baza **2**, obţinându-se câtul **0** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb = Nb + fact * rest**, adică **Nb = 111011 + 1000000 * 1 = 1111011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact = fact * 10 = 1000000 * 10 = 10000000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **1** la baza **2**, adică **0**;

Deoarece **N = 0**, condiţia **N > 0** nu se mai îndeplineşte, se continuă pe ramura **"NU"** a condiţiei **N > 0** şi se execută secvenţa următoare ciclului cu test iniţial, adică **Scrie Nb**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.15a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.15b.

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { int N,Nc, b, Nb, fact, rest; printf("\n Introduceti numarul N10 = "); scanf("%d",&N); Nc=N; do{ printf("\n Introducet baza, b = "); scanf("%d",&b); }while(b < 2 && b > 9); Nb = 0; fact = 1; do{ rest = N % b; N = N / b; Nb = Nb + fact * rest; fact *= 10; }while(N > 0); printf("\n N(10) = %d, N(%d) = %d",Nc,b,Nb); }</pre>	**Cazul 1:** Introduceti numarul N10 = **123** Introducet baza, b = **2** N(10) = 123, N(2) = 1111011 **Cazul 2:** Introduceti numarul N10 = **123** Introducet baza, b = **8** N(10) = 123, N(8) = 173

Figura 5.15b. Programul C şi rularea acestuia pentru calculul conversia unui număr natural din baza 10 în baza 2 - 9

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.16. Reversul unui număr

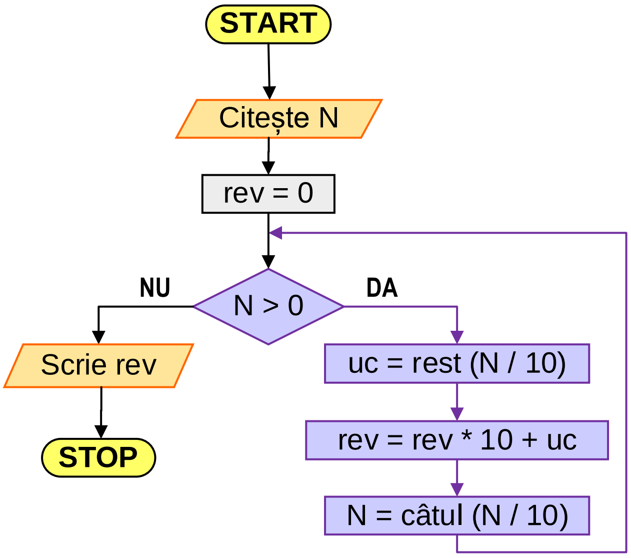
Se consideră un număr natural **N**. În continuare sunt prezentate algoritmul, schema logică si programul pentru determinarea reversului unui număr.

Algoritmul este următorul:

- se citeşte numărul natural **N**;
- se iniţializează numărul revers cu zero, adică **rev = 0**;
- cu ajutorul unui ciclu cu test iniţial, atât timp cât **N > 0**, se realizează următoarele operaţii:
    - se determină ultima cifră (notată **uc**) a numărului **N** prin împărţirea cu rest a acestuia la **10**, restul obţinut fiind ultima cifră a numărului;
    - se formează reversul numărului **N** cu ajutorul relaţiei: **rev = rev * 10 + uc**;
    - se modifică valoarea numărului natural **N**, adică se elimină ultimă cifră, obţinându-se un număr mai scurt cu o cifră. Acest lucru se realizează prin atribuirea către variabila **N** a valorii câtului împărţirii numărului natural **N** la **10**;
    - se revine la evaluarea valorii numărului natural **N** nou obţinut prin revenirea la condiţia **N > 0**;
- când valoarea variabilei **N** nu mai este strict pozitivă (devine zero), se părăseşte ciclul şi se continuă cu următoarea secvenţă din schema logică, adică se afişează valoarea numărului revers, adică **Scrie rev**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.16a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.16b.

Schema logică / **Limbajul pseudocod**




```
Început
   Citeşte N
   rev ← 0
   Cât timp N > 0 execută
      uc ← N MOD 10
      rev ← rev * 10 + uc
      N ← N DIV 10
   Sfârşit cât timp
   Scrie rev
Sfârşit
```

Figura 5.16a. Reprezentarea algoritmului pentru formarea reversului numărului

Programul C

```c
#include<stdio.h>
int main(void)
{ long int N,rev, uc;
  printf("\n Introduceti N, N = ");
  scanf("%d",&N);
  rev = 0;
  while( N > 0 )
    {  uc = N % 10;
       rev = rev*10 + uc;
       N = N / 10;
    }
   printf( "\n Inversul nr. este %d",rev );
}
```

Rularea programului

Introduceti N, N = **12345**
Inversul nr. este 54321

Figura 5.16b. Programul C şi rularea acestuia pentru formarea reversului numărului

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Exemplu numeric (tabelul 5.16):

Se consideră **N = 12345** şi se aplică algoritmul prezentat anterior.

Se iniţializează **rev = 0**.

În prima etapă, se obţine ultima cifră a numărului **12345**, prin împărţirea cu rest a numărului **12345** la **10**. Rezultatul obţinut este **5** şi se modifică valoarea variabilei **rev** conform relaţiei: **rev=rev*10+uc**, deci **rev = 0 * 10 + 5 = 5**. În continuare, se modifică valoarea numărului **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1234**, deci **N = 1234**;

Tabelul 5.16. Determinarea reversului unui număr natural

Valori obţinute pentru: **N = 12345**					
Etapa	**N**	**uc**	**rev**	**[N/10]**	**Ecran**
0	12345		0		
1	12345	5	rev = 0 * 10 + **5** = 5	1234	
2	1234	4	rev = 5 * 10 + **4** = 54	123	
3	123	3	rev = 54 * 10 + **3** = 543	12	
4	12	2	rev = 543 * 10 + **2** = 5432	1	
5	1	1	rev = 5432 * 10 + **1** = 54321	0	
					54321

În a doua etapă, se obţine ultima cifră a numărului natural **N** (**N = 1234**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **4**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev=rev*10+uc**, deci **rev = 5 * 10 + 4 = 54**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **123**, deci **N = 123**;

În a treia etapă, se obţine ultima cifră a numărului natural **N** (**N = 123**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **3**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev=rev*10+uc**, deci **rev = 54 * 10 + 3 = 543**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **12**, deci **N = 12**;

În a patra etapă, se obţine ultima cifră a numărului natural **N** (**N = 12**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **2**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev=rev*10+uc**, deci **rev = 543 * 10 + 2 = 5432**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1**, deci **N = 1**;

În a cincea etapă, se obţine ultima cifră a numărului natural **N** (**N = 1**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **1**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev=rev*10+uc**, deci **rev = 5432 * 10 + 1 = 54321**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **0**, deci **N = 0**.

Deoarece condiţia **N > 0** nu se mai îndeplineşte, se continuă pe ramura **"NU"** a condiţiei **N > 0** şi se execută secvenţa următoare ciclului cu test iniţial, adică **Afişare rev**.

5.17. Calculul sumei cifrelor unui număr natural

Se consideră un număr natural **N** şi se doreşte calculul sumei cifrelor numărului. Algoritmul este următorul:
- se citeşte numărul natural **N**;
- se iniţializează suma cifrelor **S** cu zero, adică **S = 0**;
- cu ajutorul unui ciclu cu test iniţial, atât timp cât **N > 0**, se realizează următoarele operaţii:
  - se determină ultima cifră (notată **uc**) a numărului **N** prin împărţirea cu rest a acestuia la **10**, restul obţinut fiind ultima cifră a numărului;
  - se adaugă valoarea ultimei cifre la sumă: **S = S + uc**;
  - se modifică valoarea numărului natural **N**, adică se elimină ultimă cifră, obţinându-se un număr mai scurt cu o cifră. Acest lucru se realizează prin atribuirea către variabila **N** a valorii câtului împărţirii numărului natural **N** la **10**;
  - se revine la evaluarea valorii numărului natural **N** nou obţinut prin revenirea la condiţia **N > 0**;
- când valoarea variabilei **N** nu mai este strict pozitivă (devine zero), se părăseşte ciclul şi se continuă cu următoarea secvenţă din schema logică, adică se afişează valoarea calculată a sumei, adică **Scrie S**.

Exemplu numeric (tabelul 5.17):

Se consideră **N = 12345** şi se aplică algoritmul prezentat anterior. Se iniţializează **S = 0**.

În prima etapă, se obţine ultima cifră a numărului **12345**, prin împărţirea cu rest a numărului **12345** la **10**. Rezultatul obţinut este **5** şi se adună la valoarea sumei **S** a cărui valoare iniţială este **0**, deci **S = 0 + 5 = 5**. În continuare, se modifică valoarea numărului **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1234**, deci **N = 1234**;

Tabelul 5.17. Calculul sumei cifrelor unui număr natural

Valori obţinute pentru: **N = 12345**					
Etapa	**N**	**uc**	**S**	**[N / 10]**	**Ecran**
0	12345		0		
1	12345	5	S = 0 + **5** = 5	1234	
2	1234	4	S = 5 + **4** = 9	123	
3	123	3	S = 9 + **3** = 12	12	
4	12	2	S = 12 + **2** = 14	1	
5	1	1	S = 14 + **1** = 15	0	
					15

În a doua etapă, se obţine ultima cifră a numărului natural **N** (**N = 1234**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **4**. Se adaugă această valoare la suma **S**, deci **S = 5 + 4**, adică **S = 9**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **123**, deci **N = 123**;

În a treia etapă, se obţine ultima cifră a numărului natural **N** (**N = 123**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **3**. Se adaugă această valoare la suma **S**, deci **S = 9 + 3**, adică **S = 12**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **12**, deci **N = 12**;

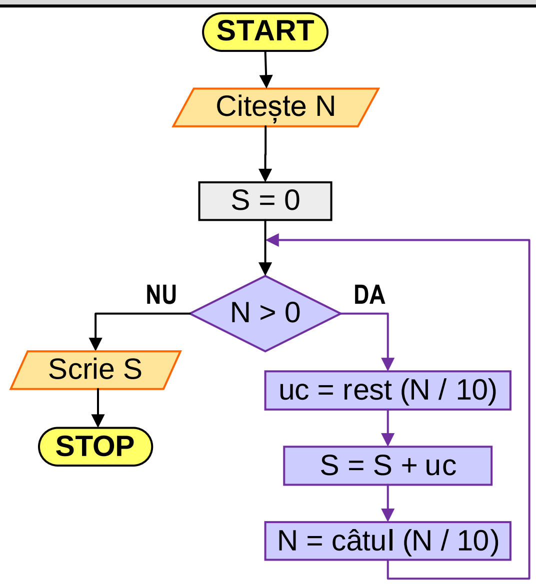
În a patra etapă, se obţine ultima cifră a numărului natural **N** (**N = 12**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **2**. Se adaugă această valoare la suma **S**, deci **S = 12 + 2**, adică **S = 14**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1**, deci **N = 1**;

În a cincea etapă, se obţine ultima cifră a numărului natural **N** (**N = 1**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **1**. Se adaugă această valoare la suma **S**, deci **S = 14 + 1**, adică **S = 15**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **0**, deci **N = 0**;

Deoarece condiţia **N > 0** nu se mai îndeplineşte, se continuă pe ramura **"NU"** a condiţiei **N > 0** şi se execută secvenţa următoare ciclului cu test iniţial, adică **Scrie S**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.17a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.17b.

Schema logică	Limbajul pseudocod




```
Început
   Citeşte N
   S ← 0
   Cât timp N > 0 execută
      uc ← N MOD 10
      S ← S + uc
      N ← N DIV 10
   Sfârşit cât timp
   Scrie S
Sfârşit
```

Figura 5.17a. Reprezentarea algoritmului pentru calculul sumei cifrelor unui număr natural

Programul C	Rularea programului

```c
#include<stdio.h>
int main(void)
{ long int N,S, uc;
  printf( "\n Introduceti N, N = ");  scanf( "%d",&N );
  S = 0;
  while( N > 0 )
   { uc = N % 10;
     S = S + uc;
     N = N / 10;
   }
   printf( "\n Suma cifrelor este S = %d",S );
}
```

Introduceti N, N = **12345**

Suma cifrelor este S = 15

Figura 5.17b. Programul C şi rularea acestuia pentru calculul sumei cifrelor unui număr natural

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.18. Verificarea CNP-ului (Codul Numeric Personal)

Codul numeric personal (**CNP**) este un cod numeric de 13 cifre, unic, atribuit fiecărei persoane născute în România. Codul numeric personal se atribuie fiecărei persoane la naştere şi este specificat în certificatul de naştere, actele de identitate (buletin sau carte de identitate) cât şi în permisul de conducere. Codul numeric personal a fost introdus ca element obligatoriu printr-un decret prezidenţial în 2 martie 1978. Codul numeric personal conţine 13 cifre, astfel:

S	A	A	L	L	Z	Z	J	J	N	N	N	C

S : primă cifră defineşte sexul (sex bărbătesc / sex femeiesc), astfel: 1 / 2 : născuți între 1 ianuarie 1900 și 31 decembrie 1999; 3 / 4 - născuți între 1 ianuarie 1800 și 31 decembrie 1899; 5 / 6 - născuți între 1 ianuarie 2000 și 31 decembrie 2099; 7 / 8 - pentru persoanele străine rezidente în România;
AA : număr format din două cifre care reprezintă ultimele două cifre din anul naşterii persoanei;
LL : număr format din două cifre care reprezintă luna naşterii persoanei;
ZZ : repezintă ziua naşterii persoanei. Pentru zilele de la 1 la 9 se adaugă un 0 înaintea zilei;
JJ : reprezintă un număr format din două cifre care reprezintă codul judeţului sau sectorului (pentru municipiul Bucureşti), astfel: 01–Alba, 02–Arad, 03–Argeş, 04–Bacău, 05–Bihor, 06–Bistriţa-Năsăud, 07–Botoşani, 08–Braşov, 09–Brăila, 10–Buzău, 11–Caraş-Severin, 12–Cluj, 13–Constanţa, 14–Covasna, 15–Dâmboviţa, 16–Dolj, 17–Galaţi, 18–Gorj, 19–Harghita, 20–Hunedoara, 21–Ialomiţa, 22–Iaşi, 23–Ilfov, 24–Maramureş, 25–Mehedinţi, 26–Mureş, 27–Neamţ, 28–Olt, 29–Prahova, 30–Satu Mare, 31–Sălaj, 32-Sibiu, 33–Suceava, 34–Teleorman, 35–Timiş, 36–Tulcea, 37–Vaslui, 38–Vâlcea, 39–Vrancea, 40–Bucureşti, 41:46 – Bucureşti – Sectorul 1:6, 51–Călăraşi, 52–Giurgiu.
NNN : număr format din trei cifre (din intervalul 001 – 999), care se împart în judeţ birourilor de Evidenţă a Persoanei, astfel încât un anumit număr din interval să fie alocat unei singure persoane, într-o anumită zi.
C : cifră de control, aflată în relaţie cu celelalte cifre ale **CNP**-ului. Valoarea cifrei de control se calculează astfel: fiecare cifră din primele 12 cifre ale **CNP**-ului se înmulţeşte cu fiecare cifră de pe aceeaşi poziţie din numărul '**279146358279**', rezultatele se însumează, valoarea obţinută se împarte cu rest la 11. Dacă restul este 10 acesta se consideră ca fiind 1. Dacă cifra de control este egală cu restul obţinut anterior atunci **CNP** este valid, în caz contrar **CNP** este invalid.

Algoritmul este următorul:
1. Cu ajutorul unui ciclu cu contor se citesc cele **13** cifre ale **CNP**;
2. Se iniţializează suma **S** cu valoarea **0**;
3. Cu ajutorul unui ciclu cu contor, în care contorul ia valori de la **0** la **11** (adică se utilizează primele **12** cifre ale **CNP**), se calculează suma, cu ajutorul relaţiei **S = S + cnp[i]*sirc[i]**;
4. Se împarte cu rest suma obţinută anterior la **11**. Dacă restul obţinut este **10**, atunci variabila control primeşte valoarea 1, altfel variabila control are valoarea restului împărţirii sumei la **11**;
5. Dacă valoarea variabilei control este egală cu ultima cifră a **CNP**-ului atunci acesta este valid, în caz contrar **CNP**-ul este invalid.

Exemplul numeric:
1. Se citeşte un **CNP** cu valoarea **1570330121114**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12
CNP	**1**	**5**	**7**	**0**	**3**	**3**	**0**	**1**	**2**	**1**	**1**	**1**	**4**
Şir de verificare	2	7	9	1	4	6	3	5	8	2	7	9	
Produse parţiale	2	35	63	0	12	18	0	5	16	2	7	9	
Suma produselor parţiale (S) =		**169**		Restul împărţirii sumei la 11 (S%11) =									**4**

În acest caz cifra de control al **CNP**-ului (ultima cifră) este egală cu restul obţinut prin împărţirea sumei la 11, deci **CNP**-ul este **valid**.

2. Se citeşte un **CNP** cu valoarea **1570330121111**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la **11**.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12
CNP	**1**	**5**	**7**	**0**	**3**	**3**	**0**	**1**	**2**	**1**	**1**	**1**	**1**
Şir de verificare	2	7	9	1	4	6	3	5	8	2	7	9	
Produse parţiale	2	35	63	0	12	18	0	5	16	2	7	9	
Suma produselor parţiale (S) =		**169**		Restul împărţirii sumei la 11 (S%11) =									**4**

În acest caz cifra de control al **CNP**-ului nu este egală cu restul obţinut prin împărţirea sumei la 11, deci **CNP nu** este **valid**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.18a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.18b.

Schema logică

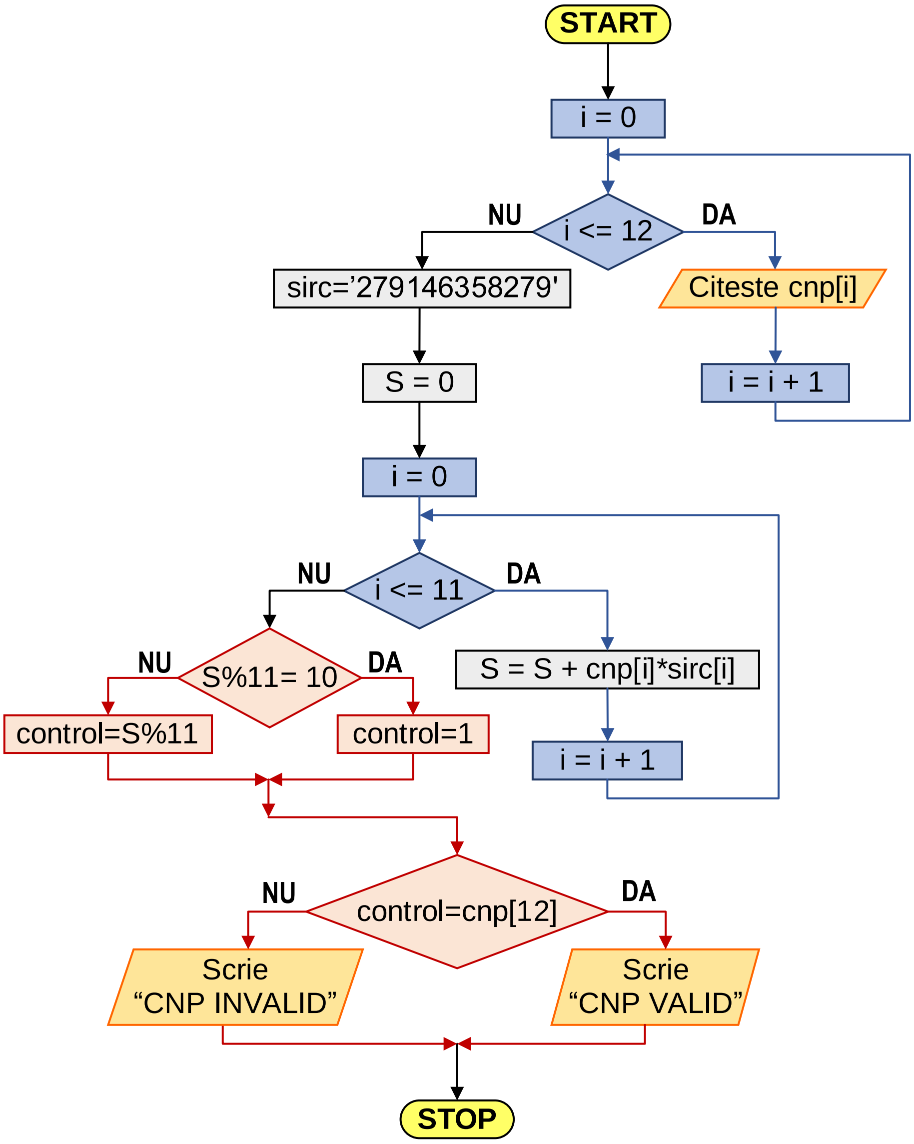


Figura 5.18a. Reprezentarea algoritmului pentru verificarea **CNP**-ului

Limbajul pseudocod

```
Început
   Pentru i ← 1,13
      Citeşte cnp_i
   Sfârşit pentru
   sirc ← [ 2,7,9,1,4,6,3,5,8,2,7,9];
   S ← 0
   Pentru i ← 1,12
      S ← S + cnp_i * sirc_i
   Sfârşit pentru
   Dacă S MOD 11 = 10 atunci
        control ← 1
   altfel
        control ← S MOD 11
   Sfârşit dacă
   Dacă control = cnp_13 atunci
        Scrie „CNP VALID”
   altfel
        Scrie „CNP INVALID”
   Sfârşit dacă
Sfârşit
```

Figura 5.18a. Reprezentarea algoritmului pentru verificarea **CNP**-ului - continuare

Programul C	Rularea programului
(see code below)	Introduceti CNP-ul: **1570330121114** CNP VALID!!! Introduceti CNP-ul: **1570330121111** CNP INVALID!!!

```c
#include<stdio.h>
#include<conio.h>
int main(void)
{  int sirc[12] = {2,7,9,1,4,6,3,5,8,2,7,9};
   int cnp[13], i, S = 0, control;  char c;
   printf("\n Introduceti CNP-ul:");
   for( i = 0 ; i <= 12 ; i++ )
      { c = getche(); cnp[i] = c - '0';  }
   for( i = 0 ; i <= 11 ; i++ )
   S += cnp[i] * sirc[i];
   if( S%11 == 10 )
        control=1;
   else
        control=S%11;
   if( control == cnp[12] )
        printf("\n CNP VALID!!!");
   else
        printf("\n CNP INVALID!!!");
}
```

Figura 5.18b. Programul C şi rularea acestuia pentru calculul sumei cifrelor unui număr natural

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.19. Verificarea codului de pe card

Majoritatea cardurilor bancare sunt identificate printr-un număr de 16 cifre. În ordine, de la stânga la dreapta acestea reprezintă:

S	B	B	B		B	B	N	N		N	N	N	N		N	N	N	C

S : reprezintă tipul cardului (4 – VISA, 5 – MasterCard) sau Major Industry Identifier (MII);
BBBBB : reprezintă codul băncii emitente;
NN ... N : reprezintă numărul contului;
C : reprezintă cifra de control.

Verificarea codului de pe card se realizează conform algoritmului lui Luhn:
Pas 1: Se înmulţeşte fiecare cifră din codul de card cu ponderea sa. Dacă un card are un număr par de cifre, prima cifră are ponderea 2, dacă nu, cifra are ponderea 1. În continuare, ponderile cifrelor alternează 1,2,1,2;
Pas 2: Dacă o cifră are o valoare ponderată mai mare decât 9, se scade 9 din valoarea ei;
Pas 3: Se adună toate valorile ponderate pentru primele 15 cifre şi se calculează restul împărţirii la 10 (MODULO 10);
Pas 4: Un cod de card este valid dacă rezultatul operatiei MODULO 10 este egal cu valoarea cifrei de control **C**.

Algoritmul este următorul:
1. Cu ajutorul unui ciclu cu contor se citesc cele 16 cifre ale codului;
2. Se iniţializează suma **S** cu valoarea **0**;
3. Cu ajutorul unui ciclu cu contor, în care contorul ia valori de la 0 la 15, se extrage fiecare cifră din cod, se înmulţeşte cu ponderea, iar dacă valoarea obţinută este mai mare decât 9, se scade 9 din valoarea obţinută. Cu noua valoare se calculează suma, cu ajutorul relaţiei **S = S + cod[i]**.
4. Se împarte cu rest suma obţinută anterior la 10;
5. Dacă restul împărţirii este egal cu zero atunci codul este valid, în caz contrar codul este invalid.

Exemplul numeric:
1. Se citeşte un cod cu valoarea **5123456789123457**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cod card	**5**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1**	**2**	**3**	**4**	**5**	**7**
Ponderea	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
Produs parţial	10	1	4	3	8	5	12	7	16	9	2	2	6	4	10	7
Produs parţial recalculat	1	1	4	3	8	5	3	7	7	9	2	2	6	4	1	7
Suma produselor parţiale (S)	**70**			Restul împărţirii sumei la 10 (S%10) =												**0**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 10, deci codul este valid.

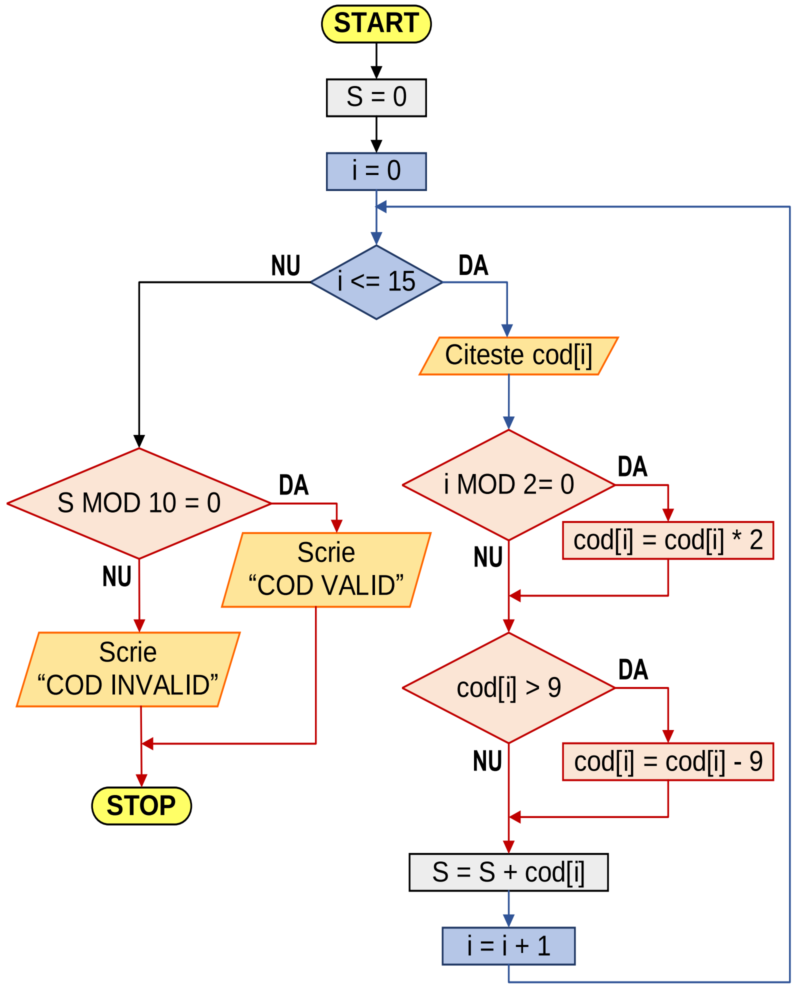
2. Se citeşte un cod cu valoarea **4123456789123455**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cod card	**4**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1**	**2**	**3**	**4**	**5**	**5**
Ponderea	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
Produs parţial	8	1	4	3	8	5	12	7	16	9	2	2	6	4	10	5
Produs parţial recalculat	8	1	4	3	8	5	3	7	7	9	2	2	6	4	1	5
Suma produselor parţiale (S)	**75**			Restul împărţirii sumei la 10 (S%10) =												**5**

În acest caz cifra de control al codului nu este egală cu restul obţinut prin împărţirea sumei la 10, deci codul este invalid.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.19a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.19b.

Schema logică	Limbajul pseudocod




```
Început
  S ← 0
  Pentru i ← 1,16
     Citeşte codᵢ
     Dacă i + 1 MOD 2 = 0 atunci
        codᵢ ← codᵢ * 2
     Sfârşit dacă
     Dacă codᵢ > 9 atunci
        codᵢ ← codᵢ - 9
     Sfârşit dacă
     S ← S + codᵢ
  Sfârşit pentru
  Dacă S MOD 10 = 0 atunci
     Scrie „COD VALID”
  altfel
     Scrie „COD INVALID”
  Sfârşit dacă
Sfârşit
```

Figura 5.19a. Reprezentarea algoritmului pentru verificarea codului de pe card

Programul C

```c
#include<stdio.h>
#include<conio.h>
int main(void)
{ int cod[16], i, S=0; char c;
  printf("\n Introduceti codul:");
  for( i = 0 ; i <= 15 ; i++ )
   { c = getche(); cod[i] = c - '0';
     if( i %2 == 0 )   cod[i] = cod[i] * 2;
     if( cod[i] > 9 )   cod[i] = cod[i] - 9;
     S = S + cod[i];
    }
  if( S%10 == 0 )
     printf("\n COD VALID!!!");
  else
     printf("\n COD INVALID!!!");
}
```

Rularea programului

Rularea programului:
Introduceti codul: **5123456789123457**
COD VALID!!!

Introduceti codul: **4123456789123455**
COD INVALID!!!

Figura 5.19b. Programul C şi rularea acestuia pentru verificarea codului de pe card

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

5.20. Verificarea codului ISBN

Codul **ISBN** (**I**nternational **S**tandard **B**ook **N**umber) este un cod internaţional de identificare a cărţilor, definit prin ISO 2108. Este format din 13 cifre grupate în 5 segmente de lungimi variabile, separate de cratimă, cu următoarele specificaţii:
- prefixul 978 : reprezintă producţia editorială de carte la nivel internaţional;
- codul de ţară : indică grupul naţional, lingvistic sau geografic şi desemnează limba editorului şi nu limba în care este publicată cartea. Pentru România se utilizează 973 sau 606;
- codul editurii: identifică editorul documentului, lungimea acestuia variază în funcţie de numărul lucrărilor publicate de editor;
- numărul de identificare a documentului : se utilizează pentru numerotarea documentului printre publicaţiile editorului;
- cifra de control : reprezintă ultima cifră a codului **ISBN** cu ajutorul căreia se verifică validitatea codului **ISBN**.

Există două categorii de coduri **ISBN**, **ISBN-10** mai vechi care nu conţine prefixul 978 şi codul **ISBN-13**.

A. Algoritmul de validare al unui cod **ISBN-10**:

A.1. Se citeşte codul. Se elimină spaţiile şi cratimele. Ultimul caracter se ignoră;

A.2. Se înmulţeşte fiecare cifră cu ponderea asociată ei, ponderea este dată sub forma **11** – poziţia cifrei. Se adună valorile obţinute;

A.3. Se determină valoarea variabilei **control**, astfel: dacă caracterul de control este ‚**X**’, atunci valoarea variabilei **control** este **10**, altfel valoarea variabilei **control** este egală cu valoarea cifrei de control;

A.4. Se verifică validitatea codului comparând restul împărţirii sumei obţinute la **11** (modulo 11) cu cifra de control. Dacă cele două valori sunt egale, codul este valid, iar dacă nu sunt egale, codul este invalid.

Exemplul numeric:

1. Se citeşte un cod cu valoarea **973-356-432-1**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**3**	**5**	**6**	**4**	**3**	**2**	**1**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	21	30	30	16	9	4	
Suma produselor parţiale S =	**287**		Restul împărţirii sumei S%11 =							**1**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **valid**.

2. Se citeşte un cod cu valoarea **973-456-432-5**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**4**	**5**	**6**	**4**	**3**	**2**	**5**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	28	30	30	16	9	4	
Suma produselor parţiale S =	**294**		Restul împărţirii sumei S%11 =							**8**

În acest caz cifra de control al codului nu este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **invalid**.

3. Se citeşte un cod cu valoarea **973-456-433-X**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**4**	**5**	**6**	**4**	**3**	**3**	**X**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	28	30	30	16	9	6	
Suma produselor parţiale S =	**296**		Restul împărţirii sumei S%11 =							**10**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **valid**.

4. Se citeşte un cod cu valoarea **973-456-439-0**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**4**	**5**	**6**	**4**	**3**	**9**	**0**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	28	30	30	16	9	18	
Suma produselor parţiale S =	**308**		Restul împărţirii sumei S%11 =							**0**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **valid**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.20a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.20b.

Schema logică

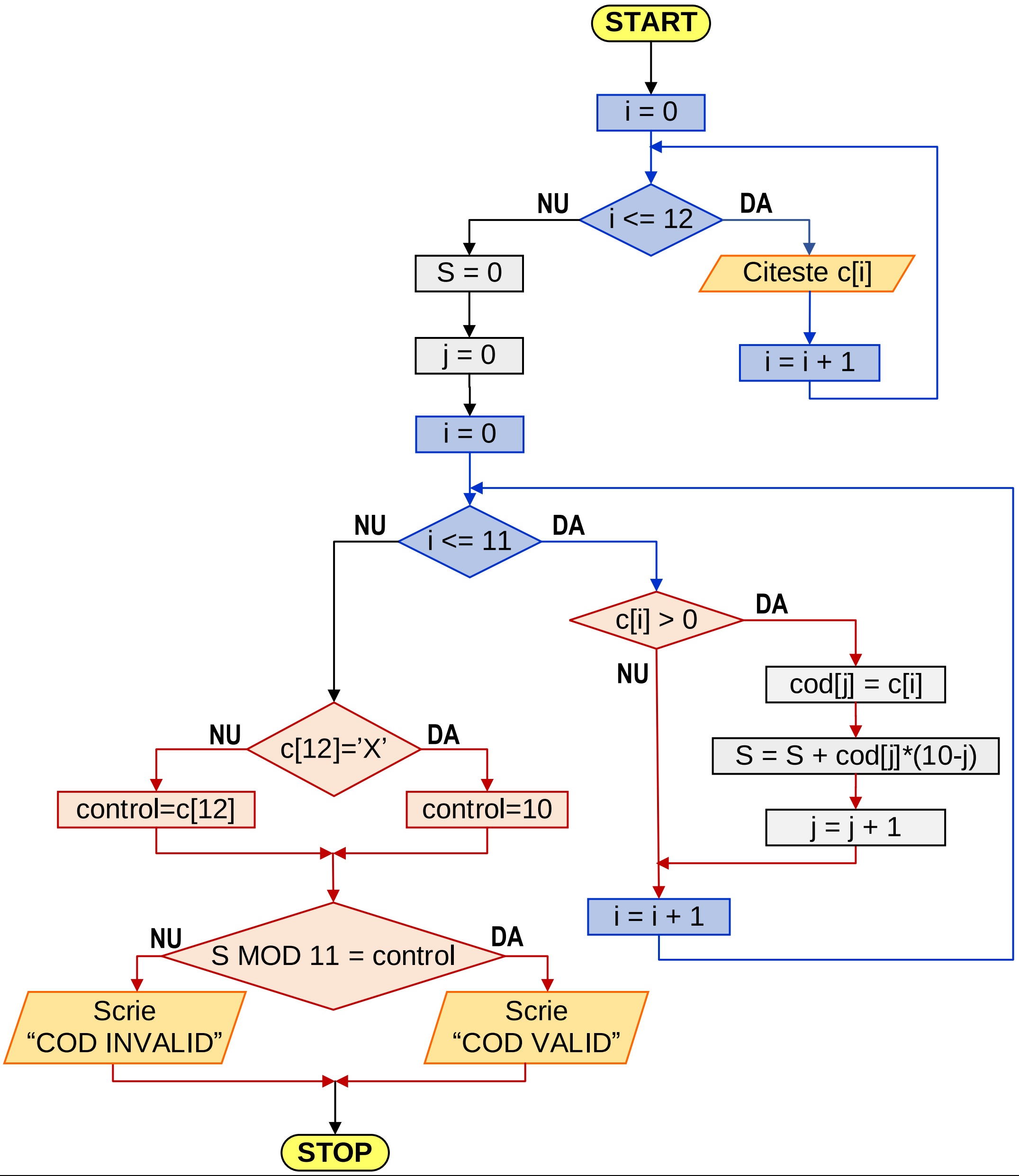


Figura 5.20a. Reprezentarea algoritmului pentru verificarea codului **ISBN-10**

Limbajul pseudocod

```
Început
    Pentru i ← 1, 13
        Citeşte c_i
    Sfârşit pentru
    S ← 0;  j ← 1
    Pentru i←1,12
        Dacă c_i >0 atunci
            cod_j ← c_i ;  S ← S + cod_j *(11 - j) ;  j ← j + 1
        Sfârşit dacă
    Sfârşit pentru
    Dacă c_13 = 'X' atunci  control ← 10
        altfel    control ← c(13)
    Sfârşit dacă
    Dacă S MOD 11=control atunci   Scrie „COD VALID"
        altfel    Scrie „COD INVALID"
    Sfârşit dacă
Sfârşit
```

Figura 5.20a. Reprezentarea algoritmului pentru verificarea codului **ISBN-10** - continuare

Programul C	Rularea programului
`#include<stdio.h>` `#include<conio.h>` `int main(void)` `{ int cod[13], i, j=0, S=0, control;  char c[13];` `  printf("\n Introduceti codul:");` `  for( i = 0 ; i <= 12 ; i++ )   c[i] = getche();` `  for( i = 0 ; i <= 11 ; i++ )` `     if( c[i] - '0' >= 0 )` `        { cod[j] = c[i] - '0'; S = S + cod[j] * (10 - j);  j++; }` `  if( c[12] == 'X' )  control = 10;` `  else   control = c[12] - '0';` `  if( S%11 == control )  printf( "\n COD VALID!!!" );` `  else   printf( "\n COD INVALID!!!" );` `}`	Introduceti codul: **973-356-432-1** COD VALID!!! Introduceti codul: **973-456-432-5** COD INVALID!!!

Figura 5.20b. Programul C şi rularea acestuia pentru pentru verificarea codului **ISBN-10**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

B. Algoritmul de validare al unui cod **ISBN-13**:

B.1. Se citeşte codul. Se elimină spaţiile şi cratimele. Ultimul caracter se ignoră;

B.2. Se înmulţeşte fiecare cifră cu ponderea asociată ei, ponderea se atribuie pentru fiecare cifră, începând cu prima cifră, sub forma 1,3,1,3.... şi se adună valorile obţinute;

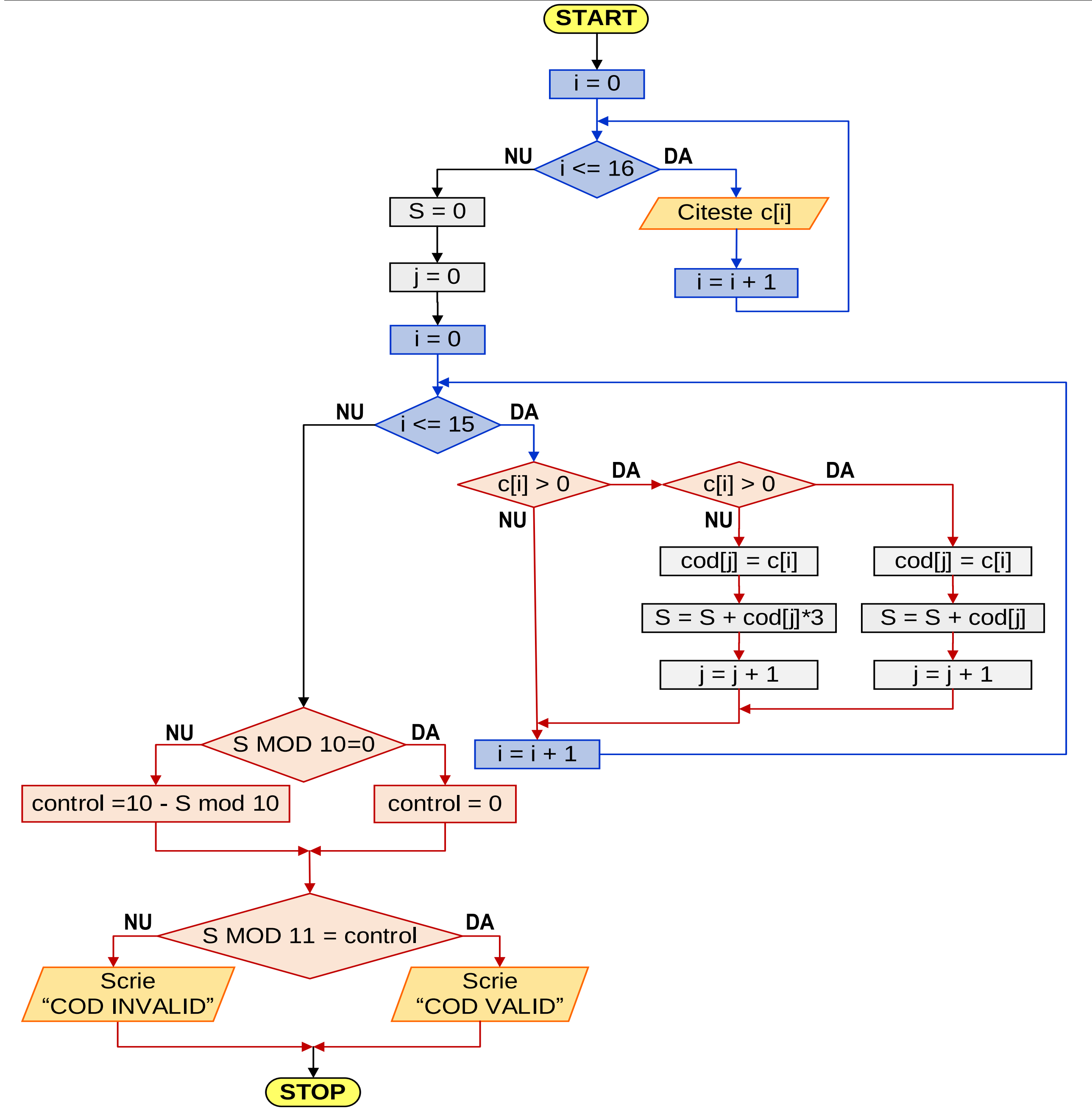
B.3. Se împarte suma obţinută la 10 şi se extrage restul împărţirii (modulo 10);

B.4. Dacă restul este 0 atunci variabila **control** trebuie să fie 0. Dacă restul este diferit de 0, atunci variabila **control** se obţine scăzând din 10 restul obţinut;

B.5. Se verifică validitatea codului comparând variabila **control** cu cifra de control, astfel: pentru un **ISBN-13** valid variabila **control** rezultată va trebui să fie egală cu ultima cifră a codului (cifra 13).

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.20c, iar programul C aferent și rularea acestuia sunt ilustrate în figura 5.20d.

Schema logică



Pseudocod

Început

Pentru i ← 1, 16

  Citeşte c_i

Sfârşit pentru

S ← 0 ; j ← 0

Pentru i ←1 , 16

  Dacă $c_i \geq 0$ **atunci**

    Dacă j + 1 MOD 2 = 0 **atunci** $cod_j \leftarrow c_i$; $S \leftarrow S + cod_j$; $j \leftarrow j + 1$

      altfel $cod_j \leftarrow c_i$; $S \leftarrow S + cod_j * 3$; $j \leftarrow j + 1$

    Sfârșit dacă

  Sfârșit dacă

Sfârşit pentru

Dacă S MOD 10 = 0 **atunci** control ← 0
  altfel control ← 10 - S MOD 10
Sfârşit dacă
Dacă S MOD 11 = control **atunci** **Scrie** „COD VALID"
  altfel **Scrie** „COD INVALID"
Sfârşit dacă
Sfârşit

Figura V.20c. Reprezentarea algoritmului pentru verificarea codului **ISBN-13**

Programul C	Rularea programului
(see code below)	Introduceti codul: **978-606-737-208-3**. COD VALID!!! Introduceti codul: **978-973-755-835-0** COD VALID!!!

```c
#include<stdio.h>
#include<conio.h>
int main(void)
{ int cod[13], i,j=0, S=0, control; char c[17];
  printf( "\n Introduceti codul:" );
  for( i = 0 ; i <= 16 ; i++ )
        c[i] = getche();
  for( i = 0 ; i <= 15 ; i++ )
  if( c[i] - '0' >= 0 )
     if( j%2 == 0 ) { cod[j] = c[i] - '0'; S = S + cod[j]; j++; }
     else { cod[j] = c[i] - '0';  S = S + cod[j] * 3 ; j++; }
  if( S%10 == 0 )   control = 0;
  else   control = 10 – S % 10;
  if( control == c[16] - '0' )  printf("\n COD VALID!!!");
  else  printf("\n COD INVALID!!!");
}
```

Figura 5.20d. Programul C şi rularea acestuia pentru verificarea codului **ISBN-13**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric:

1. Se citeşte un cod cu valoarea **978-606-737-208-3**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 10.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10	11	12	13
Cod ISBN-13	**9**	**7**	**8**	**6**	**0**	**6**	**7**	**3**	**7**	**2**	**0**	**8**	**3**
Ponderea	1	3	1	3	1	3	1	3	1	3	1	3	
Produs parţial	9	21	8	18	0	18	7	9	7	6	0	24	
Suma produselor parţiale S =	**127**					Restul împărţirii sumei S%10 =							**7**

În acest caz cifra de control al codului este egală cu diferenţa dintre 10 şi restul obţinut, deci codul este **valid**.

2. Se citeşte un cod cu valoarea **978-973-755-835-0**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 10.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10	11	12	13
Cod ISBN-13	**9**	**7**	**8**	**9**	**7**	**3**	**7**	**5**	**5**	**8**	**3**	**5**	**0**
Ponderea	1	3	1	3	1	3	1	3	1	3	1	3	
Produs parţial	9	21	8	27	7	9	7	15	5	24	3	15	
Suma produselor parţiale S =	**150**					Restul împărţirii sumei S%10 =							**0**

În acest caz restul obţinut este 0 iar cifra de control al codului este tot 0, deci codul este **valid**.

Capitolul 6. Operaţii cu tablouri bidimensionale - matrice

Matricea reprezintă o colecţie omogenă, bidimensională de elemente, care pot fi accesate prin intermediul a doi indici, numerotaţi începând de la **0**.

Reprezentarea matricelor se realizează printr-un tabel dreptunghiular de valori:

$$A_{mxn} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0n-1} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1n-1} \\ a_{20} & a_{21} & a_{22} & \dots & a_{2n-1} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m-10} & a_{m-11} & a_{m-12} & \dots & a_{m-1n-1} \end{pmatrix}$$

Elementul general al matricei se notează cu $\mathbf{a_{ij}}$, indicele „ **i** ” arată linia pe care se află elementul, iar al doilea indice, „ **j** ”, arată coloana pe care se află elementul.

6.1. Introducerea / afişarea elementelor unui tablou bidimensional

În acest exemplu sunt prezentate operaţiile de introducere (citire), respectiv afişare (scriere) a elementelor unui tablou bidimensional cu „ **m** x **n** ” elemente, numărul de linii şi numărul de coloane fiind introduse de la tastatură.

Parcurgerea matricelor se realizează linie cu linie, astfel că pentru parcurgerea liniilor se utilizează un ciclu cu contor, contorul fiind notat cu „ **i** ”, acesta luând valori de la **0** la **m - 1**, iar pentru parcurgerea fiecărei linii se utilizează un al doilea ciclu cu contor (în corpul primului ciclu cu contor), contorul fiind notat cu „ **j** ”, acesta luând valori de la **0** la **n - 1**.

Algoritmul de parcurgere a unei matrice este următorul:

1. se citeşte **m** - numărul de linii ale tabloului bidimensional, respectiv **n** - numărul de coloane ale tabloului bidimensional;

2. se iniţializează contorul „ **i** ” cu valoarea iniţială **0**;

3. se evaluează condiţia „ **i <= m - 1** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de linii. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ” (pasul 3.1), iar dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează după acest ciclu cu contor;

Pentru fiecare valoare a lui „ **i** ” (adică pentru fiecare linie) se realizează parcurgerea liniei astfel:

3.1. se iniţializează contorul „ **j** ” cu valoarea iniţială **0**;

3.2. se evaluează condiţia „ **j <= n - 1** ”. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ” (pasul 3.3), iar dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă rularea cu secvenţa care urmează, adică pasul **4**;

3.3. se execută corpul ciclului condus de variabila „ **j** ”, în care se realizează operaţia de citire sau scriere a elementelor matricei;

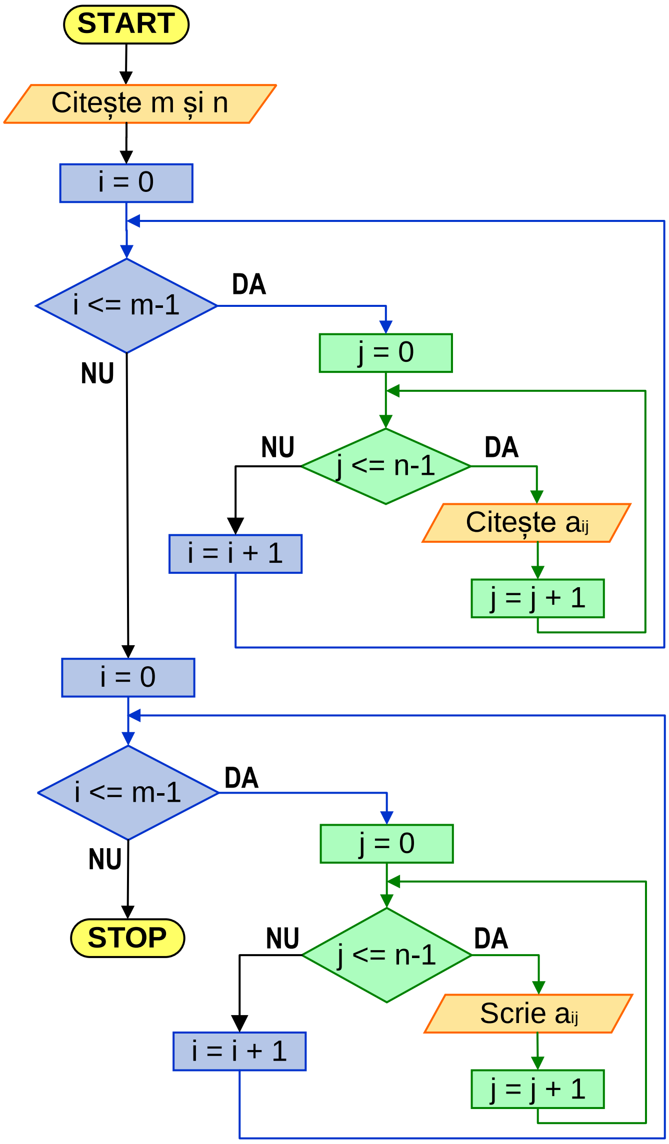
3.4. se execută instrucţiunea prin care se modifică valoarea contorului „ **i** ” cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **3**;

4. se execută instrucţiunea prin care se modifică valoarea contorului „ **i** ” cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **3**.

În exemplul următor este prezentată modalitatea de citire a elementelor unei matrice, respectiv modalitatea de afişare a elementelor unei matrice.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.1a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.1b.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte m,n
  Pentru i ← 0, m - 1
    Pentru j ← 0, n - 1
      Citeşte aij
    Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, m - 1
    Pentru j ← 0, n - 1
      Scrie aij
    Sfârşit pentru
  Sfârşit pentru
Sfârşit
```

Figura 6.1a. Reprezentarea algoritmului pentru introducerea / afişarea elementelor unui tablou bidimensional

Programul C

```
#include<stdio.h>
int main(void)
{ int i, j, m, n, a[20][20];
  printf("\n Introdu m = ");  scanf("%d",&m);
  printf("\n Introdu n = ");  scanf("%d",&n);
  for( i = 0 ; i <= m – 1 ; i++ )
      for( j = 0 ; j <= n – 1 ; j++ )
         { printf(" a[%d][%d]  = ",i,j);  scanf("%d",&a[i][j]);  }
   for( i = 0 ; i <= m – 1 ; i++ )
    { printf("\n");
       for( j = 0 ; j <= n – 1 ; j++ )   printf("%4d",a[i][j]);    }
}
```

Rularea programului

Introdu m = **2**
Introdu n = **3**
a[0][0] = **2**
a[0][1] = **4**
a[0][2] = **5**
a[1][0] = **3**
a[1][1] = **7**
a[1][2] = **1**

```
 2   4   5
 3   7   1
```

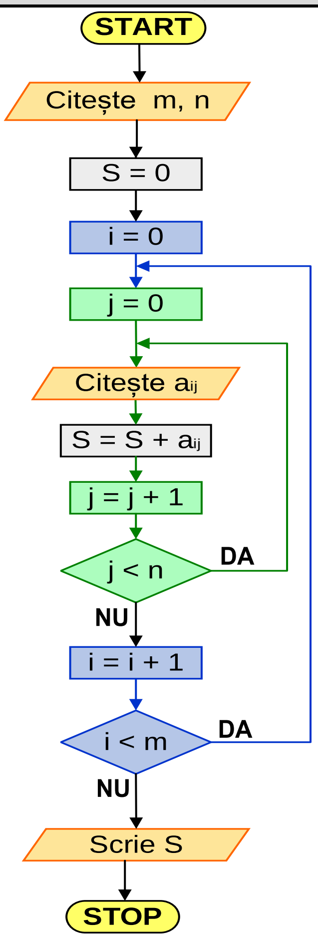
Figura 6.1b. Programul C şi rularea acestuia pentru introducerea / afişarea elementelor unui tablou bidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.2. Calculul sumei elementelor unui tablou bidimensional

Pentru calculul sumei elementelor unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se inţializează suma **S** cu valoarea zero şi se parcurge matricea, aşa cum s-a arătat în exemplul anterior. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei şi adăugarea valorii acesteia la suma **S**. Reprezentarea algoritmului prin schemă logică şi limbaj pseudocod este prezentată în figura 6.2a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.2b.

Schema logică	Limbajul pseudocod




```
Început
   Citeşte m,n
   S ← 0
   Pentru i ← 0, m-1
      Pentru j ← 0, n-1
         Citeşte aij
         S ← S + aij
      Sfârşit pentru
   Sfârşit pentru
   Scrie S
Sfârşit
```

Figura 6.2a. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou bidimensional

Programul C:

```c
#include<stdio.h>
int main(void)
{ int i, j, m, n, S, a[20][20];
  printf("\n Introdu m = "); scanf("%d",&m);
  printf("\n Introdu n = "); scanf("%d",&n);
  for(S = 0, i = 0 ; i <= m – 1 ; i++ )
     for( j = 0 ; j <= n – 1 ; j++ )
        { printf(" a[%d][%d]  = ",i,j);  scanf("%d",&a[i][j]);  S = S + a[i][j];  }
  printf("\n S = %4d",S);
}
```

Rularea programului:

```
Introdu m = 2
Introdu n = 3
 a[0][0] = 2
 a[0][1] = 4
 a[0][2] = 5
 a[1][0] = 3
 a[1][1] = 7
 a[1][2] = 1

  S = 22
```

Figura 6.2b. Programul C şi rularea acestuia pentru calculul sumei elementelor unui tablou bidimensional

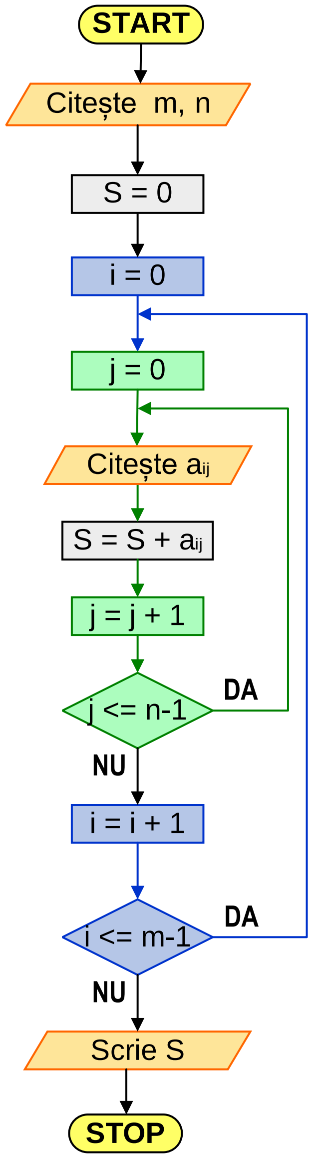
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.3. Calculul produsului elementelor strict pozitive ale unui tablou bidimensional

Pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se inţializează produsul **P** cu valoarea **1** şi se parcurge matricea. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei.

Dacă valoarea acestuia este strict pozitivă, adică condiţia **a_{ij} > 0** este adevărată, se înmulţeşte produsul **P** cu valoarea elementului curent. Dacă valoarea elementului curent nu este strict pozitivă, se continuă cu incrementarea valorii variabilei **j**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.3a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.3b.

<table>
<tr><th>Schema logică</th><th>Limbajul pseudocod</th></tr>
<tr><td>


</td><td>
Început
  Citeşte m,n
  P ← 1
  Pentru i ← 0, m - 1
    Pentru j ← 0, n - 1
    Citeşte a_{ij}
    Dacă a_{ij} > 0 atunci
      P ← P * a_{ij}
    Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Scrie P
Sfârşit
</td></tr>
</table>

Figura 6.3a. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional

Programul C	Rularea programului
#include<stdio.h>	Introdu m = **3**
int main(void)	Introdu n = **3**
{ int i, j, m, n, P, a[20][20];	a[0][0] = **2**
printf("\n Introdu m = "); scanf("%d",&m);	a[0][1] = **-4**
printf("\n Introdu n = "); scanf("%d",&n);	a[0][2] = **5**
for(P = 1, i = 0 ; i <= m – 1 ; i++)	a[1][0] = **0**
for(j = 0 ; j <= n – 1 ; j++)	a[1][1] = **3**
{ printf(" a[%d][%d] = ",i,j); scanf("%d",&a[i][j]);	a[1][2] = **1**
if(a[i][j] > 0) P = P * a[i][j];	a[2][0] = **4**
}	a[2][1] = **-2**
printf("\n P = %4d",P);	a[2][2] = **-1**
}	P = 120

Figura 6.3b. Programul C şi rularea acestuia pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.4. Calculul numărului de elemente pare ale unui tablou bidimensional

Pentru calculul numărului de elemente pare ale unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se inţializează numărul de elemente pare, notat **NEP** cu valoarea **0** şi se parcurge matricea.

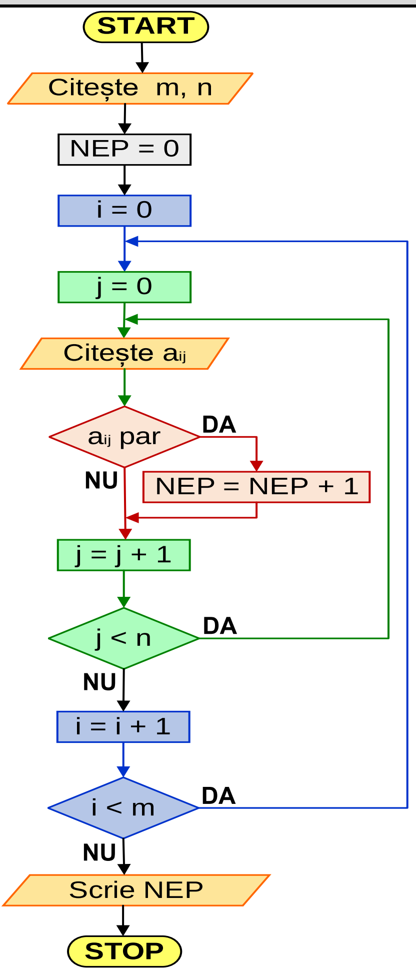
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** m, n NEP ← 0 **Pentru** i ← 0, m - 1 **Pentru** j ← 0, n - 1 **Citeşte** a_{ij} **Dacă** a_{ij} par **atunci** NEP ← NEP + 1 **Sfârşit dacă** **Sfârşit pentru** **Sfârşit pentru** **Scrie** NEP **Sfârşit**

Figura 6.4a. Reprezentarea algoritmului pentru calculul numărului de elemente pare ale unui tablou bidimensional

În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei.

Dacă valoarea acestuia este pară, adică condiţia a_{ij} **par** este adevărată, se incrementează variabila **NEP**.

Dacă valoarea elementului curent nu este pară, se continuă cu incrementarea valorii variabilei **j**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.4a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.4b.

Programul C	Rularea programului
`#include<stdio.h>` `int main(void)` `{ int i, j, m, n, NEP, a[20][20];` `  printf("\n Introdu m = ");  scanf("%d",&m);` `  printf("\n Introdu n = ");  scanf("%d",&n);` `  NEP = 0 ;` `  for( i = 0 ; i <= m – 1 ; i++ )` `      for( j = 0 ; j <= n – 1 ; j++ )` `         {  printf(" a[%d][%d]  = ",i,j);` `            scanf("%d",&a[i][j]);` `            if( a[i][j] % 2 == 0 )` `                  NEP++;` `         }` `  printf("\n NEP = %4d",NEP);` `}`	Introdu m = **3** Introdu n = **3** a[0][0] = **2** a[0][1] = **-4** a[0][2] = **5** a[1][0] = **0** a[1][1] = **3** a[1][2] = **1** a[2][0] = **4** a[2][1] = **-2** a[2][2] = **-1** NEP = 5

Figura 6.4b. Programul C şi rularea acestuia pentru calculul numărului de elemente pare ale unui tablou bidimensional

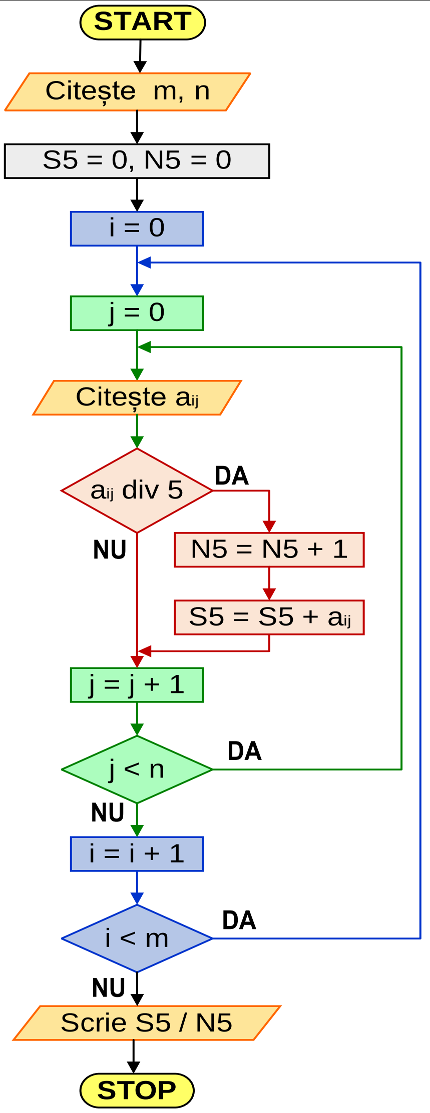
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.5. Calculul mediei aritmetice a elementelor divizibile cu 5 ale unui tablou bidimensional

Pentru calculul numărului de elemente divizibile cu **5** ale unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se inţializează numărul de elemente divizibile cu **5**, notat **N5** cu valoarea **0**, respectiv suma elementelor divizibile cu **5**, notată cu **S5** cu valoarea **0**. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei. Dacă valoarea acestuia este divizibilă cu **5**, adică condiţia a_{ij} **div 5** este adevărată, se incrementează variabila **N5**, iar la suma **S5** se adaugă valoarea a_{ij}. Dacă valoarea elementului curent nu este divizibilă cu **5**, se continuă cu incrementarea valorii variabilei **j**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.5a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.5b.

Schema logică	Limbajul pseudocod




```
Început
   Citeşte m, n
   S5 ← 0
   N5 ← 0
   Pentru i ← 0, m-1
      Pentru j ← 0, n-1
         Citeşte aij
         Dacă aij DIV 5 = 0 atunci
            N5 ← N5 + 1
            S5 ← S5 + aij
         Sfârşit dacă
      Sfârşit pentru
   Sfârşit pentru
   Scrie S5 / N5
Sfârşit
```

Figura 6.5a. Reprezentarea algoritmului pentru calculul mediei aritmetice a elementelor divizibile cu 5 pare ale unui tablou bidimensional

Programul C

```c
#include<stdio.h>
int main(void)
{ int i, j, m, n, S5, N5, a[20][20];
  printf("\n Introdu m = ");  scanf("%d",&m);
  printf("\n Introdu n = ");  scanf("%d",&n);
  S5 = 0; N5 = 0;
  for( i = 0 ; i <= m – 1 ; i++ )
      for( j = 0 ; j <= n – 1 ; j++ )
         { printf(" a[%d][%d]  = ",i,j);  scanf("%d",&a[i][j]);
            if( a[i][j] % 5 == 0 )   { N5++; S5 = S5 + a[i][j]; }
         }
  printf("\n Ma = %f",S5*1.0/N5);
}
```

Rularea programului

```
Introdu m = 3
Introdu n = 3
 a[0][0] = 15
 a[0][1] = 6
 a[0][2] = -3
 a[1][0] = 0
 a[1][1] = 5
 a[1][2] = 10
 a[2][0] = 4
 a[2][1] = 2
 a[2][2] = -5
   Ma = 5.000000
```

Figura 6.5b. Programul C şi rularea acestuia pentru calculul mediei aritmetice a elementelor divizibile cu 5 ale unui tablou bidimensional

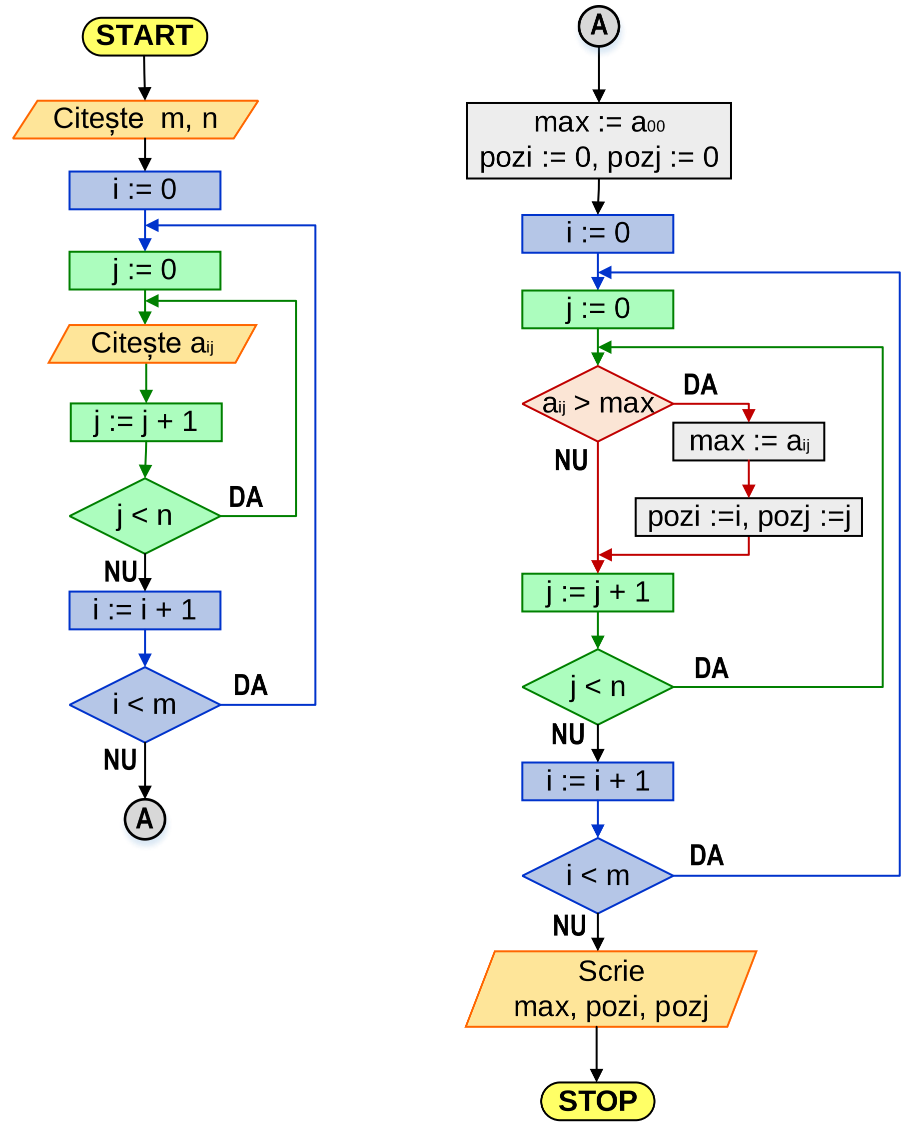
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.6. Determinarea elementului maxim şi a poziţiei acestuia dintr-un tablou bidimensional

Pentru determinarea elementului maxim dintr-un tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se inţializează variabila **max** cu valoarea elementului $\mathbf{a_{00}}$, iar variabilele **pozi**, respectiv **pozj** cu **0**. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei. Dacă valoarea acestuia este mai mare decât valoarea variabilei **max**, se atribuie lui **max** valoarea elementului curent, iar variabilelor **pozi** şi **pozj** li se atribuie valorile curente ale lui **i** şi **j**. Dacă valoarea elementului curent nu este mai mare decât **max** se continuă cu incrementarea valorii variabilei **j**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.6a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.6b.

Schema logică



Pseudocod
Început   **Citeşte** m, n   **Pentru** i ← 0, m - 1   **Pentru** j ← 0, n - 1   **Citeşte** a_{ij} **Sfârşit pentru**   **Sfârşit pentru**   max ← a_{00} pozi ← 0   pozj ← 0   **Pentru** i ← 0, m - 1   **Pentru** j ← 0, n - 1   **Dacă** a_{ij} > max **atunci**   max ← a_{ij} pozi ← i   pozj ← j   **Sfârşit dacă**   **Sfârşit pentru**   **Sfârşit pentru**   **Scrie** max, pozi, pozj **Sfârşit**

Figura 6.6a. Reprezentarea algoritmului pentru determinarea elementului maxim şi a poziţiei acestuia dintr-un tablou bidimensional

Programul C	Rularea programului
<pre>#include<stdio.h> int main(void) { int i, j, m, n, max, pozi, pozj, a[20][20]; printf("\n Introdu m = "); scanf("%d",&m); printf("\n Introdu n = "); scanf("%d",&n); for(i = 0 ; i <= m – 1 ; i++) for(j = 0 ; j <= n – 1 ; j++) { printf(" a[%d][%d] = ",i,j); scanf("%d",&a[i][j]); } max = a[0][0]; pozi = 0; pozj = 0; for(i = 0 ; i <= m – 1 ; i++) for(j = 0 ; j <= n – 1 ; j++) if(a[i][j] > max) { max = a[i][j]; pozi = i; pozj = j; } printf("\n Max = %d ",max); printf("\n pozi = %d, pozj = %d ",pozi, pozj); }</pre>	Introdu m = **3** Introdu n = **3** a[0][0] = **5** a[0][1] = **6** a[0][2] = **-3** a[1][0] = **0** a[1][1] = **5** a[1][2] = **10** a[2][0] = **4** a[2][1] = **2** a[2][2] = **-5** Max = 10 pozi = 1, pozj = 2

Figura 6.6b. Programul C şi rularea acestuia pentru determinarea elementului maxim şi a poziţiei acestuia dintr-un tablou bidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.7. Înmulţirea a două tablouri bidimensionale

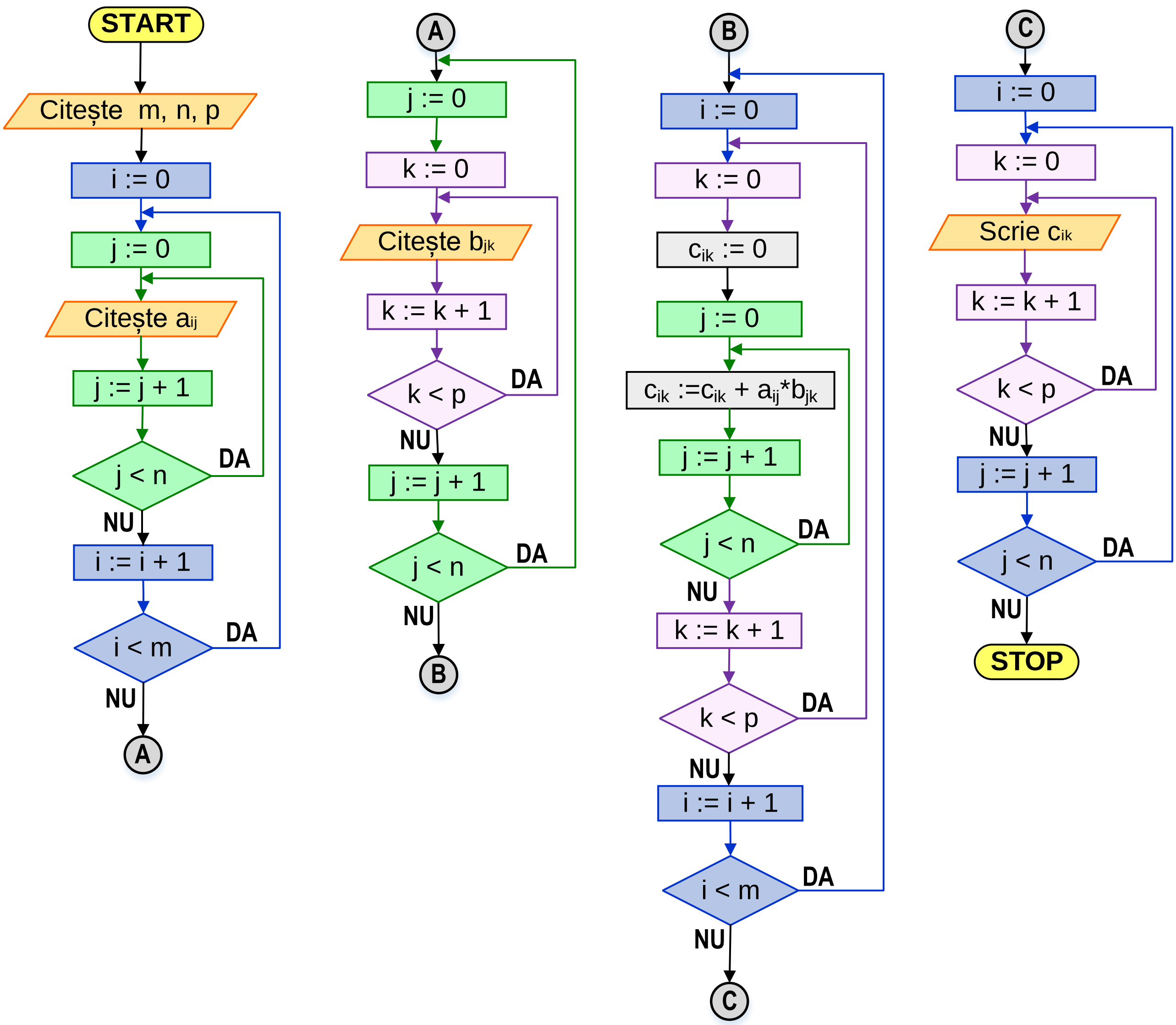
Se consideră două matrice $\mathbf{A}_{mxn}$ şi $\mathbf{B}_{nxp}$ şi se doreşte să se determine matricea $\mathbf{C}_{mxp} = \mathbf{A}_{mxn}$ **x** $\mathbf{B}_{nxp}$. Înmulţirea este posibilă doar dacă numărul de coloane de la prima matrice este egal cu numărul de linii de la a doua matrice. Relaţia de calcul a elementelor matricei **C** este:

$$c_{ik} = \sum_{j=0}^{n-1} a_{ij} \cdot b_{jk}$$

În cadrul programului se citesc cele două matrice $\mathbf{A}_{mxn}$ şi $\mathbf{B}_{nxp}$ şi se calculează elementele matricei $\mathbf{C}_{mxp} = \mathbf{A}_{mxn}$ **x** $\mathbf{B}_{nxp}$ conform relaţiei de mai sus. În final, se afişează elementele matricei **C**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.7a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.7b.

Schema logică



Pseudocod

```
Început
  Citeşte m, n, p
  Pentru i ← 0, m - 1
     Pentru j ← 0, n - 1
        Citeşte a_ij
     Sfârşit pentru
  Sfârşit pentru
  Pentru j ← 0, n - 1
     Pentru k ←0, p - 1
        Citeşte b_jk
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, m - 1
     Pentru k ← 0, p - 1
      c_ik←0
      Pentru j ← 0, n - 1
         c_ik ← c_ik + a_ij * b_jk
      Sfârşit pentru
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, m - 1
     Pentru k←0, p - 1
        Scrie c_ik
     Sfârşit pentru
  Sfârşit pentru
Sfârşit
```

Figura 6.7a. Reprezentarea algoritmului pentru înmulţirea a două tablouri bidimensionale

Programul C

```
#include<stdio.h>
int main(void)
{ int i, j, k, m, n, p, a[20][20], b[20][20], c[20][20];
  printf("\n Introdu m, n, p = ");  scanf(" %d  %d  %d ",&m, &n, &p);
  for( i = 0 ; i <= m – 1 ; i++ )
     for( j = 0 ; j <= n – 1 ; j++ )
        { printf(" a[%d][%d]  = ",i,j);   scanf("%d",&a[i][j]); }
  for( j = 0 ; j <= n – 1 ; j++ )
     for( k = 0 ; k <= p – 1 ; k++ )
        { printf(" b[%d][%d]  = ",j,k);  scanf("%d",&b[j][k]);  }
  for( i = 0 ; i <= m – 1 ; i++ )
      for( k = 0 ; k <= p – 1 ; k++ )
        { c[i][k] = 0;
           for( j = 0 ; j <= n – 1 ; j++ )   c[i][k] = c[i][k] + a[i][j] * b[j][k];  }
   printf(" \n Matricea C:");
   for( i = 0 ; i <= m – 1 ; i++ )
     { printf(”\n”);
       for( k = 0 ; k <= p – 1 ; k++ )  printf(" %4d",c[i][k]);   }
}
```

Rularea programului

```
Introdu m = 3
Introdu n = 2
Introdu p = 3
 a[0][0] = 1
 a[0][1] = 0
 a[0][2] = 2
 a[1][0] = -1
 a[1][1] = 3
 a[1][2] = 1
 b[0][0] = 3
 b[0][1] = 1
 b[1][0] = 2
 b[1][1] = 1
 b[2][0] = 1
 b[2][1] = 0
Matricea C:
    5    1
    4    2
```

Figura 6.7b. Programul C şi rularea acestuia pentru înmulţirea a două tablouri bidimensionale

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.8. Eliminarea unei linii dintr-un tablou bidimensional

Se consideră o matrice $\mathbf{A}_{mxn}$ şi se doreşte eliminarea liniei cu indicele „**k**”, cu formarea unei noi matrice $\mathbf{B}_{m-1xn}$, care conţine **m-1** linii.

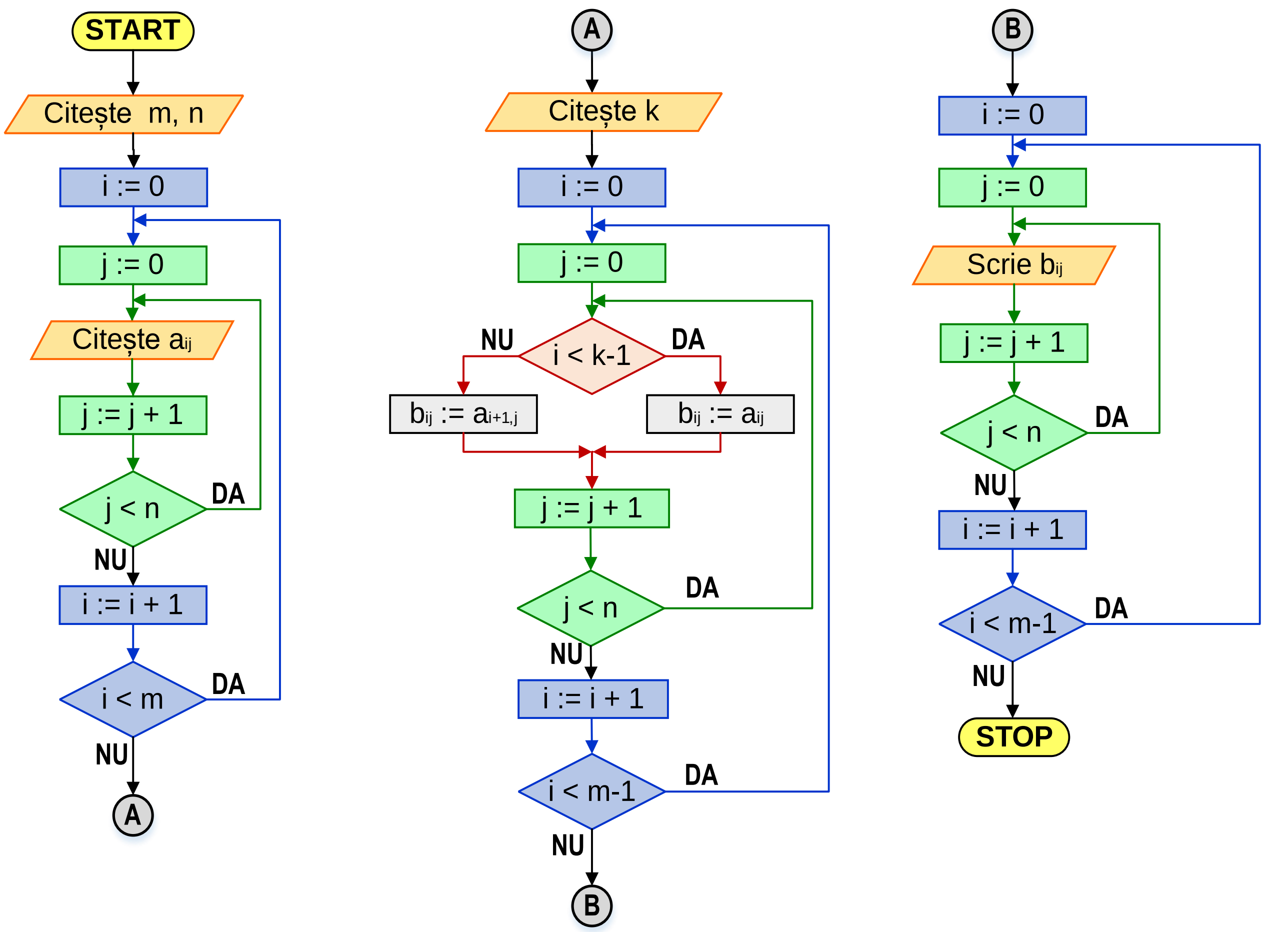
În cadrul programului se realizează citirea matricei $\mathbf{A}_{mxn}$, după care se realizează citirea valorii **k** a indicelui liniei care se elimină.

Se parcurge matricea $\mathbf{A}_{mxn}$ şi se atribuie elementelor matricei **B** valorile elementele matricei **A**, până la linia cu indicele **k**, adică atât timp cât este valabilă condiţia **i < k**.

În continuare, de la linia **k** şi până la ultima linie a matricei **B**, elementele de pe liniile **i** ale matricei **B** vor primi valorile elementelor de pe liniile **i + 1** ale matricei **A**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.8a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.8b.

Schema logică



Pseudocod

```
Început
  Citeşte m, n
  Pentru i ← 0, m - 1
     Pentru j ← 0, n - 1
       Citeşte a_ij
     Sfârşit pentru
  Sfârşit pentru
  Citeşte k
  Pentru i ← 0, m - 2
     Pentru j ← 0, n - 1
        Dacă i < k - 1 atunci
            b_ij ← a_ij
        altfel
            b_ij ← a_i+1j
        Sfârşit dacă
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, m - 2
     Pentru j ← 0, n - 1
        Scrie b_ij
     Sfârşit pentru
  Sfârşit pentru
Sfârşit
```

Figura 6.8a. Reprezentarea algoritmului pentru eliminarea liniei „k” dintr-un tablou bidimensional

Programul C

```c
#include<stdio.h>
int main(void)
{ int i, j, k, m, n, a[20][20], b[20][20];
  printf("\n Introdu m = "); scanf("%d",&m);
  printf("\n Introdu n = "); scanf("%d",&n);
  for( i = 0 ; i <= m – 1 ; i++ )
     for( j = 0 ; j <= n – 1 ; j++ )
        { printf(" a[%d][%d]  = ",i,j); scanf("%d",&a[i][j]); }
  printf("\n Introdu k = ");  scanf("%d",&k);
  for( i = 0 ; i <= m – 2 ; i++ )
     for( j = 0 ; j <= n – 1 ; j++ )
          if( i < k - 1)  b[i][j] = a[i][j];
          else b[i][j] = a[i+1][j];
 printf(" \n Matricea B:");
 for( i = 0 ; i <= m – 2 ; i++ )
     { printf(”\n”);
       for( j = 0 ; j <= n – 1 ; j++ )
          printf(" %4d",b[i][j]);
     }
}
```

Rularea programului

```
Introdu m = 4
Introdu n = 3
 a[0][0] = 1
 a[0][1] = 0
 a[0][2] = 2
 a[1][0] = -1
 a[1][1] = 3
 a[1][2] = 1
 a[2][0] = 3
 a[2][1] = 1
 a[2][2] = 2
 a[3][0] = -5
 a[3][1] = 4
 a[3][2] = -2
Introdu k = 2
Matricea B:
    1    0    2
    3    1    2
   -5    4   -2
```

Figura 6.8b. Programul C şi rularea acestuia pentru eliminarea liniei „k” dintr-un tablou bidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.9. Eliminarea unei coloane dintr-un tablou bidimensional

Se consideră o matrice $\mathbf{A}_{mxn}$ şi se doreşte eliminarea coloanei cu indicele „**k**", cu formarea unei matrice $\mathbf{B}_{mxn-1}$, care conţine **n-1** coloane.

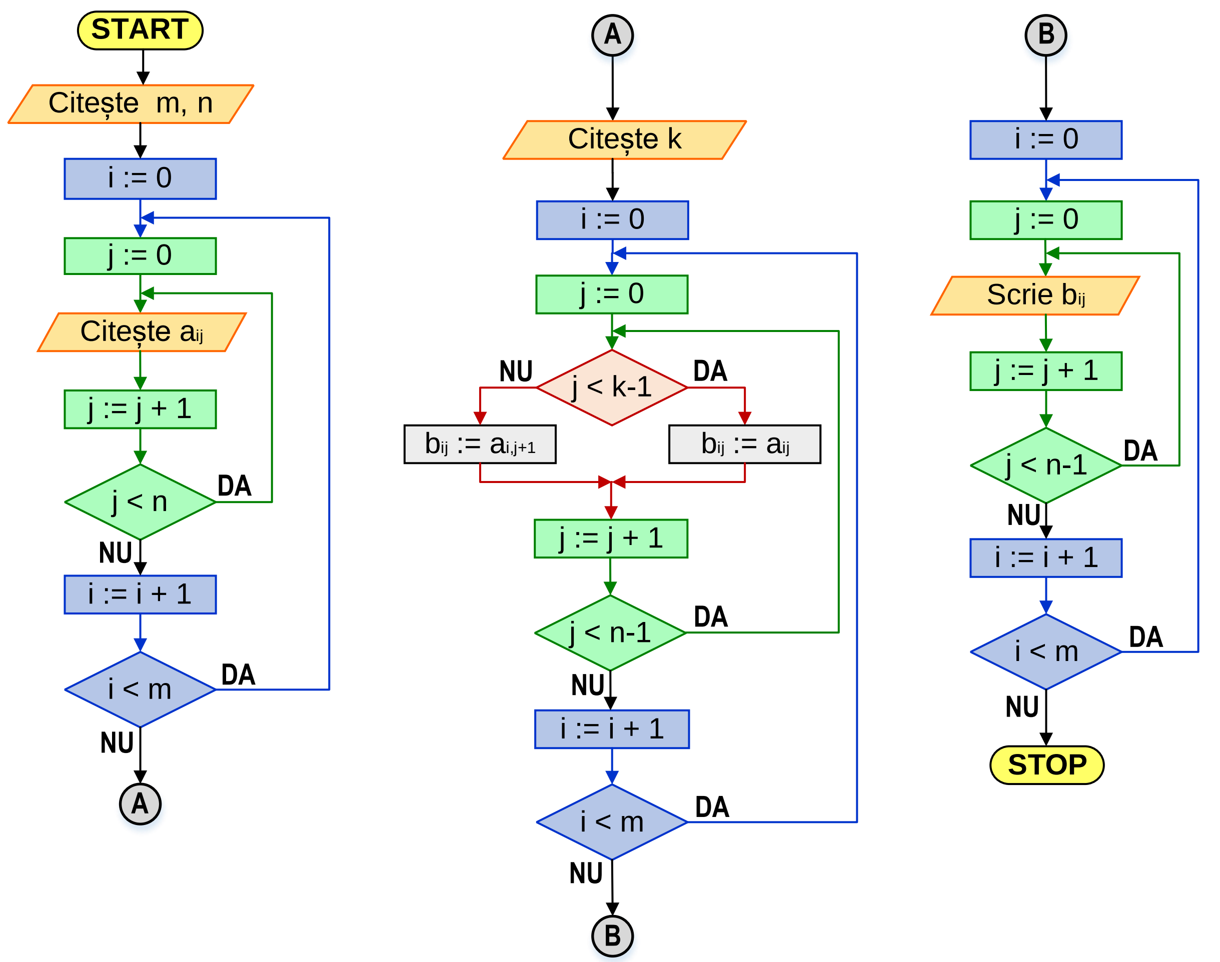
În cadrul programului se realizează citirea matricei $\mathbf{A}_{mxn}$, după care se realizează citirea valorii **k** a indicelui coloanei care se elimină.

Se parcurge matricea $\mathbf{A}_{mxn}$ şi li se atribuie elementelor matricei **B** valorile elementele matricei **A**, până la coloana cu indicele **k**, adică atât timp cât este valabilă condiţia **j < k**.

În continuare, de la coloana **k** şi până la ultima coloană a matricei **B**, elementele de pe coloanele **i** ale matricei **B** vor primi valorile elementelor de pe coloanele **j + 1** ale matricei **A**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.9a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.9b.

Schema logică



Pseudocod

```
Început
  Citeşte m,n
  Pentru i ← 0, m - 1
     Pentru j ← 0, n - 1
        Citeşte a_ij
     Sfârşit pentru
  Sfârşit pentru
  Citeşte k
  Pentru i ← 0, m - 1
     Pentru j ← 0, n - 2
        Dacă j < k - 1 atunci
           b_ij ← a_ij
        altfel
           b_ij ← a_ij+1
        Sfârşit dacă
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, m - 1
     Pentru j ← 0, n - 2
        Scrie b_ij
     Sfârşit pentru
  Sfârşit pentru
Sfârşit
```

Figura 6.9a. Reprezentarea algoritmului pentru eliminarea coloanei „k” dintr-un tablou bidimensional

Programul C

```c
#include<stdio.h>
int main(void)
{ int i, j, k, m, n, a[20][20], b[20][20];
  printf("\n Introdu m = "); scanf("%d",&m);
  printf("\n Introdu n = "); scanf("%d",&n);
  for( i = 0 ; i <= m – 1 ; i++ )
     for( j = 0 ; j <= n – 1; j++ )
        { printf(" a[%d][%d]  = ",i,j);   scanf("%d",&a[i][j]);   }
 printf("\n Introdu k = "); scanf("%d",&k);
 for( i = 0 ; i <= m – 1; i++ )
     for( j = 0 ; j < n – 1 ; j++ )
         if( j < k - 1)  b[i][j] = a[i][j];
         else b[i][j] = a[i][j+1];
 printf(" \n Matricea B:");
 for( i = 0 ; i <= m – 1 ; i++ )
    { printf(”\n”);
      for( j = 0 ; j < n – 1 ; j++ )
         printf(" %4d",b[i][j]);
    }
}
```

Rularea programului

```
Introdu m = 4
Introdu n = 3
 a[0][0] = 1
 a[0][1] = 0
 a[0][2] = 2
 a[1][0] = -1
 a[1][1] = 3
 a[1][2] = 1
 a[2][0] = 3
 a[2][1] = 1
 a[2][2] = 2
 a[3][0] = -5
 a[3][1] = 4
 a[3][2] = -2
Introdu k = 2
Matricea B:
   1    2
  -1    1
   3    2
  -5   -2
```

Figura 6.9b. Programul C şi rularea acestuia pentru eliminarea coloanei „k” dintr-un tablou bidimensional

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.10. Suma elementelor situate deasupra diagonalei principale (matrice pătratice)

Un tablou bidimensional se numeşte tablou pătratic sau matrice pătratică dacă numărul de linii este egal cu numărul de coloane. În acest caz se utilizează pentru specificarea numărului de linii şi de coloane o singură variabilă, de obicei n.

În cazul matricelor pătratice există două elemente definitorii: **diagonala principală** (figura 6.10a), respectiv **diagonala secundară** (figura 6.10b).

	0	1	2	...	n - 2	n - 1
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	...	$a_{0,n-2}$	$a_{0,n-1}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	...	$a_{1,n-2}$	$a_{1,n-1}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	...	$a_{2,n-2}$	$a_{2,n-1}$
...	...	...	...	...	...	...
n - 2	$a_{n-2,0}$	$a_{n-2,1}$	$a_{n-2,2}$	...	$a_{n-2,n-2}$	$a_{n-2,n-1}$
n - 1	$a_{n-1,0}$	$a_{n-1,1}$	$a_{n-1,3}$	...	$a_{n-1,n-2}$	$a_{n-1,n-1}$

Figura 6.10a. Diagonala principală

	0	1	2	...	n - 2	n - 1
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	...	$a_{0,n-2}$	$a_{0,n-1}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	...	$a_{1,n-2}$	$a_{1,n-1}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	...	$a_{2,n-2}$	$a_{2,n-1}$
...	...	...	...	...	...	...
n - 2	$a_{n-2,0}$	$a_{n-2,1}$	$a_{n-2,2}$	...	$a_{n-2,n-2}$	$a_{n-2,n-1}$
n - 1	$a_{n-1,0}$	$a_{n-1,1}$	$a_{n-1,3}$	...	$a_{n-1,n-2}$	$a_{n-1,n-1}$

Figura 6.10b. Diagonala secundară

Un element al matricei aparţine (sau nu) diagonalelor sau zonelor delimitate de acestea dacă respectă anumite reguli în care se utilizează indicii elementelor şi nu valoarea acestora.

Elementele situate pe diagonala principală au proprietatea că indicele liniei este egal cu indicele coloanei, adică **i = j**.

Elementele situate pe diagonala secundară au proprietatea că suma indicilor liniei şi coloanei este egală cu **n - 1**, adică **i + j = n - 1**;

Elementele situate sub diagonala principală au proprietatea că indicele liniei este strict mai mare decât indicele coloanei, adică **i > j**;

Elementele situate deasupra diagonalei principale au proprietatea că indicele liniei este strict mai mic decât indicele coloanei, adică: **i < j**;

Elementele situate sub diagonala secundară au proprietatea că suma indicilor liniei şi a coloanei este strict mai mare decât **n – 1,** adică **i + j > n - 1**;

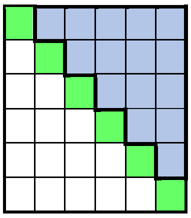
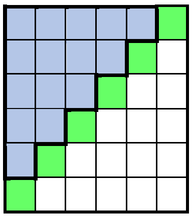
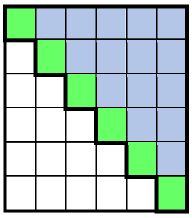
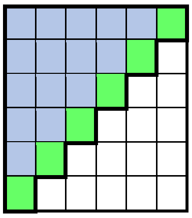
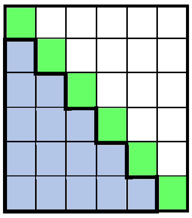
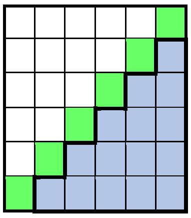
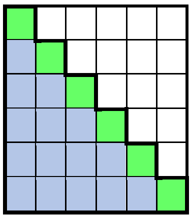
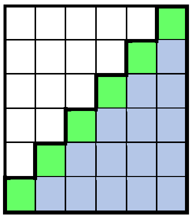
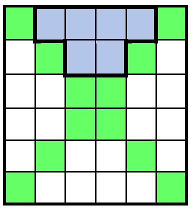
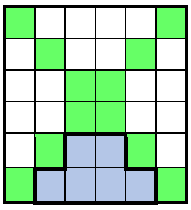
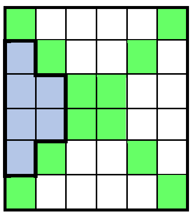
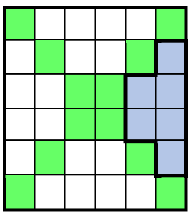
Elementele situate deasupra diagonalei secundare au proprietatea că suma indicilor liniei şi coloanei este strict mai mică decât **n - 1**, adică **i + j < n - 1**;

Relaţiile prezentate sunt valabile pentru indexarea de la **1**. În programe se utilizează indexarea de la **0** a tablourilor, relaţiile de mai sus modificându-se corespunzător doar pentru zonele care conţin diagonala secundară.

Pentru utilizarea elementelor dintr-o anumită zonă a matricei există două posibilităţi:
- se parcurge întreaga matrice şi se impun anumite condiţii indicilor elementelor matricei;
- se parcurge doar zona din matrice prin impunerea unor limite de variaţie a valorilor indicilor elementelor matricei.

În tabelul 6.10.1 sunt prezentate o serie de exemple de parcurgere a unor zone dintr-o matrice pătratică:

Tabelul 6.10.1. Modalităţi de parcurgere a unei anumite zone dintr-o matrice pătratică

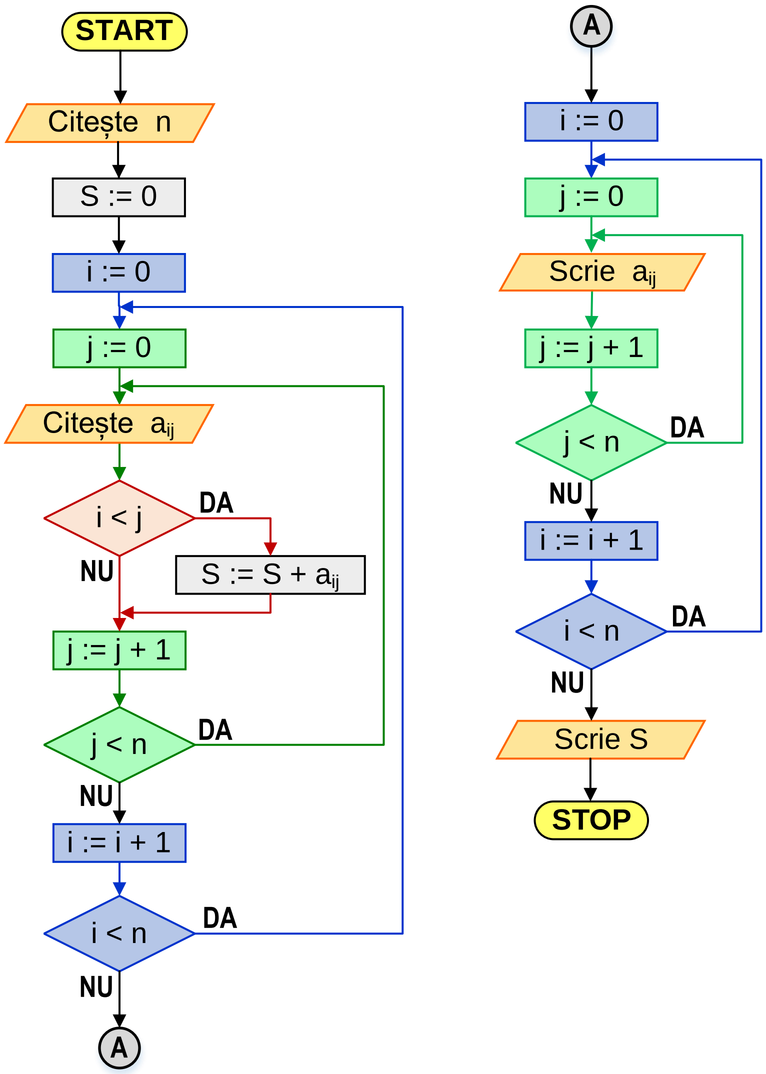
Reprezentare grafică:	Condiţii impuse indicilor	Limite impuse indicilor	Reprezentare grafică:	Condiţii impuse indicilor	Limite impuse indicilor
Deasupra diagonalei principale			**Deasupra diagonalei secundare**		
	i < j	i = 0, ..., n - 2 j = i + 1, ..., n - 1		i + j < n-1	i = 0, ..., n – 1 j = 0, ..., n – i - 2
Deasupra şi pe diagonala principală			**Deasupra şi pe diagonala secundară**		
	i <= j	i = 0, ..., n - 1 j = i, ..., n - 1		i+j <= n-1	i = 0, ..., n - 1 j = 0, ..., n – i - 1
Sub diagonala principală			**Sub diagonala secundară**		
	i > j	i = 1, ..., n - 1 j = 0, ..., i - 1		i + j > n-1	i = 1, ..., n - 1 j = n – i, ..., n - 1
Sub şi pe diagonala principală			**Sub şi pe diagonala secundară**		
	i >= j	i = 0, ..., n - 1 j = 0, ..., i		i + j > n-1	i = 0, ..., n - 1 j = n – i - 1, ..., n - 1
Deasupra diagonalei principale şi secundare			**Sub diagonala principală şi secundară**		
	i < j şi i + j < n - 1	i = 0, ..., n / 2 - 1 j = i + 1, ..., n – i - 1		i > j şi i + j > n - 1	i = n / 2 + 1, ..., n - 1 j = n - i, ..., i - 1
Sub diagonala principală şi deasupra celei secundare			**Deasupra diagonalei principale şi sub cea secundară**		
	i > j şi i + j < n - 1	j = 0, ..., n / 2 - 1 i = j + 1, ..., n – j - 1		i < j şi i + j > n-1	j = n / 2 + 1, ..,n - 1 i = n - j, ..., j – 1

Algoritmul pentru calculul sumei elementelor situate deasupra diagonalei principale poate fi conceput în două variante:

Varianta 1: Se parcurge toată matricea şi se impun condiţii indicilor:
- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila **n**;
- se iniţializează valoarea sumei cu **0**;
- utilizând un ciclu cu contor se parcurge matricea linie cu linie, atribuindu-se contorului pentru linii (variabila **i**) valori de la **0** la **n - 1**. Pentru fiecare valoare a lui **i** (adică pentru fiecare linie) se realizează parcurgerea liniei utilizând un ciclu cu contor în care variabila **j** primeşte valori de la **0** la **n - 1**. Pentru fiecare element al matricei, se realizează citirea valorii acestuia şi utilizând o instrucţiune de decizie, se verifică dacă se află situat deasupra diagonalei principale (**i < j**). Dacă condiţia este îndeplinită se adaugă valoarea acestui element la sumă;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se afişează suma calculată.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.10c, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.10d.

Schema logică



Pseudocod

Început
  Citeşte n
  S ← 0
  Pentru i ← 0, n - 1
    Pentru j ← 0, n - 1
      Citeşte a_{ij}
      Dacă i < j **atunci**
        S ← S + a_{ij}
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, n - 1
    Pentru j ← 0, n - 1
      Scrie a_{ij}
    Sfârşit pentru
  Sfârşit pentru
Scrie S
Sfârşit

Figura 6.10c. Reprezentarea algoritmului pentru calculul sumei elementelor situate deasupra diagonalei principale – v. 1

Programul C	Rularea programului

Programul C:

```
#include<stdio.h>
int main(void)
{ int n, i, j; float S, a[10][10];
 printf("\n Introduceti n, n = ");   scanf("%d",&n);
 S = 0;
 for( i = 0 ; i < n ; i++ )
   for( j = 0 ; j < n ; j++ )
     { printf(" a[%d][%d] = ",i,j);   scanf("%f",&a[i][j]);
        if( i < j )
            S = S + a[i][j];
     }
 for( i = 0 ; i < n ; i++ )
 { printf("\n");
   for( j = 0 ; j < n ; j++ )
      printf(" %6.2f ",a[i][j]);
 }
 printf("\n Suma S = %6.2f",S);
}
```

Rularea programului:

```
Introduceti n, n = 4
 a[0][0] = 1
 a[0][1] = 5
 a[0][2] = 6
 a[0][3] = 2
 a[1][0] = 3
 a[1][1] = 0
 a[1][2] = 2
 a[1][3] = 4
 a[2][0] = 8
 a[2][1] = 7
 a[2][2] = 9
 a[2][3] = 5
 a[3][0] = 2
 a[3][1] = 6
 a[3][2] = 3
 a[3][3] = 0

   1.00   5.00   6.00   2.00
   3.00   0.00   2.00   4.00
   8.00   7.00   9.00   5.00
   2.00   6.00   3.00   0.00
 Suma S = 24.00
```

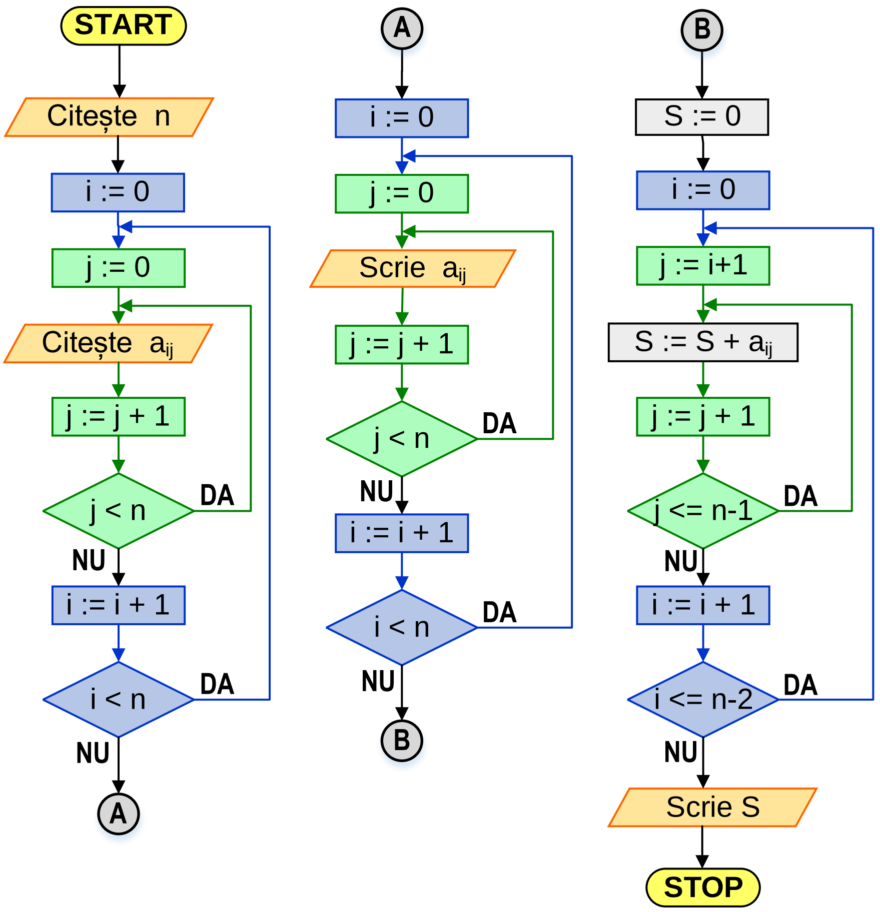
Figura 6.10d. Programul C şi rularea acestuia pentru calculul sumei elementelor situate deasupra diagonalei principale – v. 1

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Varianta 2: Se citeşte şi se afişează toată matricea, iar pentru calcularea sumei se parcurge doar zona de deasupra diagonalei principale:
- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila **n**;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează valoarea sumei cu **0**;
- utilizând două cicluri cu contor suprapuse se realizează calculul sumei impunând indicilor intervalele de variaţie;
- se afişează suma calculată.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.10e, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.10f.

Schema logică



Pseudocod

```
Început
  Citeşte n
  Pentru i ← 0, n - 1
      Pentru j ← 0, n - 1
          Citeşte a_ij
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, n - 1
      Pentru j ← 0, n - 1
          Scrie a_ij
      Sfârşit pentru
  Sfârşit pentru
  S ← 0
  Pentru i ← 0, n - 2
      Pentru j ← i + 1, n - 1
          Scrie S ← S + a_ij
      Sfârşit pentru
  Sfârşit pentru
  Scrie S
Sfârşit
```

Figura 6.10e. Reprezentarea algoritmului pentru calculul sumei elementelor situate deasupra diagonalei principale – v. 2

Programul C

```c
#include<stdio.h>
int main(void)
{ int n, i, j; float S, a[10][10];
  printf("\n Introduceti n, n = ");
  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
    for( j = 0 ; j < n ; j++ )
      { printf(" a[%d][%d] = ",i,j);
        scanf("%f",&a[i][j]);  }
  for( i = 0 ; i < n ; i++ )
  { printf("\n");
    for( j = 0 ; j < n ; j++)
       printf(" %6.2f ",a[i][j]);
  }
  S = 0;
  for( i = 0 ; i <= n – 2 ; i++ )
    for( j = i + 1 ; j <= n - 1 ; j++ )
        S = S + a[i][j];
  printf("\n Suma S = %6.3f",S);
}
```

Rularea programului

```
Introduceti n, n = 4
 a[0][0] = 1
 a[0][1] = 5
 a[0][2] = 6
 a[0][3] = 2
 a[1][0] = 3
 a[1][1] = 0
 a[1][2] = 2
 a[1][3] = 4
 a[2][0] = 8
 a[2][1] = 7
 a[2][2] = 9
 a[2][3] = 5
 a[3][0] = 2
 a[3][1] = 6
 a[3][2] = 3
 a[3][3] = 0

   1.00    5.00    6.00    2.00
   3.00    0.00    2.00    4.00
   8.00    7.00    9.00    5.00
   2.00    6.00    3.00    0.00
 Suma S = 24.00
```

Figura 6.10f. Programul C şi rularea acestuia pentru calculul sumei elementelor situate deasupra diagonalei principale – v. 2

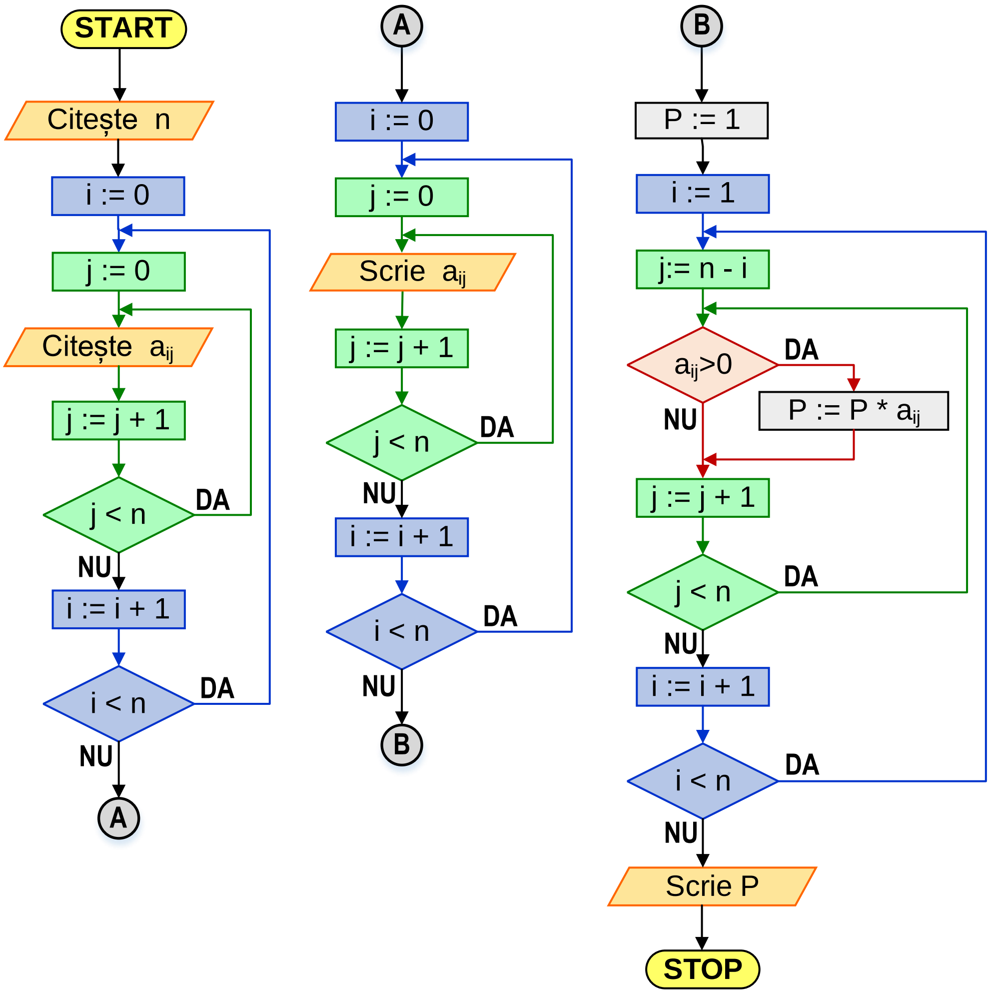
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.11. Produsul elementelor strict pozitive situate sub diagonala secundară

Algoritmul pentru calculul produsului elementelor strict pozitive situate sub diagonala secundară a unei matrice pătratice necesită parcurgerea următorilor paşi:
- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila **n**;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează valoarea produsului cu **1**;
- utilizând două cicluri cu contor suprapuse se parcurge zona din matrice de sub diagonala secundară, impunând indicilor intervalele de variaţie, adică pentru **i** de la **1** la **n - 1**, respectiv pentru **j** de la **n - i** la **n - 1**. De asemenea, pentru fiecare element al matricei care se află în această zonă se verifică, cu ajutorul unei instrucţiuni de decizie, dacă este strict pozitiv. Dacă elementul este strict pozitiv se înmulţeşte valoarea acestuia cu valoarea produsului;
- se afişează produsul calculat;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.11a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.11b.

Schema logică



Pseudocod

```
Început
  Citeşte n
  Pentru i ← 0, n - 1
      Pentru j ← 1, n - 1
          Citeşte aij
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, n - 1
      Pentru j ← 0, n - 1
          Scrie aij
      Sfârşit pentru
  Sfârşit pentru
  P ← 1
  Pentru i ← 1, n - 1
       Pentru j ← n – i , n -1
            Dacă aij > 0 atunci
                 P ← P * aij
            Sfârşit dacă
       Sfârşit pentru
  Sfârşit pentru
  Scrie P
Sfârşit
```

Figura 6.11a. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive situate sub diagonala secundară

Programul C

```c
#include<stdio.h>
int main(void)
{ int n, i, j; float P, a[10][10];
  printf("\n Introduceti n, n = ");
  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
    for( j = 0 ; j < n ; j++ )
      { printf(" a[%d][%d] = ",i,j);   scanf("%f",&a[i][j]);  }
  for( i = 0 ; i < n ; i++ )
      { printf("\n");
        for( j = 0 ; j < n ; j++ )
              printf(" %6.2f ",a[i][j]);
      }
  P = 1;
  for( i = 1 ; i <= n – 1 ; i++ )
    for( j = n – i ; j <= n – 1 ; j++ )
       if( a[i][j] > 0 )
            P = P * a[i][j];
  printf("\n Produsul P = %6.2f",P);
}
```

Rularea programului

```
Introduceti n, n = 4
 a[0][0] = 1
 a[0][1] = 2
 a[0][2] = 3
 a[0][3] = 4
 a[1][0] = -2
 a[1][1] = 5
 a[1][2] = 6
 a[1][3] = 0
 a[2][0] = -3
 a[2][1] = 2
 a[2][2] = 0
 a[2][3] = 1
 a[3][0] = -5
 a[3][1] = 2
 a[3][2] = 3
 a[3][3] = 0

  1.00    2.00    3.00    4.00
 -2.00    5.00    6.00    0.00
 -3.00    2.00    0.00    1.00
 -5.00    2.00    3.00    0.00
 Produsul P =   6.00
```

Figura 6.11b. Programul C şi rularea acestuia pentru calculul produsului elementelor strict pozitive situate sub diagonala secundară

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

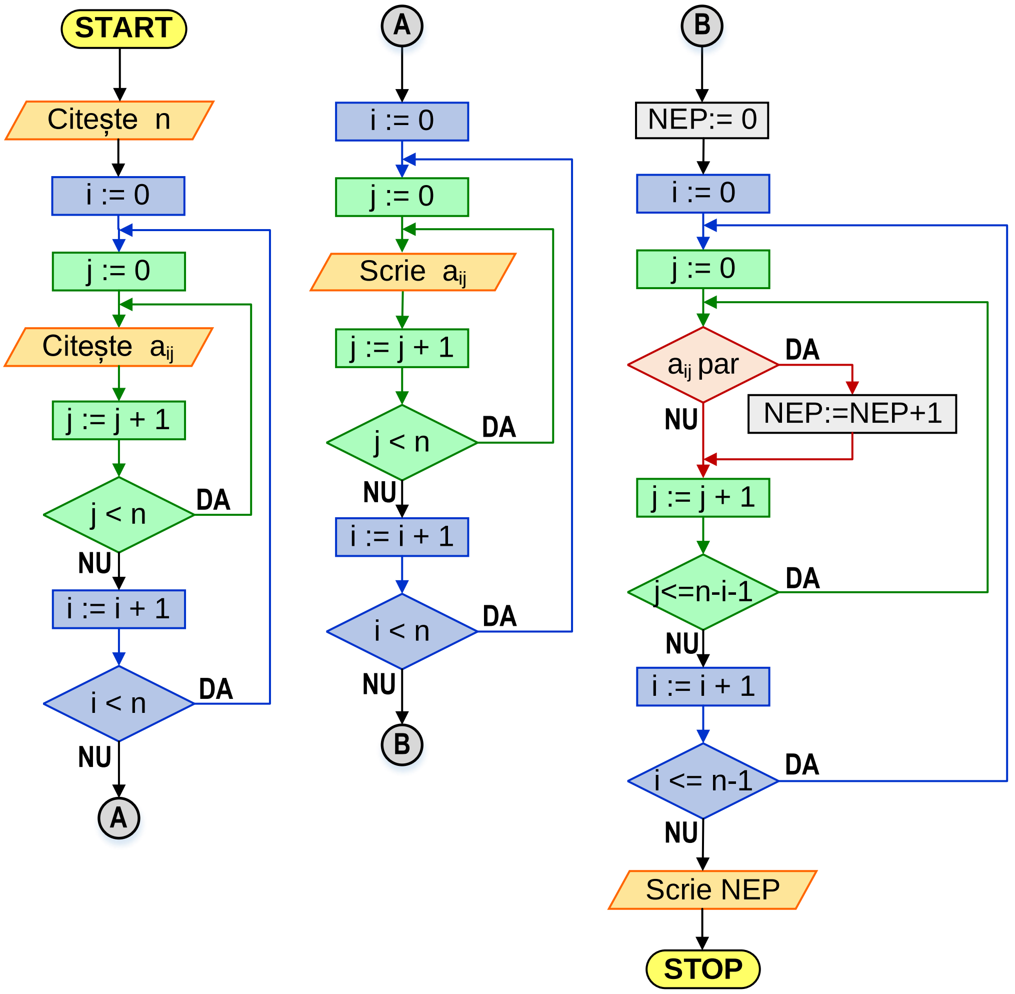
6.12. Numărul de elemente pare de pe şi deasupra diagonalei secundare (matrice pătratice)

Algoritmul pentru calculul numărului de elemente pare situate pe şi deasupra diagonalei secundare a unei matrice pătratice necesită parcurgerea următorilor paşi:

- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila **n**;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează valoarea variabilei care memorează numărul de elemente pare cu **0**, adică **NEP = 0**;
- utilizând două cicluri cu contor suprapuse se parcurge zona din matrice formată din diagonala secundară şi zona de deasupra diagonalei secundare, impunând indicilor intervalele de variaţie, adică pentru **i** de la **0** la **n - 1**, respectiv pentru **j** de la **0** la **n – i - 1**. De asemenea, pentru fiecare element al matricei care se află în această zonă se verifică, cu ajutorul unei instrucţiuni de decizie, dacă este par, adică dacă restul împărţirii acestuia la 2 este nul. Dacă elementul este par se incrementează numărul de elemente pare;
- se afişează valoarea numărului de elemente pare, adică **Scrie NEP**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.12a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.12b.

Schema logică



Pseudocod

```
Început
  Citeşte n
  Pentru i ← 0, n - 1
     Pentru j ← 0, n - 1
        Citeşte aij
     Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, n - 1
     Pentru j ← 0, n - 1
        Scrie aij
     Sfârşit pentru
  Sfârşit pentru
  NEP ← 0
  Pentru i ← 0, n - 1
     Pentru j ← 0, n - i - 1
       Dacă aij par atunci
            NEP ← NEP + 1
       Sfârşit dacă
     Sfârşit pentru
  Sfârşit pentru
  Scrie NEP
Sfârşit
```

Figura 6.12a. Reprezentarea algoritmului pentru determinarea numărului de elemente pare de pe şi deasupra diagonalei secundare

Programul C

```c
#include<stdio.h>
int main(void)
{ int n, i, j, NEP, a[10][10];
  printf("\n Introduceti n, n = ");   scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
    for( j = 0 ; j < n ; j++ )
      { printf(" a[%d][%d] = ",i,j);  scanf("%d",&a[i][j]);  }
  for( i = 0 ; i < n ; i++ )
      {  printf("\n");
         for( j = 0 ; j < n ; j++ )
              printf(" %d ",a[i][j]);
      }
  NEP = 0;
  for( i = 0 ; i <= n - 1 ; i++ )
    for( j = 0 ; j <= n – i - 1 ; j++ )
       if( a[i][j] % 2 == 0 )
              NEP++;
  printf("\n Numarul de elemente pare NEP = %d",NEP);
}
```

Rularea programului

```
Introduceti n, n = 4
 a[0][0] = 1
 a[0][1] = 2
 a[0][2] = 3
 a[0][3] = 4
 a[1][0] = 5
 a[1][1] = 6
 a[1][2] = 7
 a[1][3] = 8
 a[2][0] = 9
 a[2][1] = 0
 a[2][2] = 2
 a[2][3] = 3
 a[3][0] = 5
 a[3][1] = 6
 a[3][2] = 4
 a[3][3] = 1

 1  2  3  4
 5  6  7  8
 9  0  2  3
 5  6  4  1
 Numarul de elemente pare NEP = 4
```

Figura 6.12b. Programul C şi rularea acestuia pentru determinarea numărului de elemente pare de pe şi deasupra diagonalei secundare

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

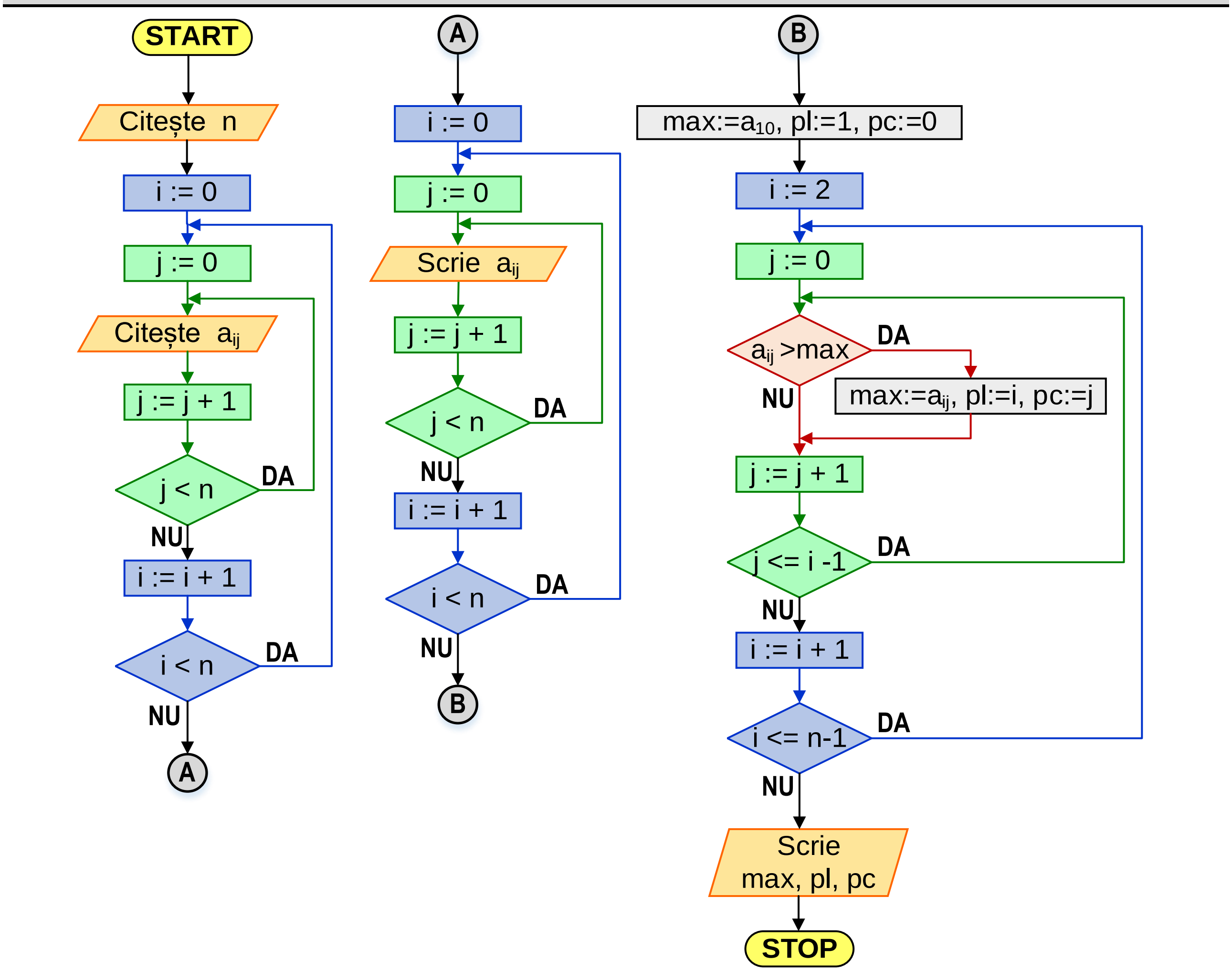
6.13. Determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală

Algoritmul pentru determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală a unei matrice pătratice presupune parcurgerea următoarelor etape:

- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila **n**;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează variabila care memorează valoarea elementului maxim cu valoarea elementului situat pe linia a doua şi pe coloana întâi, iar variabila care memorează linia pe care se află elementul maxim primeşte valoarea **1**, respectiv variabila care memorează coloana pe care se află elementul maxim primeşte valoarea **0**;
- utilizând două cicluri cu contor suprapuse se parcurge zona din matrice de sub diagonala principală, impunând limitele pentru indici, astfel indicele liniei **i** ia valori de la **2** la **n - 1**, respectiv indicele coloanei **j** ia valori de la **0** la **i - 1**. Utilizând o instrucţiune de decizie se verifică pentru fiecare element din această zonă dacă este mai mare decât maximul. Dacă elementul curent este mai mare decât maximul atunci maximul primeşte valoarea elementului curent, variabila care memorează linia pe care se află elementul maxim primeşte valoarea liniei curente, respectiv variabila care memorează coloana pe care se află elementul maxim primeşte valoarea coloanei curente;
- se afişează valoarea elementului maxim, respectiv linia şi coloana pe care se află acesta.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.13a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.13b.

Schema logică



Pseudocod

```
Început
   Citeşte n
   Pentru i ← 0, n - 1
      Pentru j ← 0, n - 1
         Citeşte a_ij
      Sfârşit pentru
   Sfârşit pentru
   Pentru i ← 0, n - 1
      Pentru j ← 0, n - 1
         Scrie a_ij
      Sfârşit pentru
   Sfârşit pentru
   max ← a_10;   pl ← 1 ;   pc ← 0
   Pentru i ← 2, n - 1
      Pentru j ← 0, i - 1
         Dacă a_ij > max atunci
            Max ← a_ij ; pl ← i ;   pc ← j
         Sfârşit dacă
      Sfârşit pentru
   Sfârşit pentru
   Scrie max, pl, pc
Sfârşit
```

Figura 6.13a. Reprezentarea algoritmului pentru determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală

Programul C

```c
#include<stdio.h>
int main(void)
{ int n, i, j, max, pl, pc, a[10][10];
  printf("\n Introduceti n, n = ");
  scanf("%d",&n);
  for( i = 0 ; i < n ; i++ )
    for( j = 0 ; j < n ; j++ )
      { printf(" a[%d][%d] = ",i,j );   scanf( "%d",&a[i][j] );  }
  for( i = 0 ; i < n ; i++ )
    { printf("\n");
      for( j = 0 ; j < n ; j++ )
         printf( " %d ",a[i][j] );
    }
  max = a[1][0]; pl = 1; pc = 0;
  for( i = 2 ; i < n ; i++ )
     for( j = 0 ; j < i - 1 ; j++ )
        if( a[i][j] > max )
           { max=a[i][j];  pl = i; pc = j; }
  printf("\n Elementul maxim este = %d",max);
  printf("\n Se afla pe linia %d si coloana %d",pl,pc);
}
```

Rularea programului

Introduceti n, n = **5**	
a[0][0] = **1**	a[2][3] = **3**
a[0][1] = **2**	a[2][4] = **7**
a[0][2] = **3**	a[3][0] = **0**
a[0][3] = **5**	a[3][1] = **6**
a[0][4] = **6**	a[3][2] = **5**
a[1][0] = **4**	a[3][3] = **4**
a[1][1] = **8**	a[3][4] = **2**
a[1][2] = **9**	a[4][0] = **3**
a[1][3] = **2**	a[4][1] = **1**
a[1][4] = **5**	a[4][2] = **8**
a[2][0] = **3**	a[4][3] = **7**
a[2][1] = **0**	a[4][4] = **0**
a[2][2] = **4**	

	1 2 3 5 6 4 8 9 2 5 3 0 4 3 7 0 6 5 4 2 3 1 8 7 0 Elementul maxim este = 8 Se afla pe linia 5 si coloana 3

Figura 6.13b. Programul C şi rularea acestuia pentru determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală

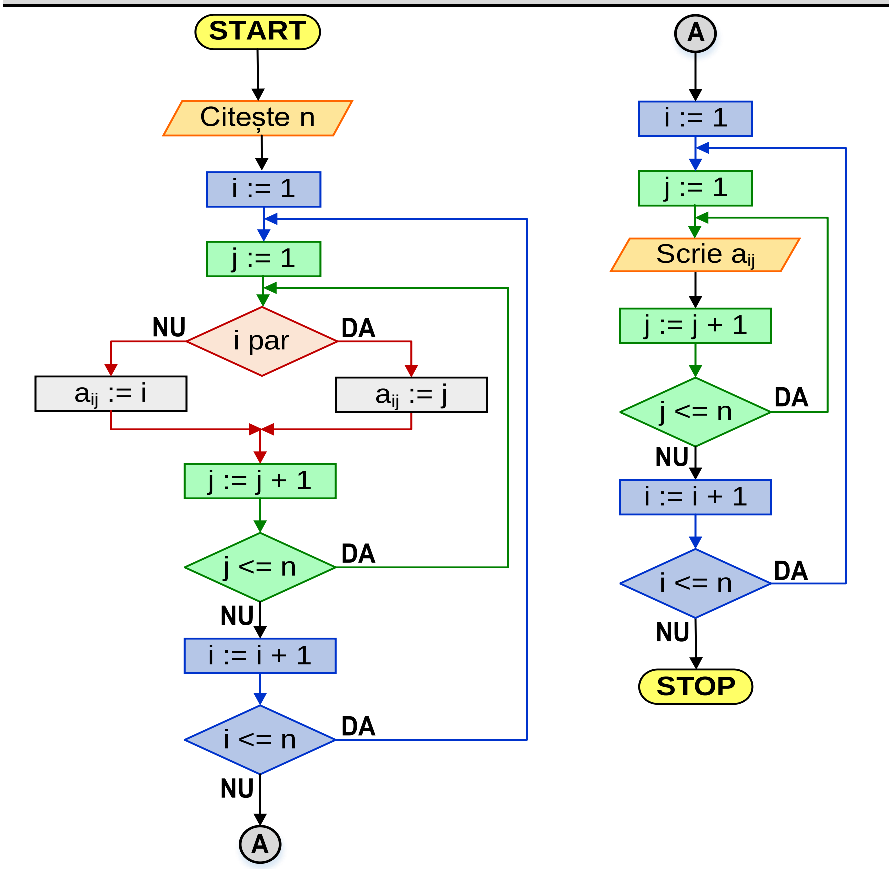
Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.14. Generarea unor matrice pătratice după o anumită cerinţă

Se doreşte generarea unor matrice pătratice după o anumită cerinţă. De exemplu, se doreşte elaborarea unui algoritm şi a unui program C pentru construirea unei o matrice pătratice cu **n** linii și coloane, numerotate de la **1** la **n**, după următoarea regulă: fiecare element din matrice aflat pe o linie impară va fi egal cu numărul liniei pe care se află iar fiecare element aflat pe o linie pară va fi egal cu numărul coloanei pe care se află.

Algoritmul pentru rezolvarea acestei cerinţe presupune parcurgerea următoarelor etape:
- se citește numărul natural **n**, care reprezintă numărul de linii și coloane ale matricei;
- utilizând două cicluri cu contor suprapuse conduse de variabilele care reprezintă indicii de linie și coloană a elementelor din matrice (**i**, respectiv **j**), se generează elementele matricei astfel: se verifică dacă indicele liniei, **i**, are o valoare pară (**i % 2 == 0**), atunci elementul matricei de pe poziția respectivă va avea valoarea egală cu numărul coloanei pe care se află, adică **a[i][j] = j**, iar dacă indicele liniei are o valoare impară, elementul matricei va avea valoarea egală cu numărul liniei pe care se află, adică **a[i][j] = i**;
- utilizând două cicluri cu contor suprapuse se afișează matricea.

Schema logică



Pseudocod

Început
  Citeşte n
  Pentru i ← 1, n
    Pentru j ← 1, n
      Dacă i par **atunci**
        a_{ij} ← j
      altfel
        a_{ij} ← i
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 1, n
    Pentru j ← 1, n
      Scrie a_{ij}
    Sfârşit pentru
  Sfârşit pentru
Sfârşit

Figura 6.14a. Reprezentarea algoritmului pentru generarea unei matrice particulare

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.14a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.14b.

Programul C	Rularea programului
see code below	see output below

```
#include<stdio.h>
int main(void)
{  int n, a[20][20], i, j;
   printf("Introduceti n, n = ");  scanf("%d",&n);
   for( i = 1 ; i <= n ; i++ )
      for( j = 1 ; j <= n ; j++ )
         {  if( i % 2 == 0 )   a[i][j] = j;
            else   a[i][j] = i;      }
   printf( "\n” );
   for( i = 1 ; i <= n ; i++ )
     {  for( j = 1 ; j <= n ; j++ )
            printf( "%5d", a[i][j] );
        printf( "\n" );    }
}
```

```
Introduceti n, n = 7

1  1  1  1  1  1  1
1  2  3  4  5  6  7
3  3  3  3  3  3  3
1  2  3  4  5  6  7
5  5  5  5  5  5  5
1  2  3  4  5  6  7
7  7  7  7  7  7  7
```

Figura 6.14b. Programul C şi rularea acestuia pentru generarea unei matrice particulare

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

6.15. Generarea unor matrice pătratice după o anumită cerinţă – triunghiul lui Pascal

Triunghiul lui Pascal reprezintă un tablou triunghiular ale cărui elemente sunt numere naturale, putând fi asociat cu o matrice pătratică având elemente în zona de pe şi de sub diagonala principală (figura 6.15a).

	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

Figura 6.15a. Triunghiul lui Pascal

Elementele de pe linia p reprezintă coeficienţii binomiali ai dezvoltării binomului lui Newton:

$$(a+b)^n = C_n^0 \cdot a^n + C_n^1 \cdot a^{n-1} \cdot b^1 + C_n^2 \cdot a^{n-2} \cdot b^2 + \cdots + C_n^k \cdot a^{n-k} \cdot b^k + \cdots + C_n^{n-1} \cdot a^1 \cdot b^{n-1} + C_n^1 b^n$$

Cunoscând că:

$$C_n^k = \begin{cases} 1, & daca\ n = k\ sau\ k = 0 \\ C_{n-1}^{k-1} + C_{n-1}^k, & altfel \end{cases}$$

se poate deduce relaţia de definiţie a valorii elementului matricei, astfel:

$$a_{ij} = \begin{cases} 1, & daca\ i = j\ sau\ j = 0 \\ a_{i-1,j-1} + a_{i-1,j}, & altfel \end{cases}$$

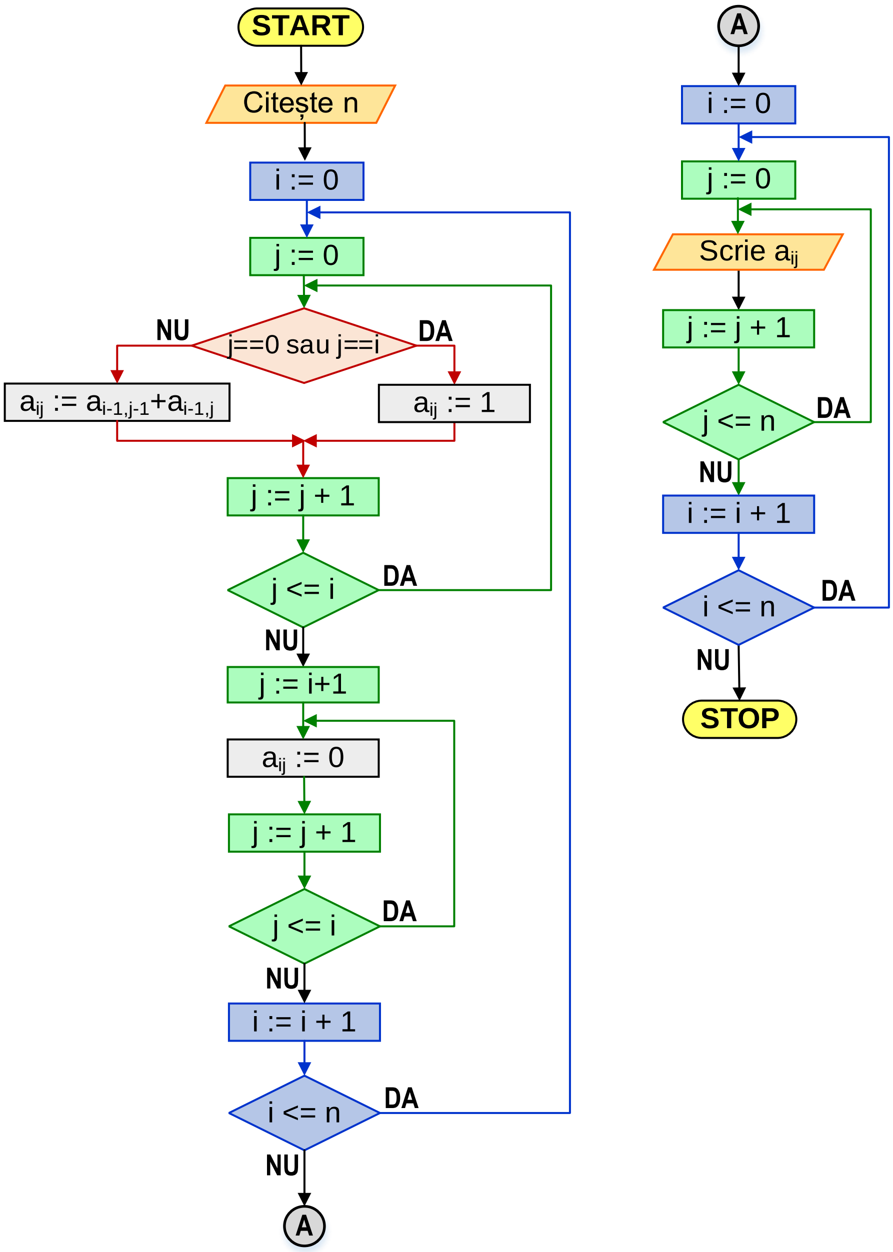
Algoritmul pentru calculul elementelor triunghiului lui Pascal necesită parcurgerea următorilor paşi:

- se citeşte valoarea variabilei **n**;
- cu ajutorul unui ciclu cu contor condus de variabila **i** se generează fiecare linie a matricei astfel:
  - cu ajutoul unui ciclu cu contor, condus de variabila **j**, se parcurge linia **i** element cu element, până când **j** este egal cu **i**. Prin intermediul unei instrucţiuni de decizie se verifică dacă **j** este egal cu **0** sau cu **i**, situaţie în care elementul matricei va fi egal cu **1**, adică **a[i][j] = 1**. În caz contrar, elementul matricei va fi calculat cu relaţia:

    a[i][j] = a[i - 1][j - 1] + a[i - 1][j]
  - cu ajutorul unui ciclu cu contor, condus tot de variabila **j**, se parcurge linia **i** de la elementul cu indicele **j + 1** până la cel cu indicele **n**, aceste elemente primind valoarea **0**;
- cu ajutorul a două cicluri cu contor se afişează matricea formată.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.15a, iar programul C aferent și rularea acestuia sunt ilustrate în figura 6.15b.

Schema logică



Pseudocod

Început
  Citeşte n
  Pentru i ← 0, n
    Pentru j ← 0, i
      Dacă j = 0 SAU j = i **atunci**
        $a_{ij} \leftarrow 1$
      altfel
        $a_{ij} \leftarrow a_{i-1j-1} + a_{i-1j}$
      Sfârşit dacă
    Sfârşit pentru
    Pentru j ← i + 1, n
      $a_{ij} \leftarrow 0$
    Sfârşit pentru
  Sfârşit pentru
  Pentru i ← 0, n
    Pentru j ← 0, n
      Scrie a_{ij}
    Sfârşit pentru
  Sfârşit pentru
Sfârşit

Figura 6.15a. Reprezentarea algoritmului pentru generarea unei matrice particulare – triunghiul lui Pascal

Programul C

```
#include<stdio.h>
int main(void)
{ int n, i, j, a[10][10];
  printf( "\n Introduceti n, n = " );
  scanf( "%d",&n );
  for( i = 0 ; i <= n ; i++ )
    {  for( j = 0 ; j <= i ; j++ )
          if( j == 0 || j == i )  a[i][j] = 1;
          else   a[i][j] = a[i - 1][j - 1] + a[i - 1][j];
       for( j = i+1 ; j<=n ; j++ )
             a[i][j] = 0;
    }
  for( i = 0 ; i <= n ; i++ )
  { printf( "\n" );
    for( j = 0 ; j <= n ; j++ )
     printf( " %3d ",a[i][j] );
  }
}
```

Rularea programului

Introduceti n, n = **7**

```
 1   0   0   0   0   0   0   0
 1   1   0   0   0   0   0   0
 1   2   1   0   0   0   0   0
 1   3   3   1   0   0   0   0
 1   4   6   4   1   0   0   0
 1   5  10  10   5   1   0   0
 1   6  15  20  15   6   1   0
 1   7  21  35  35  21   7   1
```

Figura 6.15b. Programul C şi rularea acestuia pentru generarea unei matrice particulare – triunghiul lui Pascal

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Capitolul 7. Şiruri şi serii de numere reale. Dezvoltări în serie de puteri

7.1. Şiruri de numere reale

7.1.1. Noţiuni teoretice

Prin şir de numere reale (denumit simplu şir) se înţelege o funcţie $f: \mathbb{N} \to \mathbb{R}$, care asociază oricărui număr natural ***n***, *n* ≥ *1*, numărul real notat ***a****n*, astfel că: *f(n)* = *a**n*. Uzual, ***a****n* se numeşte termenul general al şirului, iar ***n*** se numeşte rangul termenului respectiv.

Un şir poate fi dat fie precizându-se formula termenului general, fie printr-o relaţie de recurenţă.

Un şir este mărginit dacă *Im(f)* este o mulţime mărginită din $\mathbb{R}$, adică dacă şi numai dacă, există $a, b \in \mathbb{R}$ astfel încât: $a \leq a_n \leq b$ pentru orice $n \in \mathbb{N}, n \geq 1$. Dacă $a \leq a_n$, pentru orice $n \in \mathbb{N}, n \geq 1$, se spune că şirul este mărginit inferior, iar dacă $a_n \leq b$, pentru orice $n \in \mathbb{N}, n \geq 1$, se spune că şirul este mărginit superior.

Şirul $a_n, n \geq 1$ este un şir monoton dacă $f(n) = a_n$ este o funcţie monotonă, astfel că şirul este monoton crescător dacă $a_n \leq a_{n+1}$, respectiv şirul este monoton descrescător dacă $a_n \geq a_{n+1}$, oricare ar fi $n \in \mathbb{N}, n \geq 1$.

Numărul real a se numeşte limita şirului a_n, dacă $a_n \to a, n \to \infty$ şi se scrie: $\lim_{n\to\infty} a_n = a$, dacă orice vecinătate a lui a conţine toţi termenii şirului cu excepţia unui număr finit de termeni. Un şir se numeşte convergent dacă are limita un număr real. Un şir care nu este convergent se numeşte divergent.

7.1.2. Determinarea numărului de termeni necesari pentru calculul limitei unui şir, cu o precizie impusă

În acest exemplu se doreşte determinarea numărului de termeni ai şirului: $a_n = \left(1 + \frac{1}{n}\right)^n$, necesari obţinerii valorii limitei acestui şir, cu o precizie de $\varepsilon = 0.000001$. Limita acestui şir este e, baza logaritmului natural, a cărui valoare aproximativă, cu 20 de zecimale este: $e \approx 2{,}71828\ 18284\ 59045\ 23536$.

În acest caz, se calculează elementele şirului până când diferenţa dintre valoarea constantei e şi elementul curent al şirului este mai mică decât precizia impusă.

Algoritmul de calcul este următorul:

- se iniţializează variabila **n**, cu valoarea **1**;
- se iniţializează constanta **e**, cu valoarea **2.718281**;
- se iniţializează variabila **eps**, cu valoarea **0.000001**;
- se repetă următoarele operaţii cât timp diferenţa dintre valoarea precizată a constantei **e** şi cea a variabilei **a** (termenul curent) este mai mare decât precizia impusă (valoarea variabilei **eps**):
    - atribuirea valorii termenului general variabilei **a**;
    - incrementarea valorii variabilei **n**;
    - afişarea valorii termenului curent (valoarea variabilei **a**);
- se afişează valoarea ultimului termen calculat;
- se afişează valoarea variabilei **n**, adică numărul de termeni după care se obţine valoarea constantei **e**, cu precizia impusă.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.1a., iar programul C şi rularea acestuia sunt prezentate în figura 7.1b.

Se observă că se obţine valoarea constantei ***e*** cu precizia impusă de **0,000001** după **743199** termeni, valoarea termenului fiind de **2.7182800000700**.

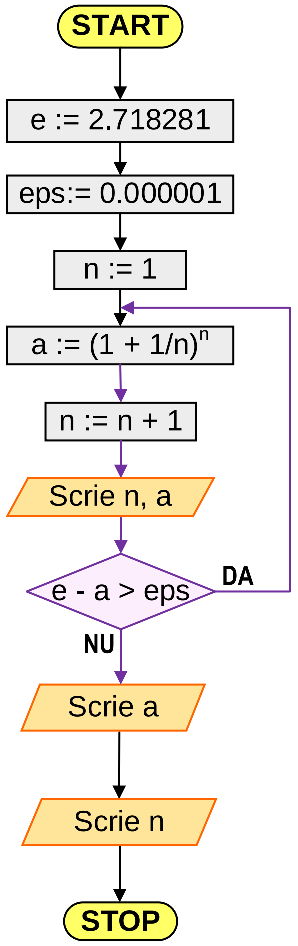
Schema logică	Limbajul pseudocod
	**Început** e ← 2.718281, eps ← 0.000001, n ← 1 **Execută** a ← $(1 + 1/n)^n$ n ← n + 1 **Scrie** n, a **Cât timp** (e - a > eps) **Scrie** a **Scrie** n **Sfârşit**

Figura 7.1a. Reprezentarea algoritmului pentru calculul termenilor şirului $a_n = \left(1 + \frac{1}{n}\right)^n$

Programul C

```
#include<stdio.h>
#include<math.h>
int main(void)
{  int n; double a, e = 2.718281, eps = 0.000001;
   n = 1;
   do { a = pow(1+1./n,n);
        n++;
        printf( "\n n = %6d \t a = %15.13lf",n,a );
      } while( e - a > eps );
   printf( "\n Termenul sirului este: a = %15.13lf",a );
   printf( "\n Nr. de termeni: n = %d",n );
}
```

Rularea programului

```
...
n = 743197      a = 2.7182799998676
n = 743198      a = 2.7182799997394
n = 743199      a = 2.7182800000700
Termenul sirului este: a = 2.7182800000700
Nr. de termeni: n = 743199
```

Figura 7.1b. Programul C şi rularea acestuia pentru calculul termenilor şirului $a_n = \left(1 + \frac{1}{n}\right)^n$

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

7.1.3. Calculul unor sume (produse) de termeni

Se doreşte calculul sumei **S = 1 + 11 + 111 + 1111 + 11111 + ...** utilizând **n** termeni.

Pentru calculul acestor sume (produse) se utilizează o relaţie de forma: **S =** $T_1 + T_2 + T_3 + T_4 + T_5 + ...$, utilizând **n** termeni. Aşa cum s-a arătat anterior, termenul general poate fi precizat fie printr-o formulă, fie printr-o relaţie de recurenţă. În continuare, sunt prezentate ambele variante care pot fi utilizate pentru calculul sumei.

Varianta I: Termenul general este definit printr-o formulă care conţine ndicele „ **i** ”:

În acest caz, termenul general este definit printr-o relaţie de calcul, de exemplu: $\boldsymbol{T_i = \sum_{j=0}^{i-1} 10^j}$.

Se iniţializează suma **S** cu **0** şi se utilizează o relaţie de forma: **S := S +** T_i, pentru toate elementele de la T_1 şi până la T_n, adică: $\boldsymbol{i = \overline{1, n}}$. Calculul sumei **S** se realizează utilizând o instrucţiune repetitivă cu contor, contorul fiind **i**, acesta primind valori de la **1** la **n**, cu pasul **1**. Se observă că fiecare termen este la rândul său o sumă, deci pentru calcul fiecărui termen T_i se va realiza iniţializarea acestuia cu **0** şi se va utiliza o instrucţiune repetitivă cu contor, contorul fiind **j**, acesta primind valori de la **0** la **i - 1**, conform relaţiei de calcul de mai sus.

Algoritmul de calcul constă în parcurgerea următorilor paşi:

1. Se citeşte numărul de termeni **n**;
2. Se iniţializează suma **S** cu **0** (zero – element neutru pentru operaţia de adunare);
3. Se iniţializează contorul **i** cu **1**, pentru calculul primului termen al sumei;
4. Se verifică condiţia **i ≤ n**, prin care se verifică dacă valoarea contorului este mai mică sau egală cu numărul de termeni, adică dacă nu am depăşit numărul de termeni specificat anterior.

Dacă condiţia este **adevărată**, se parcurg în continuare următorii paşi:

4.1. Se iniţializează termenul curent T_i cu **0**, calculul acestuia realizându-se ca o sumă de puteri a lui **10**;
4.2. Se iniţializează contorul **j**, cu valoarea **0**;
4.3. Se verifică condiţia **j <= i - 1**. Dacă condiţia este **adevărată** se modifică valoarea termenului general, astfel: $T_i := T_i + 10^j$. Dacă condiţia este falsă se trece la pasul **5**.
4.4. Se modifică valoarea contorului **j**, cu pasul **1**, astfel: **j := j + 1** după care se revine la pasul 4.3;

Dacă condiţia este **falsă**, înseamnă că s-au calculat şi adunat la sumă toţi termenii sumei, astfel că se părăseşte instrucţiunea repetitivă utilizată pentru calculul sumei, algoritmul continuând cu pasul **7**;

5. În acest pas, termenul general al sumei de la pasul **i**, adică T_i, este calculat astfel că se adaugă valoarea acestuia la suma calculată anterior, adică se utilizează relaţia: **S := S +** T_i;
6. Se modifică valoarea contorului **i**, cu pasul **1**, astfel: **i := i + 1** după care se revine la pasul **4**;
7. Se afişează valoarea calculată a sumei **S**.

În tabelul următor (tabelul 7.1) este prezentat algoritmul de calcul al sumei pentru **n = 3**.

Tabelul 7.1. Algoritmul de calcul al sumei S = 1 + 11 + 111 – varianta I

Etapa:	n	i	i <= n	j	j <= i - 1	T_i	S
0	3						0
1.1		1	DA			$T_1 = 0$	
1.2				0	DA	$T_1 := 0 + 10^0 = 0 + 1 = 1$	
1.3				1	NU		$S := 0 + T_1 = 0 + 1 = 1$
2.1		2	DA			$T_2 := 0$	
2.2				0	DA	$T_2 := 0 + 10^0 = 0 + 1 = 1$	
2.3				1	DA	$T_2 := 1 + 10^1 = 1 + 10 = 11$	
2.4				2	NU		$S := 1 + T_2 = 1 + 11 = 12$
3.1		3	DA			$T_3 := 0$	
3.2				0	DA	$T_3 := 0 + 10^0 = 0 + 1 = 1$	
3.3				1	DA	$T_3 := 1 + 10^1 = 1 + 10 = 11$	
3.4				2	DA	$T_3 := 11 + 10^2 = 11 + 100 = 111$	
3.5				3	NU		$S := 1 + 11 + T_3 = 1 + 11 + 111 = 123$
4.1		4	**NU**				**S := 1 + 11 + 111 = 123**

Etapa **0**: Se citeşte numărul de termeni, **n**, în acest caz **n = 3** şi se iniţializează suma **S** cu **0**, adică **S := 0**;
Etapa **1.1**: Contorul **i** primeşte valoarea **1**, adică **i := 1**. Se verifică dacă se îndeplineşte condiţia **i <= n**, deoarece **1 < 3**, condiţia este **adevărată**, se continuă pe ramura **DA**. Se iniţializează termenul T_i cu zero, adică $T_1 := 0$ (în acest caz **i = 1**);
Etapa **1.2**: Se iniţializează contorul **j** cu valoarea **0**, adică **j := 0** şi se verifică condiţia **j <= i - 1**. Deoarece **0 = 0**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_1 cu relaţia $T_i := T_i + 10^j$, adică: $T_1 := 0 + 10^0 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 0 + 1 = 1**;
Etapa **1.3**: Cu valoarea nouă a contorului **j** se verifică condiţia **j <= i - 1**. Deoarece **1 > 0**, condiţia este **falsă** şi se continuă pe ramura **NU**, modificându-se valoarea sumei **S** după relaţia $S := S + T_i$, adică $S := 0 + T_1 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **i**, după relaţia **i := i + 1**, deci **i := 1 + 1 = 2**;
Etapa **2.1**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **2 < 3**, condiţia este **adevărată** şi se continuă pe ramura **DA**. Se iniţializează termenul T_i cu zero, adică $T_2 := 0$ (în acest caz **i = 2**);
Etapa **2.2**: Se iniţializează din nou contorul **j** cu valoarea **0**, adică **j := 0** şi se verifică condiţia **j <= i - 1**. Deoarece **0 < 1**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_2 cu relaţia $T_i := T_i + 10^j$, adică: $T_2 := 0 + 10^0 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 0 + 1 = 1**;
Etapa **2.3**: Cu valoarea nouă a contorului **j**, adică **j := 1** şi se verifică condiţia **j <= i-1**. Deoarece **1 = 1**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_2 cu relaţia $T_i := T_i + 10^j$, adică: $T_2 := 1 + 10^1 = 1 + 10 = 11$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 1 + 1 = 2**;
Etapa **2.4**: Cu valoarea nouă a contorului **j** se verifică condiţia **j <= i-1**. Deoarece **2 > 1**, condiţia este **falsă** şi se continuă pe ramura **NU**, modificându-se valoarea sumei **S** după relaţia $S := S + T_i$, adică $S := 1 + T_2 = 1 + 11 = 12$. În continuare, se modifică valoarea contorului **i**, după relaţia **i := i + 1**, deci **i := 2 + 1 = 3**;
Etapa **3.1**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **3 = 3**, condiţia este **adevărată** şi se continuă pe ramura **DA**. Se iniţializează termenul T_i cu zero, adică $T_3 := 0$ (în acest caz **i = 3**);
Etapa **3.2**: Se iniţializează din nou contorul **j** cu valoarea **0**, adică **j := 0** şi se verifică condiţia **j <= i - 1**. Deoarece **0 < 2**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_3 cu relaţia $T_i := T_i + 10^j$, adică: $T_3 := 0 + 10^0 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 0 + 1 = 1**;
Etapa **3.3**: Cu valoarea nouă a contorului **j**, adică **j := 1** şi se verifică condiţia **j <= i - 1**. Deoarece **1 < 2**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_3 cu relaţia $T_i := T_i + 10^j$, adică: $T_3 := 1 + 10^1 = 1 + 10 = 11$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 1 + 1 = 2**;
Etapa **3.4**: Cu valoarea nouă a contorului **j**, adică **j := 2** şi se verifică condiţia **j <= i - 1**. Deoarece **2 = 2**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_3 cu relaţia $T_i := T_i + 10^j$, adică: $T_3 := 11 + 10^2 = 11 + 100 = 111$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 2 + 1 = 3**;
Etapa **3.5**: Cu valoarea nouă a contorului **j** se verifică condiţia **j <= i-1**. Deoarece **3 > 2**, condiţia este **falsă** şi se continuă pe ramura **NU**, modificându-se valoarea sumei **S** după relaţia $S := S + T_i$, adică $S := 12 + T_3 = 12 + 111 = 123$. În continuare, se modifică valoarea contorului **i**, după relaţia **i := i + 1**, deci **i := 3 + 1 = 4**;
Etapa **4.1**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **4 > 3**, condiţia este **falsă** şi se continuă pe ramura **NU**, ceea ce înseamnă că se continuă cu afişarea valorii variabilei **S**, după care se încheie algoritmul.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.1c, iar programul C şi rularea acestuia sunt prezentate în figura 7.1d.

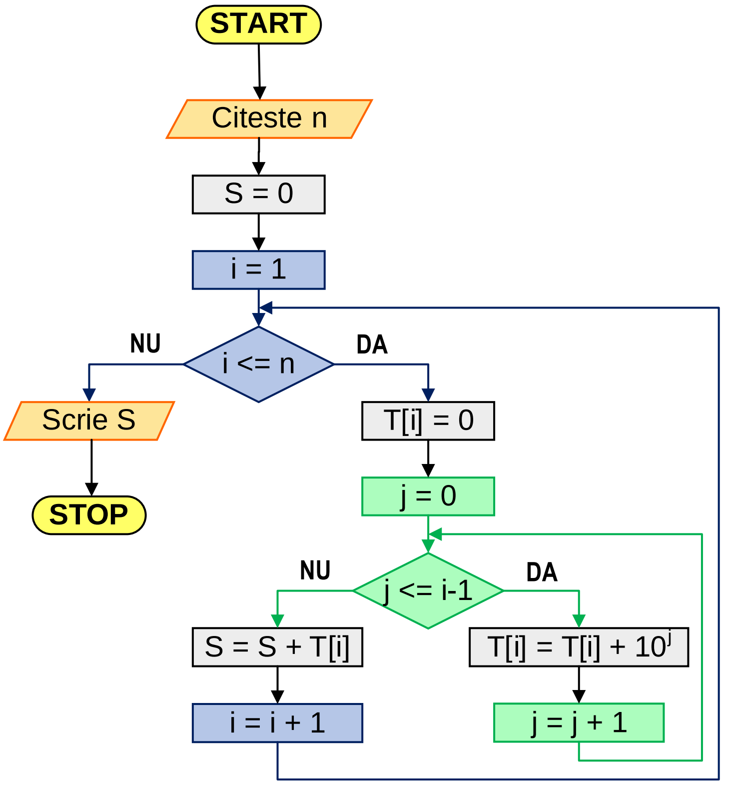
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n S ← 0 **Pentru i** = 1, n T[i] ← 0 **Pentru** j = 0, i - 1 T[i] ← T[i] + 10^j **Sfârşit pentru** S ← S + T[i] **Sfârşit pentru** **Scrie** S **Sfârşit**

Figura 7.1c. Reprezentarea algoritmului pentru calculul sumei **S = 1 + 11 + 111 + 1111 + ...**

Programul C

```
#include<stdio.h>
#include<math.h>
int main(void)
{ int i, j, n, S, T[30];
  printf( "\n Introduceti n = " );
  scanf( "%d",&n );
  S = 0;
  for( i = 1 ; i <= n ; i++ )
    { T[i] = 0;
      for( j = 0 ; j <= i - 1 ; j++ )
         T[i] = T[i] + pow(10,j);
      S = S + T[i];
    }
  printf( "\n Suma este S = %d",S );
}
```

Rularea programului

Cazul 1:

Introduceti n = **3**

Suma este S = 123

Cazul 2:

Introduceti n = **5**

Suma este S = 12345

Figura 7.1d. Programul C şi rularea acestuia pentru calculul sumei **S = 1 + 11 + 111 + 1111 + ...**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Varianta II: Termenul general este definit printr-o relaţie de recurenţă:

În acest caz, termenul general **T_i** este definit printr-o relaţie de recurenţă în care apare termenul anterior **T_{i-1}**, de exemplu: $\boldsymbol{T_i = T_{i-1} \cdot 10 + 1}$.

În această variantă, se iniţializează primul termen cu valoarea sa, adică **T_1 = 1** şi suma **S** cu valoarea primului termen, adică **S = T_1**. Pentru calculul sumei **S** se utilizează o instrucţiune repetitivă cu contor, contorul fiind **i**, acesta primind valori de la **2** la **n**, cu pasul **1**. În cadrul acestei instrucţiuni repetitive se realizează calculul valorii termenului curent **T_i** cu relaţia **T_i := T_{i-1} * 10 + 1** (relaţia de recurenţă), precum şi calculul sumei **S** cu relaţia **S := S + T_i**.

Algoritmul de calcul constă în parcurgerea următorilor paşi:

1. Se citeşte numărul de termeni **n**;
2. Se iniţializează primul termen cu valoarea sa, adică **T_1 = 1** şi suma **S** cu valoarea primului termen, adică **S = T_1**;
3. Se iniţializează contorul **i** cu **2**;
4. Se verifică condiţia **i ≤ n**, prin care se verifică dacă valoarea contorului este mai mică sau egală cu numărul de termeni, adică dacă nu am depăşit numărul de termeni specificat anterior.
Dacă condiţia este **adevărată**, se calculează valoarea termenului curent **T_i** cu relaţia **T_i := T_{i-1} * 10 + 1** şi se modifică valoarea sumei, conform relaţiei **S := S + T_i**;
Dacă condiţia este **falsă**, înseamnă că s-au calculat şi adunat la sumă toţi termenii acesteia, astfel că se părăseşte instrucţiunea repetitivă utilizată pentru calculul sumei, algorimul continuând cu pasul **6**;
5. Se modifică valoarea contorului **i**, cu pasul **1**, astfel: **i := i + 1** după care se revine la pasul **4**;
6. Se afişează valoarea calculată a sumei **S**.

În tabelul următor (tabelul 7.2) este prezentat algoritmul de calcul al sumei pentru **n = 3**.

Tabelul 7.2. Algoritmul de calcul al sumei S = 1 + 11 + 111 – varianta II

Etapa:	n	i	i <= n	T_i	S
0	3	1		T_1 := 1	S := T_1 = 1
1		2	DA	T_2 := T_1 * 10 + 1 = 1 * 10 + 1 = 11	S := 1 + T_2 = 1 + 11 = 12
2		3	DA	T_3 := T_2 * 10 + 1 = 11 * 10 + 1 = 111	S := 12 + T_3 = 12 + 111 = 123
3		4	NU		**S := 1 + 11 + 111 = 123**

Etapa **0**: Se citeşte numărul de termeni, **n**, în acest caz **n = 3** şi se iniţializează primul termen cu valoarea sa, adică **T_1 = 1** şi suma **S** cu valoarea primului termen, adică **S := T_1**;
Etapa **1**: Contorul **i** primeşte valoarea **2**, adică **i := 2**. Se verifică dacă se îndeplineşte condiţia **i <= n** şi deoarece **2 < 3**, condiţia este **adevărată**, se continuă pe ramura **DA**. Se calculează termenul curent **T_i** cu relaţia **T_i := T_{i-1} * 10 + 1**, adică **T_2 := 1 * 10 + 1**, deci **T_2 = 11**. Se calculează valoarea sumei cu ajutorul relaţiei **S := S + T_i**, adică **S : = 1 + 11 = 12** şi se modifică valoarea contorului **i** cu relaţia: **i := i + 1**, deci **i := 2 + 1 = 3**;
Etapa **2**: Cu valoarea nouă a contorul **i**, adică **i := 3** se verifică dacă se îndeplineşte condiţia **i <= n**. Deoarece **3 = 3**, condiţia este **adevărată**, se continuă pe ramura **DA**. Se calculează termenul curent **T_i** cu relaţia **T_i := T_{i-1} * 10 + 1**, adică **T_3 := 11 * 10 + 1**, deci **T_3 = 111**. Se calculează valoarea sumei cu ajutorul relaţiei **S := S + T_i**, adică **S : = 12 + 111 = 123** şi se modifică valoarea contorului **i** cu relaţia: **i := i + 1**, deci **i := 3 + 1 = 4**;
Etapa **3**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **4 > 3**, condiţia este **falsă** şi se continuă pe ramura **NU**, ceea ce înseamnă că se continuă cu afişarea valorii variabilei **S**, după care se încheie algoritmul.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.1e, iar programul C şi rularea acestuia sunt prezentate în figura 7.1f.

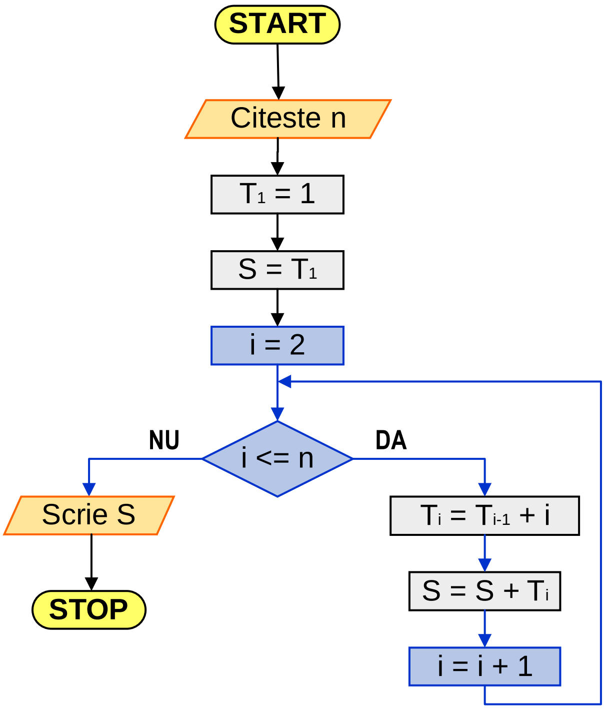
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n T[1] ← 1, S ← T[1] **Pentru** i = 2, n T[i] ← T[i] * 10 + 1 S ← S + T[i] **Sfârşit pentru** **Scrie** S **Sfârşit**

Figura 7.1e. Reprezentarea algoritmului pentru calculul sumei **S = 1 + 11 + 111 + 1111 + ...**

Programul C	Rularea programului
#include<stdio.h> **int main(void)** **{ int** i, n, S, T[30]; **printf(** "\n Introduceti n = " **);** **scanf(** "%d",&n **);** T[1]=1; S=T[1]; **for(** i = 2 ; i <= n ; i++ **)** **{** T[i] = T[i-1] * 10 + 1; S = S + T[i]; **}** **printf(** "\n Suma este S = %d",S **);** **}**	**Cazul 1:** Introduceti n = **3** Suma este S = 123 **Cazul 2:** Introduceti n = **6** Suma este S = 123456

Figura 7.1f. Programul C şi rularea acestuia pentru calculul sumei **S = 1 + 11 + 111 + 1111 + ...**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

7.2. Serii de numere reale

Se consideră $(a_n)_{n\geq 1}$ un şir de numere reale şi $(s_n)_{n\geq 1}$ şi un şir de numere reale definit prin relaţia: $s_n = \sum_{k=1}^{n} a_k$. Perechea de şiruri ($(a_n)_{n\geq 1}$, $(s_n)_{n\geq 1}$) poartă numele de serie de numere reale şi se notează: $a_1 + a_2 + \cdots + a_n + \cdots$ sau $\sum_{n=1}^{\infty} a_n$. Valorile a_1, a_2, ... se numesc termenii seriei, a_n se numeşte termenul general al seriei, iar şirul s_n se numeşte şirul sumelor parţiale ale seriei. Dacă şirul $(s_n)_{n\geq 1}$ are limita s (finită sau infinită), deci dacă există $s = \lim_{n\to\infty} s_n$ se scrie $s = \sum_{n=1}^{\infty} a_n$ sau $s = a_1 + a_2 + \cdots + a_n + \cdots$ şi se spune că suma seriei este s.

Egalitatea $s = \sum_{n=1}^{\infty} a_n$ reprezintă verbal: „suma seriei $\sum_{n=1}^{\infty} a_n$ este s şi semnifică faptul că şirul sumelor parţiale ale seriei are limita s.

Dacă şirul sumelor parţiale ale unei serii nu are limită, seria respectivă se numeşte *serie oscilantă*.

Dacă şirul sumelor parţiale ale unei serii este convergent seria se numeşte serie *convergentă*, în caz contrar seria se numeşte *divergentă*.

Fie seria $\sum_{n=1}^{\infty}\frac{1}{n(n+1)}$. Termenul general al acestei serii este $a_n = \frac{1}{n(n+1)}, n \geq 1$.

Şirul sumelor parţiale are termenul general $s_n = a_1 + a_2 + \cdots + a_n = \frac{1}{1\cdot2} + \frac{1}{2\cdot3} + \cdots + \frac{1}{n(n+1)},\ n \geq 1$.

Ţinând cont că: $\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}, k = 1,2,\dots,n$ se obţine: $s_n = \frac{1}{1} - \frac{1}{2} + \frac{1}{2} - \frac{1}{3} + \cdots + \frac{1}{n} - \frac{1}{n+1} = 1 - \frac{1}{n+1}$.

Aşadar: $\lim_{n\to\infty} s_n = \lim_{n\to\infty}\left(1 - \frac{1}{n+1}\right) = 1$, astfel că: $\sum_{n=1}^{\infty}\frac{1}{n(n+1)} = 1$.

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni, **n**;
2. Se iniţializează suma **S** cu **zero** (0 este element neutru pentru operaţia de adunare);
3. Se utilizează o instrucţiune de ciclare cu contor, contorul notat cu **i** ia valori de la **1** la **n**, în cadrul căruia se calculează valoarea termenului curent (în funcţie de **i**), precum şi calculul sumei pe baza relaţiei **S := S + a**$_i$.
4. După parcurgerea ciclului cu contor, se realizează afişarea valorii variabilei **S**.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.2a, iar programul C şi rularea acestuia sunt prezentate în figura 7.2b.

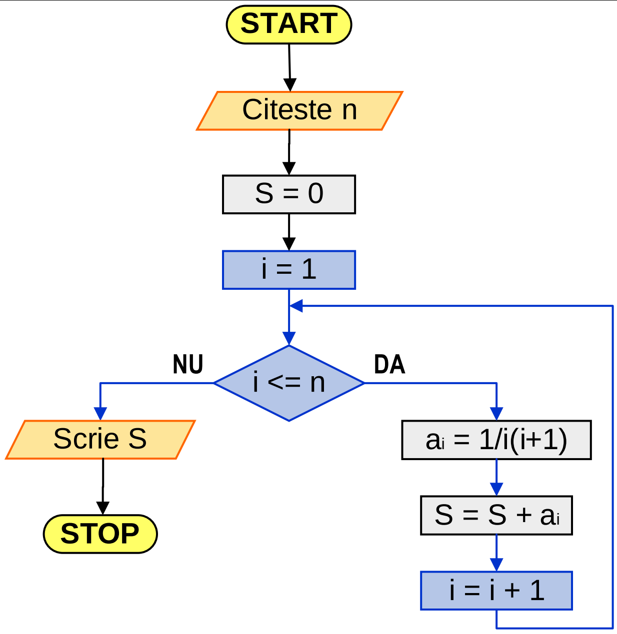
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n S ← 0 **Pentru** i = 1, n a[i] ← 1 / (i (i + 1)) S ← S + a[i] **Sfârşit pentru** **Scrie** S **Sfârşit**

Figura 7.2a. Reprezentarea algoritmului pentru calculul sumei

Programul C

```
#include<stdio.h>
int main(void)
{ int i, n; float S, a[1000];
  printf( "\n Introduceti n = " ); scanf( "%d",&n );
  S = 0;
  for( i = 1 ; i <= n ; i++ )
    { a[i] = 1. / ( i * ( i + 1 ));
      S = S + T[i];   }
  printf( "\n Suma este S = %f",S );
}
```

Rularea programului

Cazul 1:

Introduceti n = **500**
Suma este S = 0.998004

Cazul 2:

Introduceti n = **999**
Suma este S = 0.999000

Figura 7.2b. Programul C şi rularea acestuia pentru calculul sumei

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

7.3. Dezvoltări în serie de puteri

Seriile de puteri reprezintă o extindere a conceptului de funcţii polinomiale, dar şi o clasă particulară de serii de funcţii. Se numeşte serie de puteri o serie de funcţii $\sum_{i=0}^{\infty} f_i$ în care $f_i(x) = a_i \cdot x^i$ sau $f_i(x) = a_i \cdot (x-a)^i$, unde $a_0, a_1, a_2, \dots, a_i, \dots$ reprezintă coeficienţii seriei de puteri, iar $x \in \mathbb{R}$. Numărul a_i se numeşte coeficientul termenului de rang i.

Aşadar, o serie de puteri este o serie de forma:

$$a_0 + a_1 \cdot x + a_2 \cdot x^2 + \dots + a_i \cdot x^i + \dots = \sum_{i=0}^{\infty} a_i \cdot x^i$$

sau de forma:

$$a_0 + a_1 \cdot (x-a) + a_2 \cdot (x-a)^2 + \dots + a_i \cdot (x-a)^i + \dots = \sum_{i=0}^{\infty} a_i \cdot (x-a)^i$$

formă care poartă numele de serie de puteri centrată în punctul a.

O reprezentare a unei funcţii ca o sumă infinită de termeni calculaţi cu valorile derivatelor sale într-un punct poartă numele de serie Taylor. Dacă seria utilizează derivatele în zero, atunci ea poartă numele de serie MacLaurin.

Astfel, seria Taylor a unei funcţii reale sau complexe f care este funcţie indefinit derivabilă pe o vecinătate a unui număr real sau complex a, este seria de puteri:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \dots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

unde: $n!$ este factorialul lui n, iar $f^n(a)$ este a n-a derivată a lui f în punctul a.

Adesea, $f(x)$ este egală cu seria sa Taylor evaluată în x pentru orice x suficient de apropiat de a.

Exemple de serii de puteri şi serii Taylor remarcabile:

a) $\frac{1}{1-x} = 1 + x + x^2 + \dots + x^n, x \in (-1,1)$

b) $ln\frac{1+x}{1-x} = 2x\left[1 + \frac{x^2}{3} + \frac{x^4}{5} + \dots + \frac{x^{2n}}{2n+1}\right]$

c) $arctg\ x = \frac{x}{1} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots + (-1)^n \frac{x^{2n+1}}{2n+1}$

d) $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^n}{2n+1}$

e) $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} = \sum_{i=0}^{n} \frac{x^i}{i!}$

f) $e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} = \sum_{i=0}^{n} \cdot (-1)^i \cdot \frac{x^i}{i!}$

g) $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!} = \sum_{i=0}^{n} \frac{1}{i!}$

h) $\frac{1}{e} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \dots + \frac{(-1)^n}{n!} = \sum_{i=0}^{n} \cdot (-1)^i \cdot \frac{1}{i!}$

i) $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots = \sum_{i=0}^{n} (-1)^i \cdot \frac{x^{2i+1}}{(2i+1)!}$

j) $\cos x = x - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots = \sum_{i=0}^{n} (-1)^i \cdot \frac{x^{2i}}{(2i)!}$

7.3.1. Calculul funcţiei sin(x)

Se consideră dezvoltarea în serie de puteri: $\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$. Se cere să se determine şi să se afişeze valoarea aproximativă calculată pentru **n** termeni şi să se compare valoarea obţinută cu valoarea calculată a funcţiei **sinus** cu ajutorul funcţiei de bibliotecă.

Varianta I: Termenul general se calculează cu relaţia: $T_i = (-1)^{i+1} \cdot \frac{x^{2i-1}}{(2i-1)!}$

Se observă că fiecare termen conţine un produs, adică $(\boldsymbol{2i-1})!$. Astfel că, pentru fiecare termen, adică pentru fiecare **i**, trebuie să calculăm $(\boldsymbol{2i-1})!$, deci un produs.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3a., iar programul C şi rularea acestuia sunt prezentate în figura 7.3b.

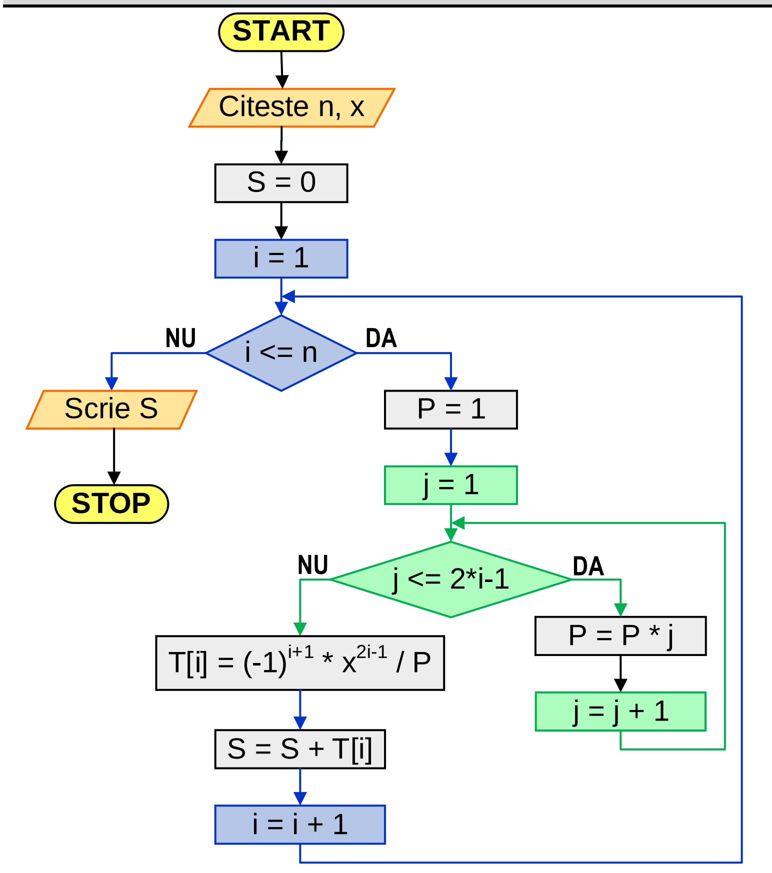
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n **Citeşte** x S ← 0 **Pentru** i = 1, n P ← 1 **Pentru** j = 1, 2*i-1 P ← P * j **Sfârşit pentru** T[i] = $(-1)^{i+1}$ * $x^{(2i-1)}$ / P S ← S + T[i] **Sfârşit pentru** **Scrie** S, sin(x) **Sfârşit**

Figura 7.3a. Reprezentarea algoritmului pentru calculul funcţiei **sin(x)** – varianta I

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni ai seriei de puteri, adică valoarea variabilei **n**;

2. Se citeşte argumentul funcţiei sinus, adică valoarea variabilei **x**;

3. Se iniţializează suma **S**, cu valoarea **0** (**zero** – element neutru pentru adunare);

4. Se utilizează o instrucţiune de ciclare cu contor (în acest caz contorul este variabila **i**), pentru calculul fiecărui termen $\mathbf{T_i}$ şi adunarea acestuia la sumă astfel:

4.1. Se iniţializează contorul **i**, cu valoarea **1**;

4.2. Se verifică condiţia $\boldsymbol{i <= n}$, adică se verifică dacă valoarea contorului este în intervalul $[\mathbf{1}, \boldsymbol{n}]$.

Dacă condiţia este **adevărată**, se trece la pasul următor (pasul 4.3). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare, algoritmul continuă cu pasul **5**;

4.3. Se iniţializează produsul **P** cu valoarea **1** (unu este element neutru pentru adunare);

4.4. Se utilizează o instrucţiune de ciclare cu contor (contorul fiind variabila **j**) pentru calculul factorialului;

4.5. Se calculează termenul T_i al cărui numitor este valoarea produsului **P** calculat anterior;

4.6. Se adună termenul T_i la suma **S**, utilizând o relaţie de forma **S := S +** T_i;

4.7. Se modifică valoarea contorului **i**, utilizând o relaţie de forma: **i := i + 1**. Se revine la pasul 4.2.

5. Se afişează valoarea calculată a sumei, precum şi valoarea calculată a funcţiei **sin(x)** utilizând funcţia din biblioteca matematică;

Programul C	Rularea programului
`#include<stdio.h>` `#include<math.h>` `int main(void)` `{ int i, j, n; double P, x, S, T[20];` `  printf( "\n Introduceti n = " ); scanf( "%d",&n );` `  printf( "\n Introduceti x = " ); scanf( "%lf",&x );` `  S = 0;` `  for( i = 1 ; i <= n ; i++ )` `    { P = 1.;` `      for( i = 1 ; i <= n ; i++ )` `              P = P * j ;` `      T[i] = pow( -1 , i+1 ) * pow( x , 2*i-1 ) / P;` `      S = S + T[i];   }` `  printf( "\n Suma este S = %11.9lf ",S );` `  printf( "\n sin(%6.4lf) = %11.9lf ", x,sin(x) );` `}`	**Cazul 1:** Introduceti n = **10** Introduceti x = **0** S = 0.000000000 sin(0.0000) = 0.000000000 **Cazul 2:** Introduceti n = **10** Introduceti x = **0.523598** S = 0.499999328 sin(0.5236) = 0.499999328 **Cazul 3:** Introduceti n = **10** Introduceti x = **1.047** S = 0.865926611 sin(1.0470) = 0.865926611

Figura 7.3b. Programul C şi rularea acestuia pentru calculul funcţiei **sin(x)** – varianta I

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Varianta II: Termenul general se calculează cu relaţia: $T_i = T_{i-1} \cdot \left(-\frac{x^2}{(2\cdot i-1)\cdot(2\cdot i-2)}\right)$

În acest caz, termenul general se exprimă în funcţie de termenul anterior, conform relaţiei de mai sus.

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni **n**;

2. Se citeşte argumentul **x**, al funcţiei sinus;

3. Se iniţializează primul termen cu valoarea sa, în acest caz **T[1] = x**;

4. Se iniţializează valoarea sumei cu valoarea primului termen, adică **S = T[1]**;

5. Se utilizează o instrucţiune de ciclare cu contor, acesta luând valori de la **2** la **n**, în cadrul căreia se realizează următoarele etape:

5.1. se iniţializează contorul **i**, cu valoarea **2**, astfel: **i = 2**;

5.2. se verifică condiţia **i <= n** prin care se verifică dacă valoarea curentă a contorului este mai mică decât **n** - numărul de termeni precizat anterior;

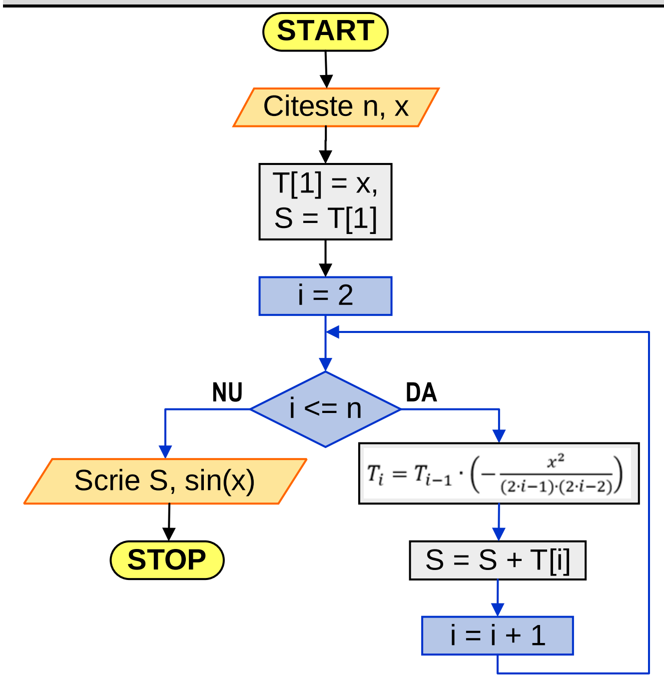
Dacă condiţia este **adevărată**, se calculează valoarea termenului curent cu relaţia de mai sus şi se adaugă la sumă valoarea termenului curent, utilizând relaţia **S = S + T[i]** şi se continuă cu incrementarea valorii contorului (etapa **5.3**). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul **6**;

5.3. Se incrementează valoarea contorului **i**, adică **i = i + 1**, după care se revine la pasul **5.2**;

6. Se afişează valoarea calculată a sumei **S**, respectiv se afişează valoarea funcţiei sinus cu argumentul **x**, adică **sin(x)**.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3c, iar programul C şi rularea acestuia sunt prezentate în figura 7.3d.

Schema logică



Limbajul pseudocod

```
Început
Citeşte n, x
T[1] = x
S ← T[1]
Pentru i = 2, n
    T[i] = T[i-1]*(-1*x*x / ((2*i - 1) * (2*i - 2)))
    S ← S + T[i]
Sfârşit pentru
Scrie S, sin(x)
Sfârşit
```

Figura 7.3c. Reprezentarea algoritmului pentru calculul funcţiei **sin(x)** – varianta II

Programul C

```c
#include<stdio.h>
#include<math.h>
int main(void)
{ int i, n; double x, S, T[20];
  printf( "\n Introduceti n = " ); scanf( "%d",&n );
  printf( "\n Introduceti x = " ); scanf( "%lf",&x );
  T[1] = x; S = T[1];
  for( i = 2 ; i <= n ; i++ )
    {  T[i] = T[i-1] * (-1*x*x / ( (2*i-1) * (2*i-2) ) );
       S = S + T[i];  }
  printf( "\n Suma este S = %11.9lf ",S );
  printf( "\n sin(%6.4lf) = %11.9lf ", x,sin(x) );
}
```

Rularea programului

Cazul 1:
Introduceti n = **10**
Introduceti x = **0.523598**
Suma este S = 0.499999328
sin(0.5236) = 0.499999328

Cazul 2:
Introduceti n = **10**
Introduceti x = **1.047**
Suma este S = 0.865926611
sin(1.0470) = 0.865926611

Figura 7.3d. Programul C şi rularea acestuia pentru calculul funcţiei **sin(x)** – varianta II

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

7.3.2. Calculul valorii constantei π

Varianta I: Se consideră următoarea relaţie de calcul pentru constanta **π** (serie Gregory):

$$\pi = 4 \cdot \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 4 \cdot \left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right)$$

Se doreşte să se calculeze şi să se afişeze valoarea aproximativă a lui **π**, pentru n termeni luaţi în considerare şi să se compare această valoare cu valoarea constantei definită în cadrul bibliotecii matematice a mediului de programare.

Formula termenului general este, în acest caz: $T_i = \frac{(-1)^i}{2i+1}$

Algoritmul de calcul, în acest caz, este următorul:

1. Se citeşte numărul de termeni ai seriei de puteri, adică valoarea variabilei **n**;

2. Se iniţializează suma **S**, cu valoarea **0** (**zero** – element neutru pentru adunare);

3. Se utilizează o instrucţiune de ciclare cu contor (în acest caz contorul este variabila **i**), pentru calculul fiecărui termen $\mathbf{T_i}$ şi adunarea acestuia la sumă astfel:

3.1. Se iniţializează contorul **i**, cu valoarea **0**;

3.2. Se verifică condiţia $\boldsymbol{i} <= \boldsymbol{n}$, adică se verifică dacă valoarea contorului este în intervalul $[\mathbf{1}, \boldsymbol{n}]$.

Dacă condiţia este **adevărată**, se trece la pasul următor (pasul 3.3). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare, algoritmul continuă cu pasul **4**;

3.3. Se calculează termenul $\mathbf{T_i}$ cu ajutorul relaţiei de mai sus;

3.4. Se adună termenul $\mathbf{T_i}$ la suma **S**, utilizând o relaţie de forma **S := S +** $\mathbf{T_i}$;

3.5. Se modifică valoarea contorului **i**, utilizând o relaţie de forma: **i := i + 1**. Se revine la pasul 3.2;

4. Se înmulţeşte valoarea calculată a sumei cu **4** şi se afişează. Se afişează valoarea constantei **π** din biblioteca matematică.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3e., iar programul C şi rularea acestuia sunt prezentate în figura 7.3f.

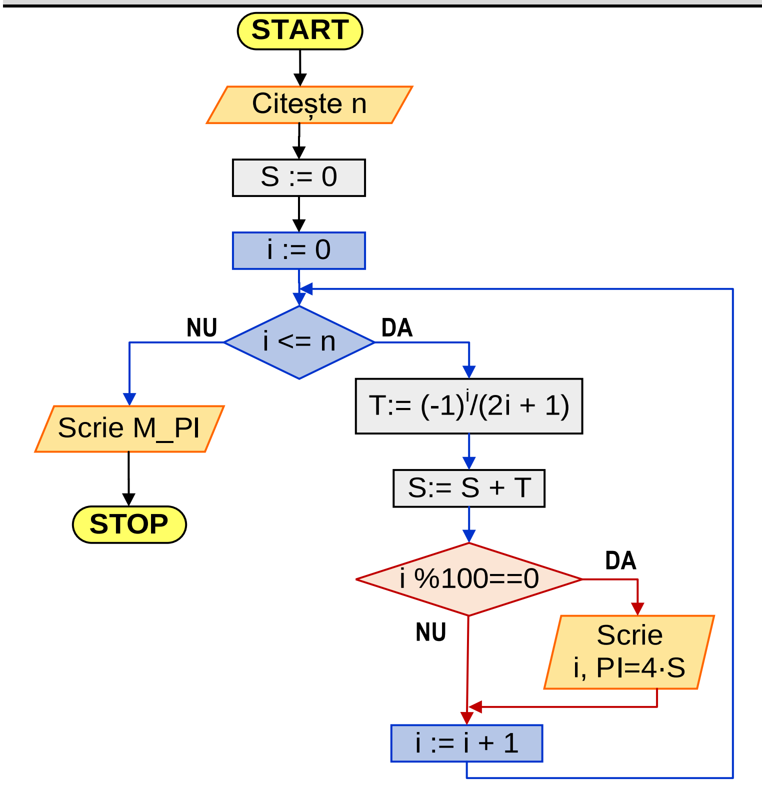
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n S ← 0 **Pentru** i = 0, n T = $(-1)^i$ / (2 * i + 1) S ← S + T **Dacă** i DIV 100 = 0 **atunci** **Scrie** i, 4 * S **Sfârşit dacă** **Sfârşit pentru** **Scrie** M_PI **Sfârşit**

Figura 7.3e. Reprezentarea algoritmului pentru calculul constantei **π** – varianta I

Programul C	Rularea programului
``#include<stdio.h>``` ```#include<math.h>``` ```int main(void)``` ```{ int i, n; double S, T;``` ``` printf("\n Introduceti n = "); scanf("%d",&n);``` ``` S = 0;``` ``` for(i = 0 ; i <= n ; i++)``` ``` { T = pow(-1. , i) / (2 * i + 1);``` ``` S = S + T;``` ``` if(i % 100 == 0)``` ``` printf("\n n = %4d \t PI = %11.9lf ",i, 4*S);``` ``` }``` ``` printf("\n Valoarea M_PI = %11.9lf ", M_PI);``` ```}```	Introduceti n = **1000** n = 0 PI = 4.000000000 n = 100 PI = 3.151493401 n = 200 PI = 3.146567747 n = 300 PI = 3.144914904 n = 400 PI = 3.144086415 n = 500 PI = 3.143588660 n = 600 PI = 3.143256546 n = 700 PI = 3.143019186 n = 800 PI = 3.142841093 n = 900 PI = 3.142702531 n = 1000 PI = 3.142591654 Valoarea M_PI = 3.141592654

Figura 7.3f. Programul C şi rularea acestuia pentru calculul constantei **π** – varianta I

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Varianta II: Se consideră următoarea relaţie de calcul pentru constanta **π**:

$$\pi = 2 \cdot \left(1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9} + \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{3 \cdot 5 \cdot 7 \cdot 9 \cdot 11} + \cdots\right)$$

Care mai poate fi scrisă:

$$\pi = 2 \cdot (1 + S)$$

unde:

$$S = \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9} + \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{3 \cdot 5 \cdot 7 \cdot 9 \cdot 11} + \cdots$$

Astfel, pentru calculul valorii aproximative a lui **π** trebuie să calculăm suma **S**, formula termenului general fiind:

$$T_i = \sum_{i=1}^{n} \frac{i!}{(2i+1)!!}.$$

Se observă că în formula termenului general apar două produse: factorial de **i** la numărător (notat cu **P1**) şi dublu factorial de **2i+1** la numitor (notat cu **P2**). Deci, pentru fiecare termen **T_i** este necesar să se calculeze două produse, algoritmul de calcul fiind următorul:

1. Se citeşte numărul de termeni ai sumei **S**, adică valoarea variabilei **n**;

2. Se iniţializează suma **S**, cu valoarea **0** (**zero** – element neutru pentru adunare);

3. Se utilizează o instrucţiune de ciclare cu contor (în acest caz contorul este variabila **i**), pentru calculul fiecărui termen **T_i** şi adunarea acestuia la sumă astfel:

3.1. Se iniţializează contorul **i**, cu valoarea **1**;

3.2. Se verifică condiţia $\boldsymbol{i} <= \boldsymbol{n}$, adică se verifică dacă valoarea contorului este în intervalul $[\mathbf{1}, \boldsymbol{n}]$.

Dacă condiţia este **adevărată**, se trece la pasul următor (pasul 3.3). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare, algoritmul continuă cu pasul **4**;

3.3. Se iniţializează produsele **P1** şi **P2** cu valoarea **1** (unu este element neutru pentru adunare);

3.4. Se utilizează o instrucţiune de ciclare cu contor (contorul fiind variabila **j**) pentru calculul factorialului, astfel:

3.4.1. Se iniţializează variabila **j** cu valoarea **1**;

3.4.2. Se verifică condiţia **j <= i**, dacă este **adevărată** se aplică relaţia **P1 := P1 * j**. Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul 3.5;

3.4.3. Se modifică valoarea contorului **j**, astfel: **j := j + 1**, după care se revine la pasul 3.4.2;

3.5. Se utilizează o instrucţiune de ciclare cu contor (contorul fiind variabila **k**) pentru calculul dublului factorial, astfel:

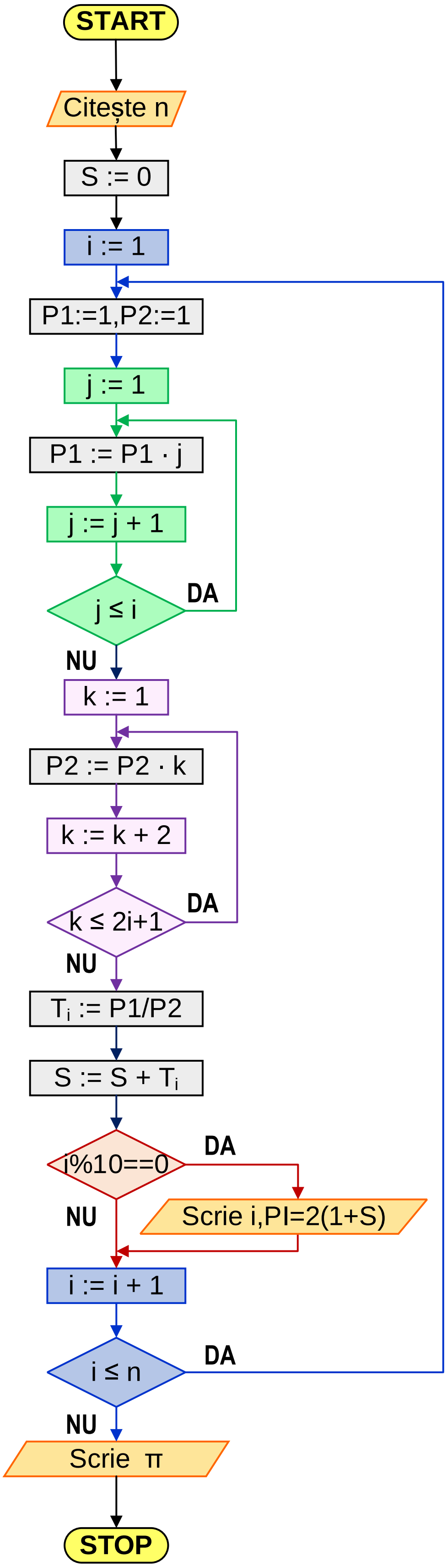
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n S ← 0 **Pentru** i = 1, n P1 ← 1 P2 ← 1 **Pentru** j = 1, i P1 ← P1 * j **Sfârşit pentru** **Pentru** k = 1, 2 * i + 1, 2 P2 ← P2 * k **Sfârşit pentru** T[i] = P1 / P2 S ← S + T[i] **Dacă** i DIV 10 = 0 **atunci** **Scrie** i, 2 * (1 + S) **Sfârşit dacă** **Sfârşit pentru** **Scrie** S, M_PI **Sfârşit**

Figura 7.3g. Reprezentarea algoritmului pentru calculul constantei π – varianta II

3.5.1. Se iniţializează variabila **k** cu valoarea **1**;

3.5.2. Se verifică condiţia **k <= 2*i+1**, dacă este **adevărată** se aplică relaţia **P2 := P2 * k**. Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul 3.6;

3.5.3. Se modifică valoarea contorului **k**, astfel: **k := k + 2**, după care se revine la pasul 3.5.2;

3.6. Se calculează termenul $\mathbf{T_i}$ astfel: $\mathbf{T_i}$ **:= P1 / P2** calculat anterior;

3.6. Se adună termenul $\mathbf{T_i}$ la suma **S**, utilizând o relaţie de forma **S := S +** $\mathbf{T_i}$;

3.7. Se modifică valoarea contorului **i**, utilizând o relaţie de forma: **i := i + 1**. Se revine la pasul 3.2;

4. Se afişează valoarea relaţiei 2(1+S), precum şi valoarea calculată a constantei **π** din biblioteca matematică.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3g., iar programul C şi rularea acestuia sunt prezentate în figura 7.3h.

<table>
<tr><th>Programul C</th><th>Rularea programului</th></tr>
<tr><td>

```
#include<stdio.h>
#include<math.h>
int main(void)
{ int i, j, k, n; double P1, P2, S, T[100];
  printf( "\n Introduceti n = " ); scanf( "%d",&n );
  S = 0;
  for( i = 1 ; i <= n ; i++ )
   { P1 = 1; P2 = 1;
     for( j = 1 ; j <= i ; j++ )   P1 = P1 * j;
     for( k = 1 ; k <= 2*i+1 ; k = k + 2 )   P2 = P2 * k;
     T[i] = P1 / P2;  S=S+T[i];
     if( i % 10 == 0 )
        printf( "\n i = %4d \t PI = %11.9lf ",i,2*(1+S) );
   }
  printf( "\n Valoarea M_PI = %11.9lf ",M_PI );
}
```

</td><td>

```
Introduceti n = 50

 i =   10       PI = 3.141106022
 i =   20       PI = 3.141592299
 i =   30       PI = 3.141592653
 i =   40       PI = 3.141592654
 i =   50       PI = 3.141592654
 Valoarea M_PI = 3.141592654
```

</td></tr>
</table>

Figura 7.3h. Programul C şi rularea acestuia pentru calculul constantei **π** – varianta II

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Varianta III: Termenul general se calculează cu relaţia: $T_i = T_{i-1} \cdot \left(\frac{i}{2i+1}\right)$

În acest caz, termenul general se exprimă în funcţie de termenul anterior, conform relaţiei de mai sus.

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni **n**;

2. Se iniţializează primul termen cu valoarea sa, în acest caz **T[1] = 1/3**;

3. Se iniţializează valoarea sumei cu valoarea primului termen, adică **S = T[1]**;

4. Se utilizează o instrucţiune de ciclare cu contor, acesta luând valori de la **2** la **n**, în cadrul căreia se realizează următoarele etape:

4.1. se iniţializează contorul **i**, cu valoarea **2**, astfel: **i = 2**;

4.2. se verifică condiţia **i <= n** prin care se verifică dacă valoarea curentă a contorului este mai mică decât **n** - numărul de termeni precizat anterior;

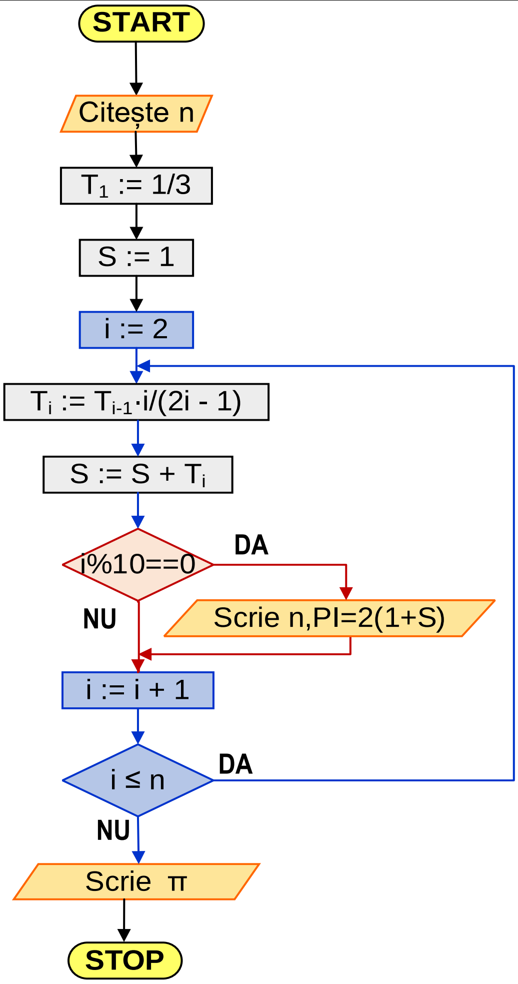
Dacă condiţia este **adevărată**, se calculează valoarea termenului curent cu relaţia de mai sus şi se adaugă la sumă valoarea termenului curent, utilizând relaţia **S = S + T[i]** şi se continuă cu incrementarea valorii contorului (etapa **4.3**). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul **5**;

4.3. Se incrementează valoarea contorului **i**, adică **i = i + 1**, după care se revine la pasul **4.2**;

5. Se afişează valoarea produsului **2*(1 + S)**, respectiv se afişează valoarea constantei **π** din biblioteca matematică.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3i, iar programul C şi rularea acestuia sunt prezentate în figura 7.3j.

Schema logică



Limbajul pseudocod

```
Început
Citeşte n
T[1] ← 1. / 3
S ← T[1]
Pentru i = 2, n
    T[i] ← T[i-1] * i / ( 2 * i - 1)
    S ← S + T[i]
    Dacă i DIV 10 = 0 atunci
          Scrie i, 2 * (1 + S)
    Sfârşit dacă
Sfârşit pentru
Scrie  M_PI
Sfârşit
```

Figura 7.3i. Reprezentarea algoritmului pentru calculul constantei **π** – varianta III

Programul C

```
#include<stdio.h>
#include<math.h>
int main(void)
{ int i, n; double S, T[100];
  printf( "\n Introduceti n = " ); scanf( "%d",&n );
  T[1] = 1. / 3; S = T[1];
  for( i = 2 ; i <= n ; i++ )
    { T[i] = T[i-1] * i / (2*i+1); S = S + T[i];
          if( i % 10 == 0 )
            printf( "\n n = %4d \t PI = %11.9lf ",i,2*(1+S) );   }
  printf( "\n Valoarea M_PI = %11.9lf ",M_PI );
}
```

Rularea programului

```
Introduceti n = 50

 n =   10       PI = 3.141106022
 n =   20       PI = 3.141592299
 n =   30       PI = 3.141592653
 n =   40       PI = 3.141592654
 n =   50       PI = 3.141592654
 Valoarea M_PI = 3.141592654
```

Figura 7.3j. Programul C şi rularea acestuia pentru calculul constantei **π** – varianta III

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Capitolul 8. Aplicaţii în domeniul ingineriei mecanice

8.1. Transformări de coordonate

În geometrie, dar şi în alte domenii (mecanica clasică, topografia, etc) se utilizează sisteme de coordonate. Un sistem de coordonate reprezintă o modalitate prin care unui punct i se asociază în mod unic o mulţime ordonată de numere reale, ce poartă denumirea de coordonatele punctului. Cu ajutorul acestora se defineşte poziţia punctului în raport cu originea sistemului de coordonate.

În spaţiul euclidian sunt necesare trei coordonate (abscisa, ordonata şi cota), în plan sunt necesare două coordonate (abscisa şi ordonata), iar pentru localizarea punctelor pe o dreaptă este necesară o singură coordonată.

Termenul de coordonată a fost introdus pentru prima oară de **Gottfried Wilhelm Freiherr von Leibniz** în 1692.

În studiul mecanicii clasice (newtoniene) se utilizează trei sisteme de coordonate: **cartezian**, **cilindric** şi **sferic**. În figura 8.1a sunt ilustraţi parametri în cazul celor trei sisteme de coordonate: **cartezian** (stânga), **cilindric** (mijloc), respectiv **sferic** (dreapta).

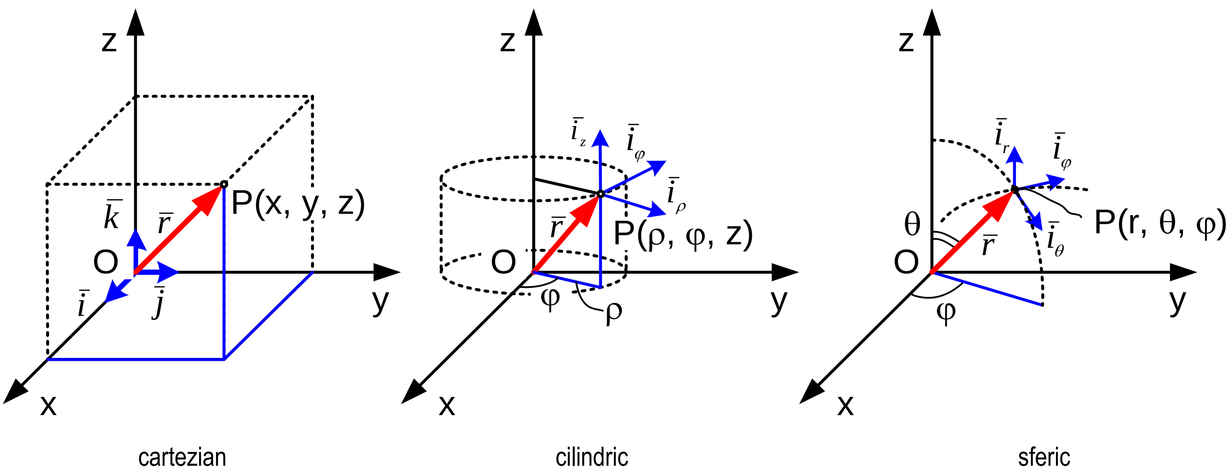


Figura 8.1a. Sisteme de coordonate

În sistemul de coordonate **cartezian** se aleg trei axe de coordonate, **Ox**, **Oy**, **Oz**, perpendiculare între ele. În acest caz, coordonatele punctului **P** vor fi **x**, **y**, **z**. Notând cu $\bar{\imath}, \bar{\jmath}, \bar{k}$ versorii celor trei axe, orice vector de poziţie $\bar{r}$ poate fi scris sub forma:

$$\bar{r} = x \cdot \bar{\imath} + y \cdot \bar{\jmath} + z \cdot \bar{k} \tag{8.1}$$

În sistemul de coordonate **cilindrice**, coordonatele sunt proiecţia **ρ** a vectorului $\bar{r}$ pe planul **xOy**, unghiul **φ** dintre axa **Ox** şi proiecţia vectorului $\bar{r}$ pe planul **xOy** şi **z**. Coordonatele se numesc **cilindrice** deoarece, păstrând raza cilindrului constantă şi modificând unghiul **φ** (de la 0 la 360°) şi coordonata **z**, se generează toate punctele unui cilindru.

În sistemul de coordonate **sferic**, coordonatele sunt raza **r**, unghiul **φ** dintre axa **Ox** şi proiecţia vectorului $\bar{r}$ pe planul **xOy**, respectiv unghiul **θ** dintre raza $\bar{r}$ şi axa **Oz**. Coordonatele se numesc **sferice** deoarece modificând raza de la **0** la **r** şi modificând coordonatele **θ** (de la 0 la 180°), respectiv **φ** (de la 0 la 360°) se pot genera toate punctele unei sfere.

Între coordonatele carteziene, cilindrice şi sferice sunt valabile relaţiile:

$$\begin{cases} x = \rho \cdot cos\varphi = r \cdot sin\theta \cdot cos\varphi; \\ y = \rho \cdot sin\varphi = r \cdot sin\theta \cdot sin\varphi; \\ \quad z = z = r \cdot cos\theta; \end{cases} \tag{8.2}$$

O problemă de interes practic constă în determinarea coordonatelor cilindrice şi sferice atunci când se cunosc coordonatele carteziene.

Se consideră aşadar cunoscute coordonatele carteziene **x**, **y**, **z** şi se doreşte determinarea coordonatelor cilindrice: **ρ**, **φ**, **z**, respectiv a coordonatelor sferice: **r**, **φ**, **θ**.

Pentru determinarea coordonatelor cilindrice se utilizează relaţiile:

$$\begin{cases} \rho = x^2 + y^2 \\ \varphi = arctg\left(\frac{y}{x}\right) \\ z = z \end{cases} \tag{8.3}$$

Pentru determinarea coordonatelor sferice se utilizează relaţiile:

$$\begin{cases} r = x^2 + y^2 + z^2 \\ \varphi = arctg\left(\frac{y}{x}\right) \\ \theta = arccos\left(\frac{z}{r}\right) \end{cases} \tag{8.4}$$

Algoritmul de calcul se compune din următorii paşi:
- citirea coordonatelor carteziene: **x**, **y**, **z**;
- calculul şi afişarea coordonatelor cilindrice: **ρ**, **φ**, **z**, pe baza relaţiilor (8.3);
- calculul şi afişarea coordonatelor sferice: **r**, **φ**, **θ**, pe baza relaţiilor (8.4).

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 8.1b, iar programul C şi rularea acestuia sunt prezentate în figura 8.1c.

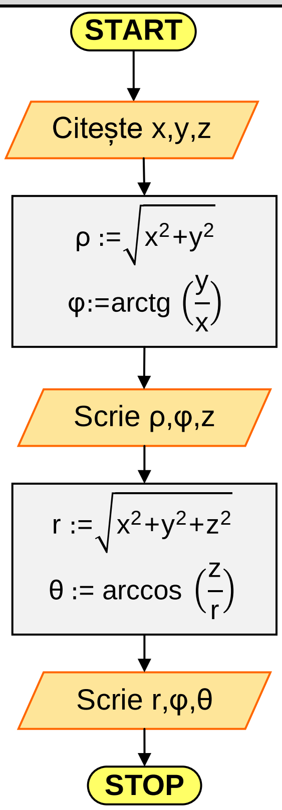
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** **x**, **y**, **z** ro ← sqrt(x^2 + y^2) fi ← arctg(y/x) **Scrie** ro, fi, z r ← sqrt(x^2 + y^2+z^2) fi ← arctg(y/x) teta ← arccos(z/r) **Scrie** r, fi, teta **Sfârşit**

Figura 8.1b. Reprezentarea algoritmului pentru calculul coordonatelor cilindrice şi sferice

Programul C şi rularea acestuia

```
#include<stdio.h>
#include<math.h>
int main(void)
{ float x, y, z, ro, r, fi, teta;
  printf( "\n Coordonatele carteziene:" );
  printf( "\n Coordonata x = " );
  scanf(  "%f",&x );
  printf( " Coordonata y = " );
  scanf( "%f",&y );
  printf( " Coordonata z = " );
  scanf( "%f",&z );
  ro = sqrt( x*x+y*y );
  fi = atan( y/x );
  printf( "\n Coordonate cilindrice:" );
  printf( "\n Coordonata ro = %6.3f",ro );
  printf( "\n Coordonata fi = %6.3f [rad] \t %6.3f [grad]",fi,fi*180/M_PI );
  printf( "\n Coordonata z  = %6.3f",z );
  r = sqrt( x*x+y*y+z*z );
  teta = acos( z/r );
  printf( "\n\n Coordonate sferice:" );
  printf(" \n Coordonata r    = %6.3f",r );
  printf( "\n Coordonata fi   = %6.3f [rad] \t %6.3f [grad]",fi,fi*180/M_PI );
  printf( "\n Coordonata teta = %6.3f [rad] \t %6.3f [grad]",teta,teta*180/M_PI );
}
```

Rularea programului:

Coordonatele carteziene:
Coordonata x = **3**
Coordonata y = **4**
Coordonata z = **5**

Coordonate cilindrice:
Coordonata ro = 5.000
Coordonata fi = 0.927 [rad]   53.130 [grad]
Coordonata z = 5.000

Coordonate sferice:
Coordonata r   = 7.071
Coordonata fi   = 0.927 [rad]   53.130 [grad]
Coordonata teta = 0.785 [rad]   45.000 [grad]

Figura 8.1c. Programul C şi rularea acestuia pentru calculul coordonatelor cilindrice şi sferice

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.2. Calculul simetricului unui punct faţă de o dreaptă, în plan

Se consideră un punct **P(x_P, y_P)** şi se doreşte determinarea coordonatelor punctului **S(x_S, y_S)** care este simetricul punctului **P** faţă de o dreaptă de ecuaţie **ax + by + c = 0**.

Pentru determinarea coordonatelor punctului **S**, se determină distanţa **d** de la punctul **P** la dreaptă, astfel:

$$d = \frac{|a \cdot x_P + b \cdot y_P + c|}{\sqrt{a^2+b^2}} \tag{8.5}$$

respectiv cu **(m_1, m_2)** cosinuşii directori ai vectorului **n(a, b)** normal la dreaptă, conform următoarelor relaţii:

$$m_1 = \pm\frac{a}{\sqrt{a^2+b^2}}, \quad m_2 = \pm\frac{b}{\sqrt{a^2+b^2}} \tag{8.6}$$

În relaţiile (8.6) se alege semnul **minus** dacă $a \cdot x_P + b \cdot y_P + c > 0$, respectiv semnul **plus** dacă $a \cdot x_P + b \cdot y_P + c < 0$.

Punctul **S** va avea coordonatele:

$$\begin{cases} x_S = x_P + 2 \cdot d \cdot m_1 \\ y_S = y_P + 2 \cdot d \cdot m_2 \end{cases} \tag{8.7}$$

Algoritmul de calcul al coordonatelor punctului simetric presupune parcurgerea următoarelor etape:
- citirea coordonatelor punctului **P**: **x_P**, **y_P**;
- citirea coeficienţilor dreptei **d**: **a**, **b**, **c**;
- se calculează variabilele **m_1** şi **m_2**, pe baza relaţiilor (8.6) şi ţinând cont de valoarea expresiei $a \cdot x_P + b \cdot y_P + c$;
- se calculează coordonatele simetricului (punctul **S**) **x_S**, respectiv **y_S** cu ajutorul relaţiilor (8.7).

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.2a, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.2b.

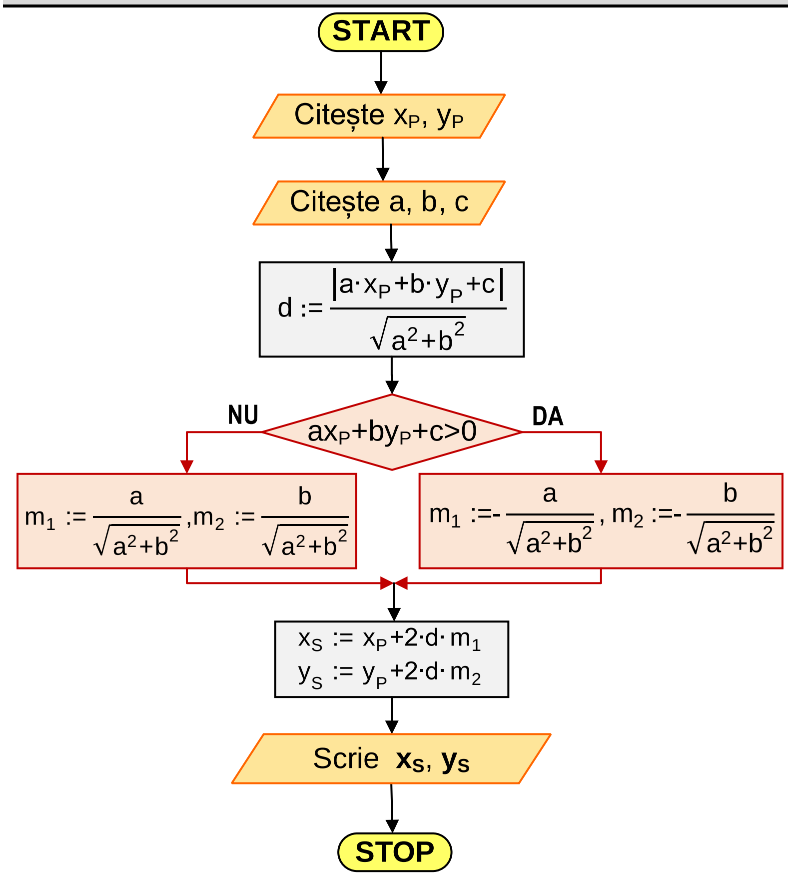
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** x_P, y_P **Citeşte** a, b, c d ← abs(a*x_P+b*y_P+c)/sqrt(a^2+b^2) **Dacă** a*x_P+b*y_P+c > 0 **atunci** $m_1 \leftarrow -\frac{a}{\sqrt{a^2+b^2}}, m_2 \leftarrow -\frac{b}{\sqrt{a^2+b^2}}$ **altfel** $m_1 \leftarrow +\frac{a}{\sqrt{a^2+b^2}}, m_2 \leftarrow +\frac{b}{\sqrt{a^2+b^2}}$ **Sfârşit dacă** $x_S \leftarrow x_P + 2 \cdot d \cdot m_1$ $y_S \leftarrow y_P + 2 \cdot d \cdot m_2$ **Scrie** x_S, y_S **Sfârşit**

Figura 8.2a. Reprezentarea algoritmului pentru calculul simetricului unui punct faţă de o dreaptă

Programul C	Rularea programului
<pre>#include<stdio.h> #include<math.h> int main(void) { float xp, yp, a, b, c, d, m1, m2, xs, ys; printf("\n Coordonatele punctului P:"); printf("\n Coordonata xp = "); scanf("%f",&xp); printf(" Coordonata yp = "); scanf("%f",&yp); printf("\n Coeficientii dreptei:"); printf("\n Coeficient a = "); scanf("%f",&a); printf(" Coeficient b = "); scanf("%f",&b); printf(" Coeficient c = "); scanf("%f",&c); d = abs(a*xp+b*yp+c)/sqrt(a*a+b*b); if(a*xp+b*yp+c>0) { m1 = -a/sqrt(a*a+b*b); m2 = -b/sqrt(a*a+b*b); } else { m1 = a/sqrt(a*a+b*b); m2 = b/sqrt(a*a+b*b); } xs = xp + 2*d*m1; ys = yp + 2*d*m2; printf("\n Coordonatele punctului S:"); printf("\n Coordonata xs = %6.3f",xs); printf("\n Coordonata ys = %6.3f",ys); }</pre>	Coordonatele punctului P: Coordonata xp = **1** Coordonata yp = **3** Coeficientii dreptei: Coeficient a = **-1** Coeficient b = **1** Coeficient c = **0** Coordonatele punctului S: Coordonata xs = 3.000 Coordonata ys = 1.000

Figura 8.2b. Programul C şi rularea acestuia pentru calculul simetricului unui punct faţă de o dreaptă

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.3. Calculul coordonatelor unui punct rotit în sens trigonometric, în plan

Se consideră un punct **P(x_P, y_P)** şi se doreşte determinarea noilor coordonate ale punctului **P(x_{PR}, y_{PR})**, obţinute în urma rotaţiei acestuia, în sens trigonometric, în jurul unui punct **O(x_O, y_O)** (numit **centru de rotaţie**) după un unghi fix **α** (numit **unghi de rotaţie**).

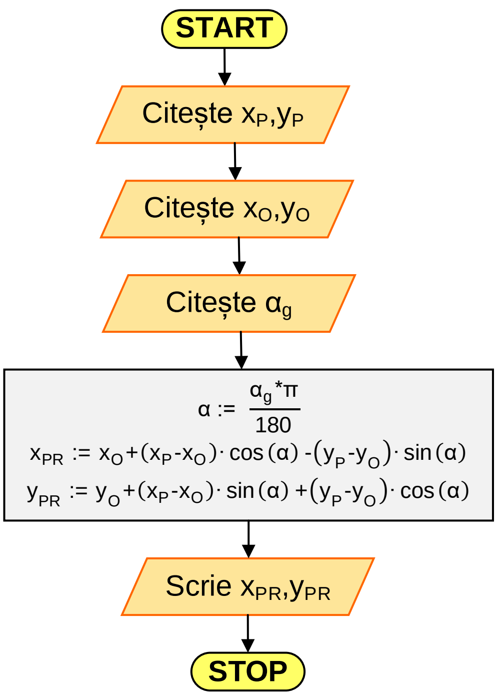
În acest caz, coordonatele noi ale punctului **P(x_{PR}, y_{PR})**, rezultate în urma rotaţiei punctului în sens trigonometric în jurul centrului de rotaţie **O(x_O, y_O)**, cu unghiul fix **α** se determină cu ajutorul relaţiilor:

$$\begin{cases} x_{PR} = x_O + (x_P - x_O)\cdot\cos(\alpha) - (y_P - y_O)\cdot\sin(\alpha) \\ y_{PR} = y_O + (x_P - x_O)\cdot\sin(\alpha) + (y_P - y_O)\cdot\cos(\alpha) \end{cases} \tag{8.8}$$

Algoritmul de calcul al coordonatelor obţinute în urma rotaţiei presupune parcurgerea următoarelor etape:
- citirea coordonatelor punctului **P**: x_P, y_P;
- citirea coordonatelor punctului **O**: x_O, y_O;
- citirea unghiului **α**;
- se calculează şi se afişează noile coordonate ale punctului **P(x_{PR}, y_{PR})**, determinate cu ajutorul relaţiilor (8.8).

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.3a, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.3b.

Schema logică



Limbajul pseudocod

Început
  Citeşte x_P, y_P, x_O, y_O, alfa
  alfar ← alfa*M_PI/180.0
  $x_{PR} \leftarrow x_O + (x_P - x_O)\cdot\cos(alfar) - (y_P - y_O)\cdot\sin(alfar)$
  $y_{PR} \leftarrow y_O + (x_P - x_O)\cdot\sin(alfar) + (y_P - y_O)\cdot\cos(alfar)$
  Scrie x_{PR}, y_{PR}
Sfârşit

Figura 8.3a. Reprezentarea algoritmului pentru calculul coordonatelor unui punct rotit cu unghiul **α** în jurul unui punct **O(x_O, y_O)**

Programul C

```c
#include<stdio.h>
#include<math.h>
int main(void)
{ float xp, yp, xo, yo, xpr, ypr, alfag, alfar ;
  printf( "\n Coordonatele punctului P:" );
  printf( "\n Coordonata xp = " ); scanf( "%f",&xp );
  printf( " Coordonata yp = " ); scanf( "%f",&yp );
  printf( "\n Coordonatele punctului O:" );
  printf( "\n Coordonata xo = " ); scanf( "%f",&xo );
  printf( " Coordonata yo = " ); scanf( "%f",&yo );
  printf( "\n Unghiul a [grade] = " ); scanf( "%f",&alfag );
  alfar = alfag*M_PI/180.0;
  xpr = xo + (xp-xo)*cos(alfar)-(yp-yo)*sin(alfar) ;
  ypr = yo + (xp-xo)*sin(alfar)+(yp-yo)*cos(alfar) ;
  printf( "\n Coordonatele punctului P rotit:" );
  printf( "\n Coordonata xpr = %6.3f",xpr );
  printf( "\n Coordonata ypr = %6.3f",ypr );
}
```

Rularea programului

```
Coordonatele punctului P:
Coordonata xp = 10
Coordonata yp = 10

Coordonatele punctului O:
Coordonata xo = 0
Coordonata yo = 0

Unghiul a [grade] = 90

Coordonatele punctului P rotit:
Coordonata xpr = -10.000
Coordonata ypr = 10.000
```

Figura 8.3b. Programul C şi rularea acestuia pentru calculul coordonatelor unui punct rotit cu unghiul **α** în jurul unui punct **O(x_O, y_O)**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.4. Calculul ariei unui poligon neregulat

Se consideră un poligon neregulat cu „**n**” laturi (deci „**n**” vârfuri) şi se cere să se calculeze aria acestuia (figura 8.4a).

Pentru calculul ariei unui poligon cu vârfurile A_1, A_2, A_3, ..., A_n ($A_1 = A_n$) se consideră un punct **O** ale cărui coordonate se consideră ($x_O = 0$, $y_O = 0$), arbitrar ales în planul poligonului.

Se „împarte” poligonul în triunghiuri PA_iA_{i+1} şi se calculează „aria cu semn” a fiecărui triunghi, notată cu T_i, fiind posibile două cazuri:

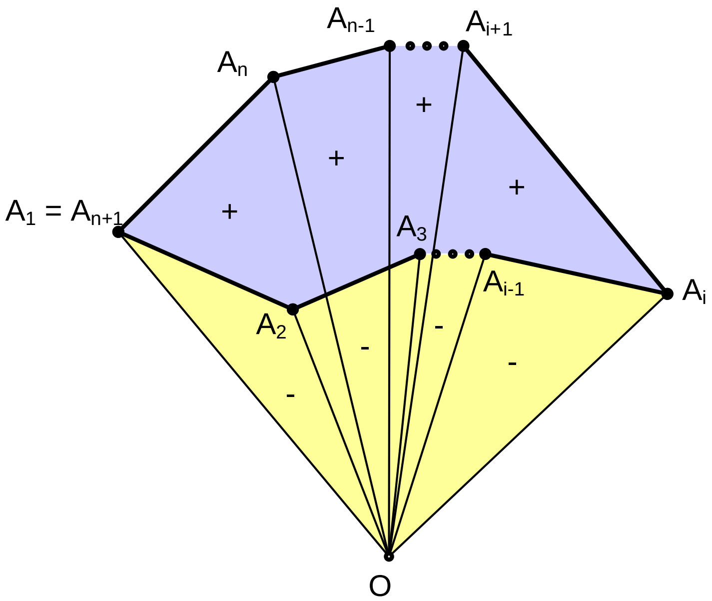


Figura 8.4a. Calculul ariei unui poligon neregulat

- dacă vârfurile poligonului sunt orientate trigonometric, pentru fiecare latură orientată „spre dreapta”, aria triunghiului va fi **negativă**, iar pentru fiecare latură orientată „spre stânga”, aria triunghiului va fi **pozitivă**;
- dacă vârfurile poligonului sunt orientate în sens orar, pentru fiecare latură orientată „spre dreapta”, aria triunghiului va fi **pozitivă**, iar pentru fiecare latură orientată „spre stânga”, aria triunghiului va fi **negativă**.

Ariile triunghiurilor T_i se calculează cu relaţia:

$$T_i = \frac{1}{2} \cdot \begin{vmatrix} 0 & 0 & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix} = \frac{1}{2} \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \qquad (8.9)$$

Indiferent de situaţie, însumând ariile triunghiurilor T_i se obţine aria poligonului ale cărui vârfuri sunt punctele A_1, A_2, A_3, ..., A_n :

$$A = \sum_{i=1}^{n} T_i = \frac{1}{2} \cdot \sum_{i=1}^{n} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \qquad (8.10)$$

Relaţia (8.10) mai poate fi scrisă şi sub forma:

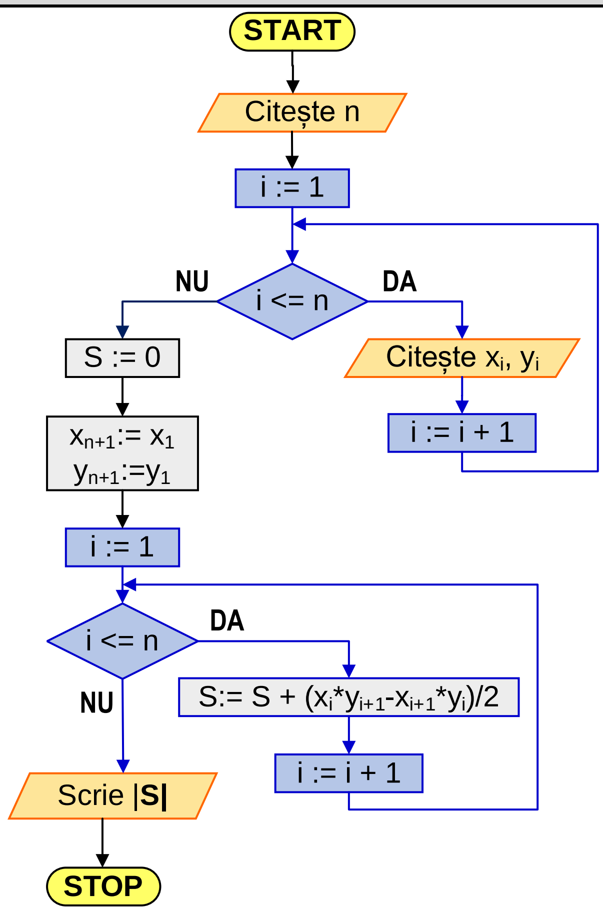
$$A = \frac{1}{2} \cdot |[(x_1 - x_2)(y_1 + y_2) + (x_2 - x_3)(y_2 + y_3) + \cdots (x_n - x_1)(y_n + y_1)]| \qquad (8.11)$$

Pe baza relaţiilor prezentate, se poate concepe un algoritm pentru calculul ariei unui poligon oarecare, astfel:

- se citeşte **n** = numărul de puncte – vârfurile poligonului;
- se citesc coordonatele x_i, respectiv y_i ale fiecărui vârf al poligonului;
- deoarece calculul ariei se reduce la calculul unei sume, se iniţializează variabila care reprezintă suma cu zero, **S = 0**;
- se atribuie coordonatelor punctului A_{n+1} coordonatele punctului A_1, astfel: $x_{n+1} = x_1$, $y_{n+1} = y_1$;
- se repetă operaţia de calcul a sumei, utilizând relaţia: $S = S + \frac{1}{2} \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$, pentru valori ale lui **i** de la **1** la **n**;
- deoarece valoarea sumei obţinute poate fi pozitivă sau negativă, se calculează şi se afişează valoarea modulului sumei obţinute;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.4b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.4c.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte n
  Pentru i = 1, n
     Citeşte x[i]
     Citeşte y[i]
  Sfârşit pentru
     S ← 0
     x[n+1] ← x[1]
     y[n+1] ← y[1];
  Pentru i = 1, n
     S ← S + (x[i] * y[i+1] - x[i+1] * y[i]) / 2.;
  Sfârşit pentru
  Scrie |S|
Sfârşit
```

Figura 8.4b. Reprezentarea algoritmului pentru calculul ariei unui poligon neregulat

Programul C	Rularea programului

```c
#include<stdio.h>
#include<math.h>
int main(void)
{ int n, i; float x[20], y[20], S;
  printf( "\n Introduceti numarul de varfuri, n = " );
  scanf( "%d",&n );
  for( i = 1 ; i <= n ; i++ )
    { printf( "\n x[%d] = ",i ); scanf( "%f",&x[i] );
      printf( "\n y[%d] = ",i ); scanf( "%f",&y[i] );
    }
  S=0; x[n+1] = x[1]; y[n+1] = y[1];
  for( i = 1 ; i <= n ; i++ )
     S = S + (x[i] * y[i+1] - x[i+1] * y[i]) / 2.;
  printf( "\n Aria poligonului este: S = %f",fabs(S) );
}
```

Rularea programului:

Introduceti numarul de varfuri, n = **3**
x[1] = **0**
y[1] = **0**
x[2] = **2**
y[2] = **0**
x[3] = **0**
y[3] = **2**
Aria poligonului este: S = 2.000000

Figura 8.4c. Programul C şi rularea acestuia pentru calculul ariei unui poligon neregulat

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.5. Calculul centrului de masă al unui poligon neregulat

Centrul de masă, cunoscut şi sub denumirea de **centroid**, sau **centru de greutate** (într-un câmp gravitaţional uniform), al unui corp este definit ca poziţia „medie ponderată” a masei într-un corp. În Mecanică se utilizează centrul de masă, considerându-se masa corpului concentrată în acel punct, ecuaţiile utilizate în dinamică fiind aplicabile în raport cu acest punct.

Între centrul de masă (care este o noţiune mai generală) şi centru de greutate (definit într-un câmp gravitaţional) există o diferenţă. Centrul de greutate se defineşte ca fiind punctul în raport cu care suma momentelor forţelor gravitaţionale este zero (centrul forţelor paralele). În câmp gravitaţional uniform cele două puncte sunt identice.

Printre proprietăţile centrului de masă, amintim:

a) poziţia centrului de masă nu depinde de sistemul de referinţă ales, fiind un punct intrinsec al sistemului material;

b) dacă corpul (sistemul de puncte materiale) are un plan, o axă sau un punct de simetrie, atunci centrul de masă se găseşte în acel plan, pe aceea axă sau în punctul de simetrie.

În plan, în cazul unui poligon, centrul de greutate poate fi calculat cu următoarele formule:

$$XC = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1}) \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \tag{8.12}$$

$$YC = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1}) \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \tag{8.13}$$

A fiind aria poligonului, determinată în subcapitolul precedent.

Se consideră un poligon închis, definit de „ ” puncte: **A$_1$**, **A$_2$**, **A$_3$**, ..., **A$_n$**, figura 8.5a. Se doreşte să se calculeze coordonatele centrului de masă al poligonului.

Pe baza relaţiilor prezentate, se poate concepe un algoritm pentru calculul coordonatelor, astfel:

- se citeşte **n** = numărul de puncte – vârfurile poligonului;
- se citesc coordonatele **x$_i$**, respectiv **y$_i$** ale fiecărui vârf al poligonului;
- deoarece calculul ariei se reduce la calculul unei sume, se iniţializează variabila care reprezintă aria cu zero, **A = 0**;
- deoarece calculul coordonatelor **X$_C$**, respectiv **Y$_C$** se reduce la calculul unei sume, se iniţializează cele două variabile cu zero, astfel: **X$_C$ = 0**, **Y$_C$ = 0**;
- se atribuie coordonatelor punctului **A$_{n+1}$** coordonatele punctului **A$_1$**, astfel: **x$_{n+1}$ = x$_1$**, **y$_{n+1}$ = y$_1$**;
- se repetă operaţiile de calcul ale ariei şi ale celor două sume **X$_C$**, respectiv **Y$_C$**, utilizând relaţiile de mai sus, pentru valori ale lui **i** de la **1** la **n**;
- se calculează valorile celor două coordonate **X$_C$**, respectiv **Y$_C$**;
- se afişează valorile obţinute.

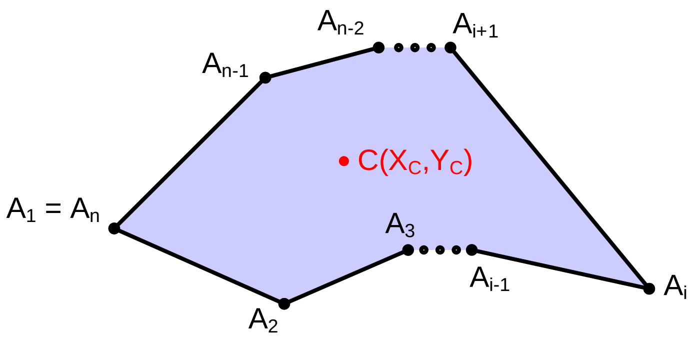
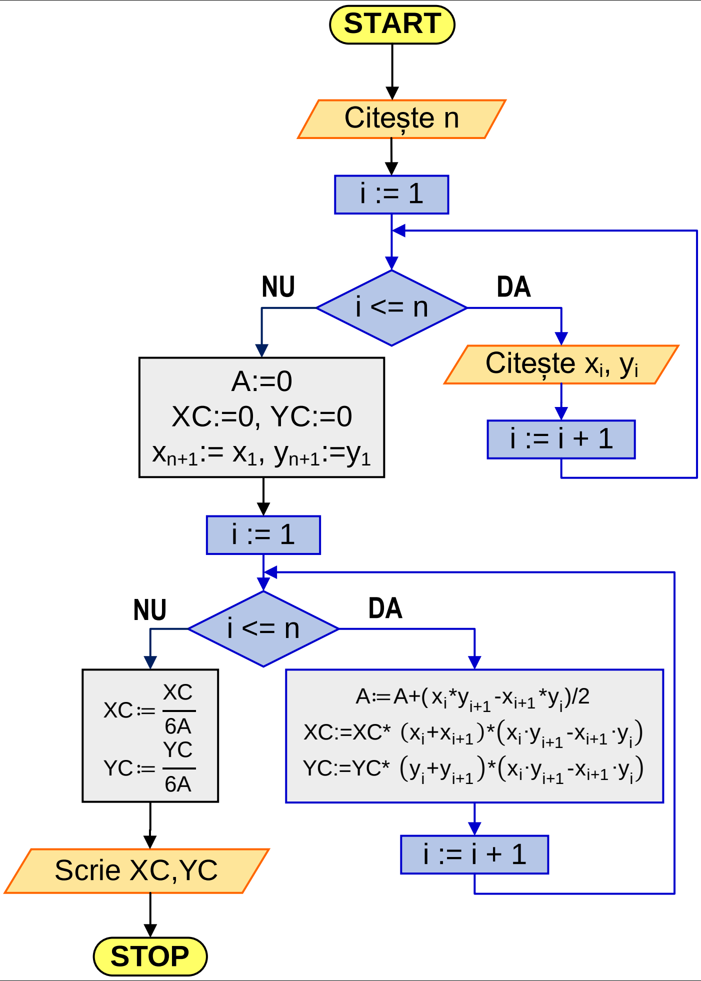


Figura 8.5a. Centrul de masă al unui poligon neregulat

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.5b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.5c.

Schema logică



Limbajul pseudocod

Început
  Citeşte n
  Pentru i = 1, n
    Citeşte x[i], y[i]
  Sfârşit pentru
    A ← 0
    XC ← 0, YC ← 0
    x[n+1] ← x[1], y[n+1] ← y[1]
  Pentru i = 1, n
    A ← A + (x[i] * y[i+1] - x[i+1] * y[i]) / 2.;
    XC ← XC + (x[i] + x[i+1]) * (x[i] * y[i+1] - x[i+1] * y[i])
    YC ← YC + (y[i] + y[i+1]) * (x[i] * y[i+1] - x[i+1] * y[i])
  Sfârşit pentru
    XC ← XC / (6 * A)
    YC ← YC / (6 * A)
  Scrie XC, YC
Sfârşit

Figura 8.5b. Reprezentarea algoritmului pentru calculul **coordonatelor centrului de masă** al unui poligon neregulat

Programul C	Rularea programului
#include<stdio.h>	Introduceti numarul de varfuri, n = **8**
#include<math.h>	x[1] = **0**
int main(void)	y[1] = **0**
{ int n, i; float x[20], y[20], A, XC, YC;	x[2] = **5**
printf("\n Introduceti numarul de varfuri, n = ");	y[2] = **0**
scanf("%d",&n);	x[3] = **5**
for(i = 1 ; i <= n ; i++)	y[3] = **1**
{ printf("\n x[%d] = ",i); scanf("%f",&x[i]);	x[4] = **2**
printf("\n y[%d] = ",i); scanf("%f",&y[i]); }	y[4] = **1**
A = 0; XC = 0; YC = 0; x[n+1] = x[1]; y[n+1] = y[1];	x[5] = **2**
for(i = 1 ; i <= n ; i++)	y[5] = **5**
{ A = A + (x[i]*y[i+1]-x[i+1]*y[i])/2.;	x[6] = **5**
XC = XC + (x[i]+x[i+1])*(x[i]*y[i+1]-x[i+1]*y[i]);	y[6] = **5**
YC = YC + (y[i]+y[i+1])*(x[i]*y[i+1]-x[i+1]*y[i]); }	x[7] = **5**
XC = XC / 6 / A; YC = YC / 6 / A;	y[7] = **6**
printf("\n Coordonata X este: XC = %f ",XC);	x[8] = **0**
printf("\n Coordonata Y este: YC = %f",YC);	y[8] = **6**
}	Aria A este: A = 18.000000
	Coordonata X este: XC = 1.833333
	Coordonata Y este: YC = 3.000000

Figura 8.5c. Programul C şi rularea acestuia pentru calculul **coordonatelor centrului de masă** al unui poligon neregulat

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

În figura 8.5d este ilustrată determinarea coordonatelor centrului de masă utilizând platforma de modelare tridimensională www.onshape.com.

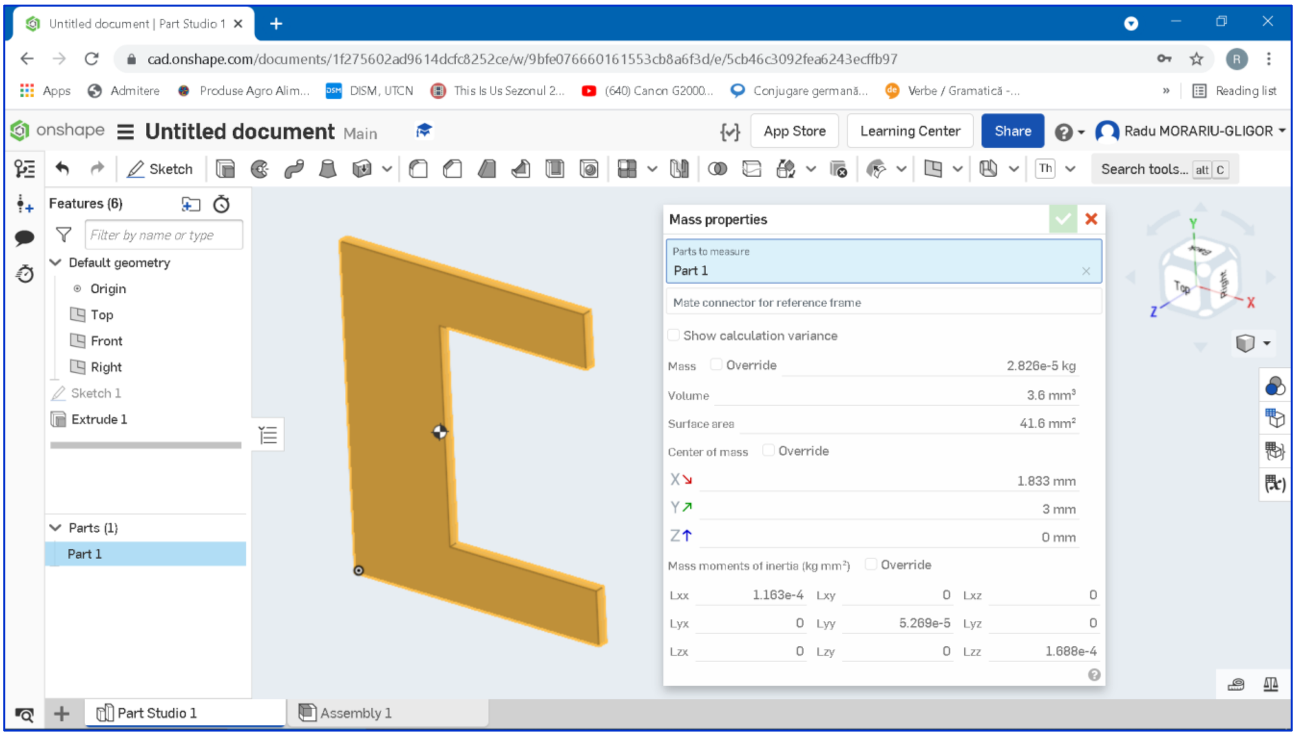


Figura 8.5d. Determinarea coordonatelor centrului de masă pentru un poligon neregulat, utilizând platforma www.onshape.com

8.6. Determinarea perioadei oscilaţiilor libere ale unui pendul matematic

Se doreşte să se determine perioadei oscilațiilor libere ale unui pendul matematic având lungimea firului inextensibil **ℓ** și amplitudinea unghiulară a oscilațiilor **α** (figura 8.6a).

În cazul pendulului matematic, perioada oscilațiilor se exprimă cu relația:

$$T = 4\sqrt{\frac{\ell}{g}}\int_0^{\pi/2}\frac{d\varphi}{\sqrt{1-\sin^2\frac{\alpha}{2}\sin^2\varphi}} \tag{8.14}$$

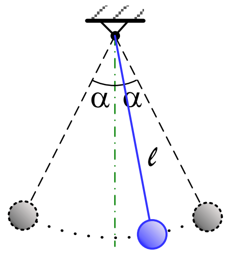


Figura 8.6a. Pendul matematic

unde: **g** reprezintă accelerația gravitațională.

În urma dezvoltării în serie a expresiei de sub semnul integralei și a integrării termen cu termen, perioada **T** se poate scrie sub forma:

$$T = 2\pi\sqrt{\frac{\ell}{g}}\cdot\left\{1+\sum_{i=1}^{\infty}\left[\frac{1\cdot3\cdot5\cdot\ldots\cdot(2i-1)}{2\cdot4\cdot6\cdot\ldots\cdot(2i)}\right]^2\cdot\sin^{2i}\cdot\frac{\alpha}{2}\right\} \tag{8.15}$$

Pentru calculul perioadei **T** trebuie să se calculeze următoarea sumă, cu **n** termeni:

$$S = \left(\frac{1}{2}\right)^2\cdot c^2+\left(\frac{1\cdot3}{2\cdot4}\right)^2\cdot c^4+\left(\frac{1\cdot3\cdot5}{2\cdot4\cdot6}\right)^2\cdot c^6+\cdots+\left(\frac{1\cdot3\cdot5\cdot\ldots\cdot(2i+1)}{2\cdot4\cdot6\cdot\ldots\cdot(2i)}\right)^2\cdot c^{2i}+\cdots \tag{8.16}$$

unde s-a notat $c = \sin\frac{\alpha}{2}$.

Dacă se notează $\boldsymbol{t_i}$ termenul de rang **i** al seriei, pentru calculul acestuia se utilizează următoarea relație de recurență:

$$t_i = c^2\cdot t_{i-1}\cdot\left(\frac{2i-1}{2i}\right)^2 \tag{8.17}$$

Pentru a calcula suma seriei, se inițializează primul termen cu valoarea sa, adică $t_1 = \left(\frac{1}{2}\right)^2\cdot c^2$, se consideră inițial că $S = t_1$, și apoi se aplică relația de calcul $S = S + t_i$ pentru toate valorile lui **i** de la **2** până la **n**.

Perioada **T** se va calcula cu relația:

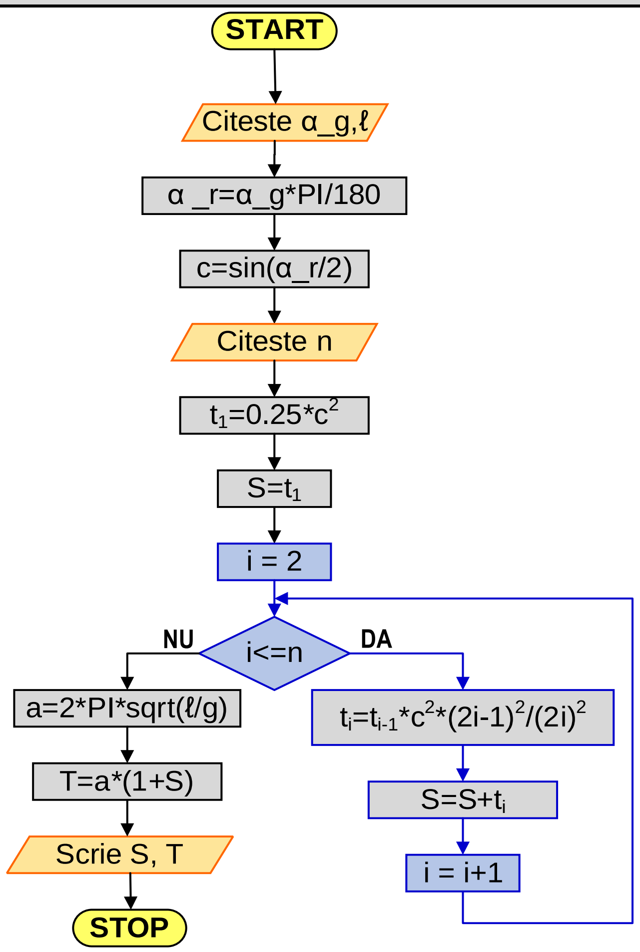
$$T = a\cdot(1+S) \tag{8.18}$$

unde: $a = 2\pi\sqrt{\frac{\ell}{g}}$.

Algoritmul pentru rezolvarea problemei presupune parcurgerea următorilor paşi:
- se citesc valorile amplitudinii unghiulare ale oscilațiilor **α**, în grade, precum și lungimea firului pendulului matematic;
- se transformă valoarea lui **α**, din grade în radiani, cu formula: $\alpha = \frac{\alpha^\circ\cdot3.14159}{180.0}$;
- se calculează variabila **c**, cu relația $c = \sin\frac{\alpha}{2}$ și se alege **n**, numărul de termeni pentru care se calculează suma;
- se inițializează primul termen din serie, $t_1 = \left(\frac{1}{2}\right)^2\cdot c^2$ și suma, $S = t_1$;
- în mod repetat, cu condiția ca poziția elementului din sumă să fie mai mică sau egală cu numărul maxim de elemente (**i ≤ n**), se calculează ceilalți **n** termeni ai seriei precum și suma lor. Tot în cadrul aceleiași iterații se afișează fiecare element al seriei după ce s-a calculat;
- după calculul sumei celor **n** termeni, se modifică valoarea variabilei a cu ajutorul relației $a = 2\pi\sqrt{\frac{\ell}{g}}$ și se calculează perioada oscilațiilor pendulului matematic cu formula: $T = a\cdot(1+S)$.
- se afișează valoarea perioadei **T**.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.6b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.6c.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte alfa_g, l
  alfa_r ← alfa_g * PI / 180
  c ← sin(alfa_r / 2)
  Citeşte n
  t[1] ← 0,25 * c^2
  S ← t[1]
  Pentru i = 2, n
     t[i] ← t[i-1] * c^2 * (2 * i - 1)^2 / (2 * i)^2
     S ← S + t[i]
  Sfârşit pentru
  a ← 2 * PI * sqrt(l / g)
  T ← a * (1 + S)
  Scrie T
Sfârşit
```

Figura 8.6b. Reprezentarea algoritmului pentru determinarea perioadei oscilaţiilor libere ale unui pendul matematic

Programul C	Rularea programului

```c
#include<stdio.h>
#include<math.h>
#define G 9.81
int main(void)
{ int n, alfa_g, l, i; float a, S, c, T, t[30], alfa_r;
printf( "Introduceti unghiul [grade], alfa_g = " );
scanf( "%d",&alfa_g );
printf( "Introduceti lungimea firului, l = " );
scanf( "%d",&l );
alfa_r = (alfa_g*M_PI)/180; c = sin(alfa_r/2);
printf( "Introduceti n = " ); scanf( "%d",&n );
t[1] = 0.25*c*c; S = t[1];
for( i = 2 ; i <= n ; i++ )
{  t[i]=t[i-1]*c*c*((2.*i-1)/(2*i))*((2.*i-1)/(2*i));  S=S+t[i];  }
a=2*M_PI*sqrt(l/G);  T=a*(1+S);
printf( "\n Perioada pendulului este: %6.3f",T );
}
```

Cazul 1:

Introduceti unghiul [grade], alfa_g = **30**

Introduceti lungimea firului, l = **4**

Introduceti n = **4**

Perioada pendulului este: 4.079

Cazul 2:

Introduceti unghiul [grade], alfa_g = **60**

Introduceti lungimea firului, l = **10**

Introduceti n = **10**

Perioada pendulului este: 6.804

Figura 8.6c. Programul C şi rularea acestuia pentru determinarea perioadei oscilaţiilor libere ale unui pendul matematic

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.7. Reducerea unui sistem de forţe coplanare

Se consideră o placă asupra căreia acționează un sistem de forțe coplanare situate în planul plăcii, în acest caz planul **xOy** (figura 8.7a). Se doreşte să se determine elementele torsorului de reducere în raport cu punctul **O** – originea sistemului de coordonate.

Torsorul de reducere se compune din vectorul **R** – rezultanta forţelor care acţionează asupra plăcii, respectiv vectorul **M** – momentul rezultant, adică suma momentelor fiecărei forțe care acţionează asupra plăcii în raport cu punctul **O** – originea sistemului de coordonate.

În acest caz, deoarece: $z_i = 0, F_{iz} = 0$ relațiile de calcul ale componentelor rezultantei, respectiv momentului rezultant sunt:

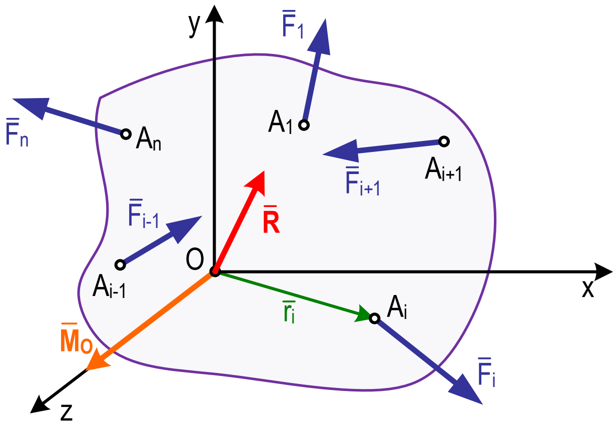


Figura 8.7a. Sistem de forțe coplanare

$$R_x = \sum_{i=1}^{n} F_{ix} = \sum_{i=1}^{n} F_i \cdot cos(\alpha_i) \tag{8.19}$$

$$R_y = \sum_{i=1}^{n} F_{iy} = \sum_{i=1}^{n} F_i \cdot sin(\alpha_i) \tag{8.20}$$

Modulul rezultantei se obține pe baza relației:

$$R = \sqrt{R_x^2 + R_y^2}. \tag{8.21}$$

Componenta momentului rezultant pe axa Oz, se calculează astfel:

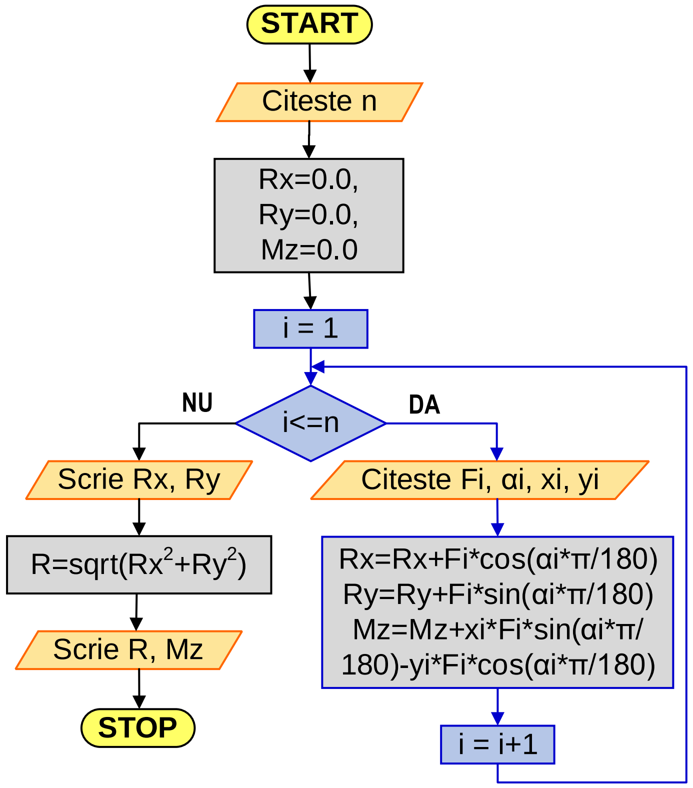
$$M_z = \sum_{i=1}^{n} \big(x_i \cdot F_i \cdot sin(\alpha_i) - y_i \cdot F_i \cdot cos(\alpha_i)\big) \tag{8.22}$$

Pentru calculul elementelor torsorului de reducere sunt necesare numărul și valorile modulelor forțelor care acționează asupra plăcii plane, direcția forțelor (unghiurile **α**), precum și coordonatele punctelor în care acționează aceste forțe asupra plăcii.

Algoritm pentru rezolvarea problemei presupune parcurgerea următoarelor etape:

- citirea valorii variabilei **n**, numărul de forţe;
- deoarece pentru calculul variabilelor $\mathbf{R_x}$, $\mathbf{R_y}$, respectiv $\mathbf{M_z}$ se utilizează sume, acestea se iniţializează cu **0**;
- utilizând un ciclu cu contor se citesc valorile modulelor forțelor, unghiurile care indică direcția lor (în grade), precum și coordonatele **x** şi **y** ale punctelor unde acestea acționează și se calculează componentele vectorilor rezultanți $\mathbf{R_x}$ și $\mathbf{R_y}$ cu relaţiile (8.19) şi (8.20), respectiv componenta momentului rezultant pe axa **Oz** notat cu $\mathbf{M_z}$ cu ajutorul relației (8.22);
- se calculează modulul rezultantei, pe baza componentelor acesteia, utilizând relaţia (8.21);
- se afișează valorile obţinute ale variabilelor: $\mathbf{R_x}$, $\mathbf{R_y}$, **R** și $\mathbf{M_z}$..

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.7b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.7c.

Schema logică	Limbajul pseudocod
	see below


```
Început
  Citeşte n
  Rx ← 0, Ry ← 0, Mz ← 0,
  Pentru i = 1, n
    Citeşte F[i], alfa[i], x[i], y[i]
    Rx ← Rx + F[i] * cos(alfa[i]*PI/180)
    Ry ← Ry + F[i] * sin(alfa[i]*PI/180)
    Mz ← Mz + x[i]*F[i]*sin(alfa[i]*PI/180)
    Mz ← Mz - y[i]*F[i]*cos(alfa[i]*PI/180)
  Sfârşit pentru
  Scrie Rx, Ry
  R ← sqrt(Rx² + Ry²)
  Scrie R, Mz
Sfârşit
```

Figura 8.7b. Reprezentarea algoritmului pentru reducerea unui sistem de forţe coplanare

Programul C

```
#include<stdio.h>
#include<math.h>
int main(void)
{ int n, i;
  float Rx=0,Ry=0,R, Mz=0, F[10], alfa[10], x[10], y[10];
  printf( "\n Introduceti numarul fortelor n = " );
  scanf( "%d",&n );
  for( i = 1 ; i <= n ; i++ )
   { printf( "\n F[%d] = ",i );  scanf( "%f",&F[i] );
     printf( "alfa[%d] = ",i );  scanf( "%f",&alfa[i] );
     printf( "x[%d] = ",i );  scanf( "%f",&x[i] );
     printf( "y[%d] = ",i );  scanf( "%f",&y[i] );
     Rx = Rx + F[i] * cos(alfa[i] * M_PI / 180.);
     Ry = Ry + F[i] * sin(alfa[i] * M_PI / 180.);
     Mz = Mz + x[i] * F[i] * sin(alfa[i] * M_PI / 180.);
     Mz = Mz - y[i] * F[i] * cos(alfa[i] * M_PI / 180.);  }
  printf( "\n Rx=%6.3f",Rx );  printf( "\n Ry=%6.3f",Ry );
  R = sqrt(Rx*Rx+Ry*Ry);
  printf( "\n Modulul rezultantei: R = %6.3f",R );
  printf( "\n Momentului pe axa Oz: Mz = %6.3f ",Mz );
}
```

Rularea programului

Introduceti numarul fortelor n = **3**
F[1] = **55**
alfa[1] = **315**
x[1] = **50**
y[1] = **0**
F[2] = **50**
alfa[2] = **15**
x[2] = **0**
y[2] = **0**
F[3] = **60**
alfa[3] = **45**
x[3] = **100**
y[3] = **60**
Rx = 129.614
Ry = 16.476
Modulul rezultantei: R = 130.657
Momentului pe axa Oz: Mz=-247.487

Figura 8.7c. Programul C şi rularea acestuia pentru reducerea unui sistem de forţe coplanare

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.8. Mişcarea în vid a punctului material greu

Se consideră un punct material **M**, de masă **m**, lansat din punctul fix **O** în plan vertical cu viteza inițială $\mathbf{v_0}$, înclinată cu unghiul **α** faţă de orizontală. Asupra punctului acționează greutatea sa $\mathbf{\bar{G}} = \mathbf{m} \cdot \mathbf{\bar{g}}$. Pornind de la ecuațiile diferențiale de mișcare în planul **xOy** (figura 8.8a):

$$\begin{cases} m \cdot \ddot{x} = 0 \\ m \cdot \ddot{y} = -m \cdot g \end{cases} \tag{8.23}$$

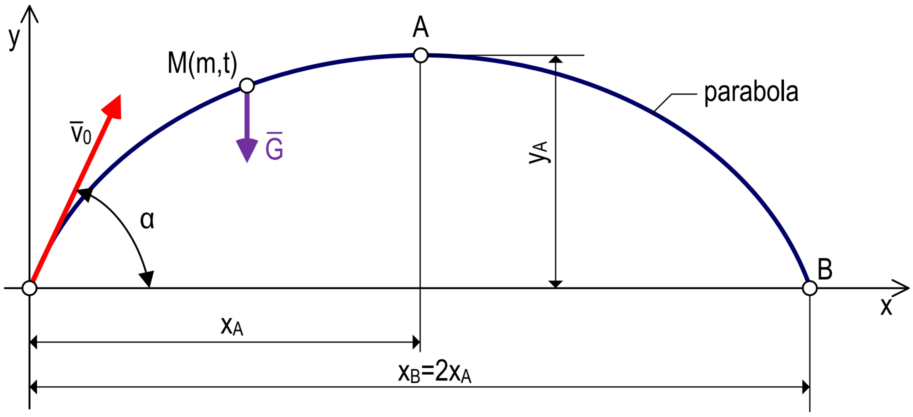


Figura 8.8a. Mişcarea în vid a punctului material greu

prin integrare succesivă se obțin soluțiile generale:

$$\begin{cases} \dot{x} = C_1 \\ x = C_1 \cdot t + C_2 \end{cases}; \quad \begin{cases} \dot{y} = -g \cdot t + C_3 \\ y = -g \cdot \frac{t^2}{2} + C_3 \cdot t + C_4 \end{cases} \tag{8.24}$$

Ţinând cont de condiţiile iniţiale în ceea ce priveşte poziţia şi viteza la momentul **t = 0**, adică:

$$\begin{cases} x = 0 \\ y = 0 \end{cases}; \quad \begin{cases} \dot{x} = v_0 \cdot cos(\alpha) \\ \dot{y} = v_0 \cdot sin(\alpha) \end{cases} \tag{8.25}$$

se determină constantele de integrare: $\begin{cases} C_1 = v_0 \cdot cos(\alpha) \\ C_2 = 0 \end{cases}; \quad \begin{cases} C_3 = v_0 \cdot sin(\alpha) \\ C_4 = 0 \end{cases}$

Astfel, *ecuaţiile parametrice de mişcare* a punctului material greu în vid sunt:

$$\begin{cases} x = v_0 \cdot t \cdot cos(\alpha) \\ y = -g \cdot \frac{t^2}{2} + v_0 \cdot t \cdot sin(\alpha) \end{cases} \tag{8.26}$$

Traiectoria de mişcare a punctului se obţine eliminând timpul din cele două ecuaţii, astfel rezultă :

$$y = -\frac{g}{2 \cdot v_0^2 \cdot cos^2(\alpha)} \cdot x^2 + x \cdot tg(\alpha) \tag{8.27}$$

Relaţia (8.27) reprezintă ecuaţia unei **parabole** având ca axă de simetrie verticala ce trece prin punctul **A** de înălţime maximă.

Coordonatele acestui punct se determină din condiţia ca $\frac{dy}{dx} = 0$ adică:

$$\begin{cases} x_A = \frac{v_0^2 \cdot sin(2\alpha)}{2 \cdot g} \\ y_A = \frac{v_0^2 \cdot sin^2(\alpha)}{2 \cdot g} \end{cases} \tag{8.28}$$

În acest punct, tangenta la traiectorie este orizontală, astfel că viteza nu va avea proiecţie pe axa verticală:

$$v_{Ay} = -g t_A + v_0 \cdot sin\alpha = 0 \tag{8.29}$$

de unde rezultă expresia timpului de urcare: $$t_A = \frac{v_0 \cdot sin\alpha}{g} \tag{8.30}$$

Distanţa $\mathbf{OB} = \mathbf{x_B} = \mathbf{2} \cdot \mathbf{x_A}$ poartă numele de "bătaie".

În programul următor se vor introduce ca date de intrare viteza iniţială **v_0** şi unghiul de lansare **α** determinându-se pe baza relaţiilor prezentate anterior înălţimea maximă la care ajunge punctul **y_A**, distanţa la care punctul atinge din nou suprafaţa orizontală (bătaia), adică **$2*x_A$** şi timpul [sec] cât punctul s-a aflat în aer, de la momentul lansării, adică **$2*t_A$**.

De asemenea, după determinarea timpului în care punctul material s-a aflat în aer, se determină poziţia punctului în funcţie de timp utilizând un interval de timp precizat de utilizator.

Algoritmul pentru rezolvarea acestei probleme necesită parcurgerea următorilor paşi :

- citirea de la tastatură a datelor de intrare, adică a vitezei iniţiale **v_0** [m/s] şi unghiul de lansare **α** [grade] ;
- se calculează înălţimea maximă, bătaia şi timpul în care punctul material se află în aer, utilizând relaţiile prezentate anterior ;
- se citeşte de la tastatură pasul utilizat pentru calculul poziţiei punctului material de la momentul lansării până la momentul atingerii suprafeţei orizontale (notat cu **pas_t**) ;
- cu ajutorul unui ciclu cu contor se calculează şi se afişează poziţia punctului material în fiecare moment, de la momentul lansării până la momentul atingerii suprafeţei orizontale.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.8b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.8c.

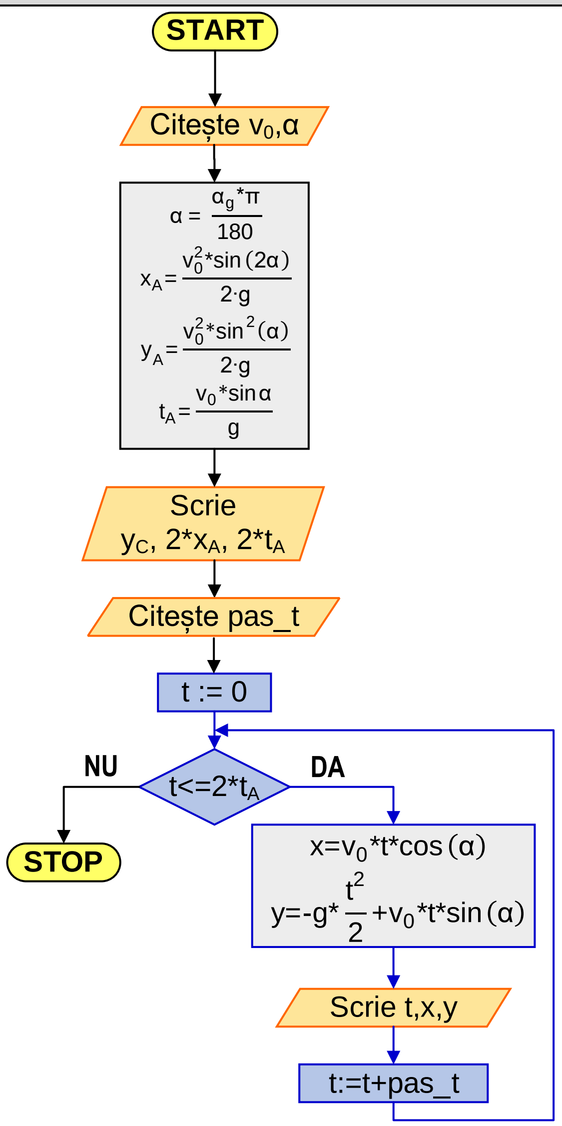
Schema logică	Limbajul pseudocod
	**Început** Citeşte v0, alfag alfar ← alfag*PI/180.0 xA ← v0*v0*sin(2*alfar)/(2*G) yA ← v0*v0*sin(alfar)*sin(alfar)/(2*G) tA ← v0*sin(alfar)/G Scrie yA, 2*xA, 2*tA Citeşte pas_t **Pentru** t = 0, 2 * tA **pas** pas_t x ← v0 * t * cos(alfar) y ← v0 * t * sin(alfar) – G * t * t / 2 Scrie t, x, y **Sfârşit pentru** **Sfârşit**

Figura 8.8b. Reprezentarea algoritmului pentru studiul mişcării în vid a punctului material greu

Programul C	Rularea programului

Programul C

```c
#include<stdio.h>
#include<math.h>
#define G 9.81
int main(void)
{  float v0,alfag,alfar,xA,yA,tA,x,y,t,pas_t;
   printf( "\n Viteza initiala [m/s]: v0 = " );
   scanf( "%f",&v0 );
   printf( " Unghiul de lansare [grade]: alfa = " );
   scanf( "%f",&alfag );
   alfar = alfag * M_PI/180.0;
   xA = v0*v0*sin(2*alfar)/(2*G);
   yA = v0*v0*pow(sin(alfar),2)/(2*G);
   tA = v0*sin(alfar)/G;
   printf( "\n Inaltimea maxima = %6.3f [m]",yA );
   printf( "\n Bataia = %6.3f [m]",2*xA );
   printf( "\n Timpul total = %6.3f [s]",2*tA );
   printf( "\n \n Pasul interv. de timp [s]: pas_t = " );
   scanf( "%f",&pas_t );
   for( t = 0 ; t <= 2*tA ; t = t+pas_t )
    {  x = v0*t*cos(alfar); y = v0*t*sin(alfar)-G*t*t/2;
       printf( "\n t = %6.3f \t x = %6.3f \t y = %6.3f",t,x,y );
    }
}
```

Rularea programului

```
Viteza initiala [m/s]: v0 = 15
Unghiul de lansare [grade]]: alfa = 42

 Inaltimea maxima =  5.135 [m]
 Bataia = 22.810 [m]
 Timpul total =  2.046 [s]

 Pasul interv. de timp [s]: pas_t = 0.25

 t =  0.000     x =  0.000     y =  0.000
 t =  0.250     x =  2.787     y =  2.203
 t =  0.500     x =  5.574     y =  3.792
 t =  0.750     x =  8.360     y =  4.769
 t =  1.000     x = 11.147     y =  5.132
 t =  1.250     x = 13.934     y =  4.882
 t =  1.500     x = 16.721     y =  4.019
 t =  1.750     x = 19.508     y =  2.543
 t =  2.000     x = 22.294     y =  0.454
```

Figura 8.8c. Programul C şi rularea acestuia pentru studiul mişcării în vid a punctului material greu

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

O problemă de interes practic este aceea de a determina unghiul de lansare **α** astfel încât, pentru o viteză iniţială de modul cunoscut $\mathbf{v_0}$, să fie atinsă o ţintă **P(x_P, y_P)** situată în planul vertical **xOy**. Pentru ca acest lucru să se întâmple, coordonatele punctului ţintă trebuie să verifice ecuaţia traiectoriei:

$$y_P = -\frac{g}{2\cdot v_0^2\cdot cos^2(\alpha)}\cdot x_P^2 + x_P\cdot tg(\alpha) = -\frac{g}{2\cdot v_0^2}\cdot x_P^2\cdot\left(1+tg^2(\alpha)\right) + x_P\cdot tg(\alpha) \qquad (8.31)$$

După ordonare în funcţie de **tg(α)** rezultă : $tg^2(\alpha) - \frac{2\cdot v_0^2}{g\cdot x_P}\cdot tg(\alpha) + \frac{2\cdot v_0^2}{g\cdot x_P^2}\cdot y_P + 1 = 0$ (8.32)

Soluţiile acestei ecuaţii sunt:

$$tg(\alpha) = \frac{1}{g\cdot x_P}\cdot\left[v_0^2 \pm \sqrt{v_0^4 - (2\cdot v_0^2\cdot g\cdot y_P + g^2\cdot x_P^2)}\right] \qquad (8.33)$$

Se constată astfel că, în general, există două soluţii pentru **tgα**, deci sunt două unghiuri sub care poate fi lansat punctul material greu pentru a atinge ţinta **P** (figura 8.8d):

- **α_1** - corespunzător ***traiectoriei razante*** (Γ_1);
- **α_2** - corespunzător ***traiectoriei boltite*** (Γ_2) ;

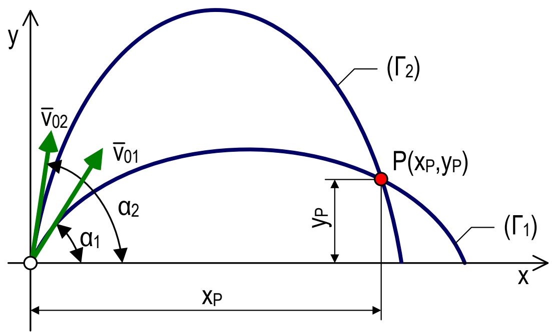
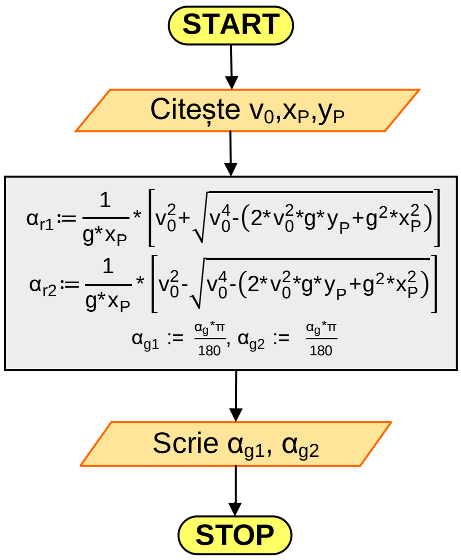


Figura 8.8d

Algoritmul de calcul al unghiurilor de lansare a punctului material presupune parcurgerea următoarelor etape:
- citirea de la tastatură a vitezei iniţiale $\mathbf{v_0}$;
- citirea de la tastatură a coordonatelor punctului ţintă: $\mathbf{x_P}$, respectiv $\mathbf{y_P}$;
- calculul şi afişarea celor două valori ale unghiului de lansare $\boldsymbol{\alpha}$ pentru care punctul ţintă este atins ;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.8e, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.8f.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte v0, xP, yP
   alfar1 ← arctg((v0*v0+sqrt(v0*v0*v0*v0-(2*v0*v0*G*yP+G*G*xP*xP)))/(G*xP))
   alfar2 ← arctg((v0*v0-sqrt(v0*v0*v0*v0-(2*v0*v0*G*yP+G*G*xP*xP)))/(G*xP))
   alfag1 ← alfar1*180/PI
   alfag2 ← alfar2*180/PI
   Scrie alfag1, alfag2
Sfârşit
```

Figura 8.8e. Reprezentarea algoritmului pentru determinarea unghiului de lansare $\boldsymbol{\alpha}$

Programul C şi rularea acestuia

```c
#include<stdio.h>
#include<math.h>
#define G 9.81
int main(void)
{  float v0,alfag1,alfag2,alfar1,alfar2,xP,yP;
   printf( "\n Viteza initiala [m/s]: v0 = " ); scanf( "%f",&v0 );
   printf( " Coordonata x a tintei [m]: xP = " ); scanf( "%f",&xP );
   printf( " Coordonata y a tintei [m]: yP = " ); scanf( "%f",&yP );
   alfar1 = atan((v0*v0+sqrt(v0*v0*v0*v0 - (2*v0*v0*G*yP+G*G*xP*xP))) / (G*xP));
   alfar2 = atan((v0*v0-sqrt(v0*v0*v0*v0 - (2*v0*v0*G*yP+G*G*xP*xP))) / (G*xP));
   alfag1 = alfar1 * 180.0/M_PI;    alfag2 = alfar2 * 180.0/M_PI;
   printf( "\n Unghiul de lansare alfa 1 = %6.3f [grade]",alfag1 );
   printf( "\n Unghiul de lansare alfa 2 = %6.3f [grade]",alfag2 );
}
```

Rularea programului:

Viteza initiala [m/s]: v0 = **20**
Coordonata x a tintei [m]: xP = **20**
Coordonata y a tintei [m]: yP = **8**

Unghiul de lansare alfa 1 = 73.015 [grade]
Unghiul de lansare alfa 2 = 38.787 [grade]

Figura 8.8f. Programul C şi rularea acestuia pentru determinarea unghiului de lansare **α**

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.9. Studiul mişcării cu frecare a unui corp pe planul înclinat

Se consideră un corp de masă **m**, aflat pe un plan înclinat faţă de orizontală la unghiul **α** (figura 8.9a). Asupra corpului acţionează forţa de greutate **G**, precum şi o forţă de tracţiune **F** sub un unghi oarecare **β** faţă de orizontală, coeficientul de frecare dintre corp şi planul înclinat fiind **μ**. De asemenea, se consideră coeficientul de frecare statică egal cu cel de frecare dinamică iar corpul se deplasează pe planul înclinat cu viteză constantă sau stă în repaus.

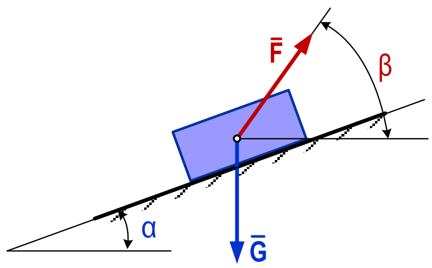


Figura 8.9a

Pe baza datelor de intrare (masa corpului **m**, valoarea forţei de tracţiune **F**, unghiurile **α** şi **β**, respectiv coeficientul de frecare dintre corp şi planul înclinat **μ** se doreşte să se determine felul în care se deplasează corpul pe planul înclinat (în jos pe planul înclinat, în repaus faţă de planul înclinat sau se deplasează în sus de-a lungul planului înclinat).

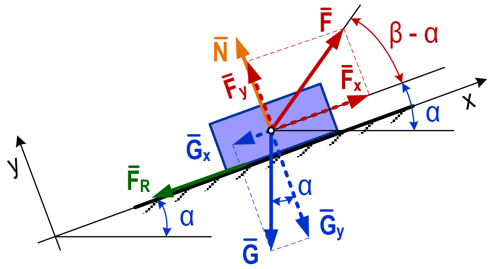


Figura 8.9b

În figura 8.9b este prezentat cazul în care corpul se deplasează în sus pe planul înclinat, forţa de frecare F_R acţionând în acest caz în jos, de-a lungul planului înclinat.

În figura 8.9c este prezentat cazul în care corpul se deplasează în jos pe planul înclinat, situaţie în care forţa de frecare F_R acţionează în sus, de-a lungul planului înclinat.

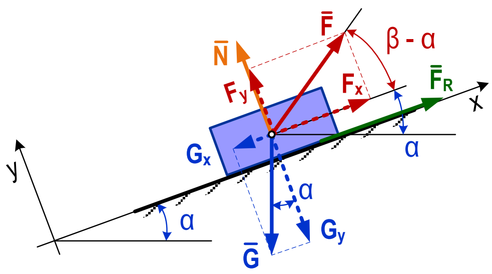


Figura 8.9c

În primul caz (figura 8.9.b), ecuaţiile de echilibru sunt:

$$\begin{cases} \sum F_x = 0: \quad F_x - G_x - F_R = 0 \\ \sum F_y = 0: \quad N + F_y - G_y = 0 \\ F_R \leq \mu \cdot N \end{cases} \tag{8.34}$$

$$N = G_y - F_y = G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha) \tag{8.35}$$

$$F \cdot \cos(\beta - \alpha) > G \cdot \sin\alpha + \mu \cdot (G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha)) \tag{8.36}$$

$$F \cdot \cos(\beta - \alpha) + \mu \cdot F \cdot \sin(\beta - \alpha) > G \cdot \sin\alpha + \mu \cdot G \cdot \cos\alpha \tag{8.37}$$

$$F > G \cdot \frac{\sin\alpha + \mu \cdot \cos\alpha}{\cos(\beta - \alpha) + \mu \cdot \sin(\beta - \alpha)} \tag{8.38}$$

În al doilea caz (figura 8.9.c), ecuaţiile de echilibru sunt:

$$\begin{cases} \sum F_x = 0: \quad F_x - G_x + F_R = 0 \\ \sum F_y = 0: \quad N + F_y - G_y = 0 \\ F_R = \mu \cdot N \end{cases} \tag{8.39}$$

$$N = G_y - F_y = G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha) \tag{8.40}$$

$$F \cdot \cos(\beta - \alpha) + \mu \cdot (G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha)) < G \cdot \sin\alpha \qquad (8.41)$$

$$F \cdot \cos(\beta - \alpha) - \mu \cdot F \cdot \sin(\beta - \alpha) < G \cdot \sin\alpha - \mu \cdot G \cdot \cos\alpha \qquad (8.42)$$

$$F < G \cdot \frac{\sin\alpha - \mu \cdot \cos\alpha}{\cos(\beta - \alpha) - \mu \cdot \sin(\beta - \alpha)} \qquad (8.43)$$

Prin urmare, se pot întâlni trei situaţii:

- corpul se deplasează în sus pe planul înclinat dacă:

$$F > G \cdot \frac{\sin\alpha + \mu \cdot \cos\alpha}{\cos(\beta - \alpha) + \mu \cdot \sin(\beta - \alpha)} \qquad (8.44)$$

- corpul stă în repaus pe planul înclinat dacă:

$$G \cdot \frac{\sin\alpha - \mu \cdot \cos\alpha}{\cos(\beta - \alpha) - \mu \cdot \sin(\beta - \alpha)} \leq F \leq G \cdot \frac{\sin\alpha + \mu \cdot \cos\alpha}{\cos(\beta - \alpha) + \mu \cdot \sin(\beta - \alpha)} \qquad (8.45)$$

- corpul coboară pe planul înclinat dacă:

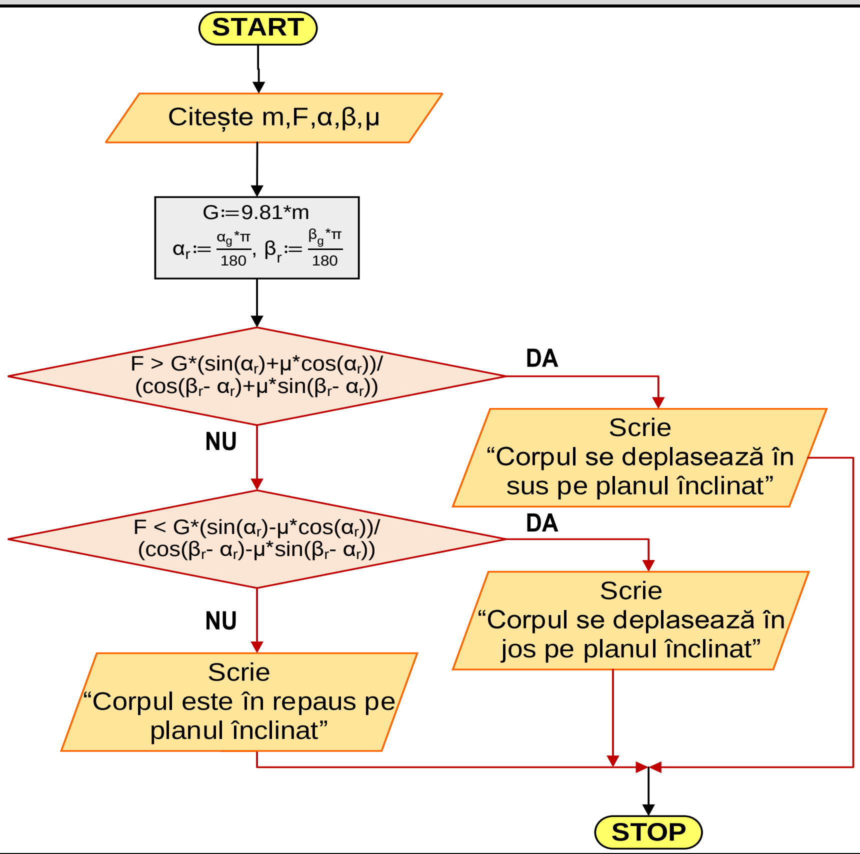
$$F < G \cdot \frac{\sin\alpha - \mu \cdot \cos\alpha}{\cos(\beta - \alpha) - \mu \cdot \sin(\beta - \alpha)} \qquad (8.46)$$

Algoritmul de rezolvare a problemei presupune parcurgerea următorilor paşi:

- se citesc de la tastatură următoarele date de intrare: masa corpului **m**, forţa de tracţiune **F**, unghiul planului înclinat **α**, unghiul sub care acţionează forţa de tracţiune **β**, coeficientul de frecare **μ**;
- se verifică prima condiţie, dată de relaţia (8.44), dacă această condiţie este adevărată se afişează un mesaj prin care se specifică că corpul se deplasează în sus pe planul înclinat;
- dacă condiţia (8.44) nu se îndeplineşte, se verifică condiţia (8.46). Dacă aceasta este adevărată se afişează un mesaj prin care se specifică că corpul se deplasează în jos pe planul înclinat;
- dacă condiţia (8.46) nu este adevărată, înseamnă că corpul stă în repaus pe planul înclinat, pe ecran se afişează un mesaj corespunzător.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.9d, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.9e.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte m, F, α, β, μ
  G ← 9,81*m
  Dacă F > G * ( sin(α) + μ * cos(α) ) / ( cos(β- α) + μ * sin(β- α) ) atunci
     Scrie „ Corpul se deplasează în sus pe planul înclinat!”
  altfel
     Dacă F < G * ( sin(α) - μ * cos(α) ) / ( cos(β- α) - μ * sin(β- α) ) atunci
           Scrie „ Corpul se deplasează în jos pe planul înclinat!”
     altfel
           Scrie „ Corpul este in repaus pe planul inclinat!”
     Sfârşit dacă
  Sfârşit dacă
Sfârşit
```

Figura 8.9d. Reprezentarea algoritmului pentru studiul mişcării unui corp pe planul înclinat

Programul C şi rularea acestuia

```c
#include<stdio.h>
#include<math.h>
int main(void)
{ float m, F, ag, ar, bg, br, u, G;
  printf( "\n Masa corpului [kg], m = " );  scanf( "%f",&m );
  printf( "\n Forta de tractiune [N], F = " );  scanf( "%f",&F );
  printf( "\n Unghiul alfa [grade], alfa = " );  scanf( "%f",&ag );
  printf( "\n Unghiul beta [grade], beta = " );  scanf( "%f",&bg );
  printf( "\n Coeficientul de frecare, miu = " );  scanf( "%f",&u );
  G = 9.81*m; ar = ag * M_PI / 180.0; br = bg * M_PI / 180.0;
  if( F > G*(sin(ar)+u*cos(ar))/(cos(br-ar)+u*sin(br-ar)) )
       printf( "\n Corpul se deplaseaza in sus pe planul înclinat!" );
  else
      if( F < G*(sin(ar)-u*cos(ar))/(cos(br-ar)-u*sin(br-ar)) )
           printf( "\n Corpul se deplaseaza in jos pe planul înclinat!" );
      else
           printf( "\n Corpul este in repaus pe planul inclinat!" );
}
```

Rularea programului:

Cazul 1:

Masa corpului [kg], m = **5**
Forta de tractiune [N], F = **60**
Unghiul alfa [grade], alfa = **30**
Unghiul beta [grade], beta = **45**
Coeficientul de frecare, miu = **0.3**
Corpul se deplaseaza in sus pe planul inclinat!

Cazul 2:

Masa corpului [kg], m = **5**
Forta de tractiune [N], F = **30**
Unghiul alfa [grade], alfa = **30**
Unghiul beta [grade], beta = **45**
Coeficientul de frecare, miu = **0.3**
Corpul este in repaus pe planul inclinat!

Cazul 3:

Masa corpului [kg], m = **5**
Forta de tractiune [N], F = **10**
Unghiul alfa [grade], alfa = **30**
Unghiul beta [grade], beta = **30**
Coeficientul de frecare, miu = **0.3**
Corpul se deplaseaza in jos pe planul inclinat!

Figura 8.9e. Programul C şi rularea acestuia pentru studiul mişcării unui corp pe planul înclinat

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.10. Cinematica mecanismului bielă-manivelă

Mecanismul bielă-manivelă (figura 8.10a) se utilizează pentru transformarea mișcării de translație alternativă în mișcare de rotație continuă (motoare cu ardere internă), respectiv pentru transformarea mișcării de rotație continuă în mișcare de translație alternativă (compresoare, pompe, prese).

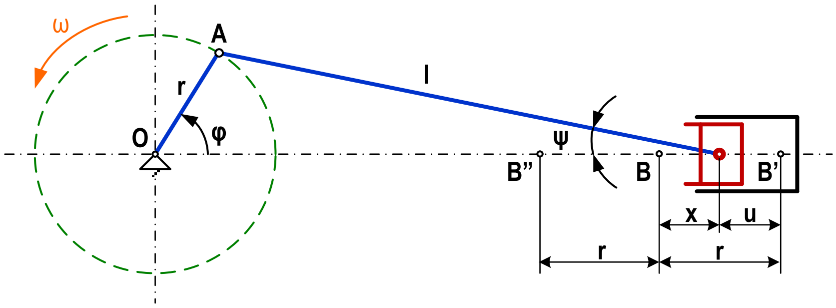


Figura 8.10a. Mecanismul bielă - manivelă

În figură, notaţiile au următoarea semnificaţie: **l** – lungimea bielei, **r** – raza manivelei, **A** – centru de articulaţie dintre bielă şi axa cilindrului, **B'** şi **B"** – poziţiile extreme ale capătului bielei, **x** – deplasarea capătului bielei la un anumit moment. Turaţia manivelei se notează cu „**n**" şi se măsoară în [rot/min].

Manivela se roteşte cu viteza unghiulară constantă: **ω = φ'**, iar cupla de translaţie **B** are cursa B'B"=2r.

Poziţia mecanismului este determinată de unghiul **φ**, care este o funcţie de timp, astfel:

$$\varphi = \omega \cdot t = \frac{\pi \cdot n}{30} \cdot t \tag{8.47}$$

Poziţia pistonului se raportează faţă de punctul mort exterior (**B'**), astfel coordonata punctului **B** faţă de punctul limită **B'** este:

$$u = r \cdot (1 - \cos\varphi) + l(1 - \cos\psi); \tag{8.48}$$

Ţinând seama de relaţia geometrică:

$$\cos\psi = \sqrt{1 - \left(\frac{r}{l}\sin\varphi\right)^2} \tag{8.50}$$

şi dezvoltând în serie de puteri expresia (8.50), şi considerând numai primii doi termeni, se obţine legea de mişcare a pistonului:

$$u \cong r \cdot \left(1 - \cos\varphi + \frac{\lambda}{2}\sin^2\varphi\right) \tag{8.51}$$

În expresia (8.51) au fost neglijaţi termenii cu puteri mai mari decât 2 şi s-a notat $\lambda = \frac{r}{l}$, raport denumit caracteristica structurală a mecansimului.

Pentru determinarea legii de variație a vitezei, se derivează legea de mișcare în raport cu timpul:

$$v = \frac{du}{dt} = \frac{du}{d\varphi} \cdot \frac{d\varphi}{dt} = \omega \cdot \frac{du}{d\varphi}$$

rezultă:

$$\begin{aligned} v(\varphi) &= \omega \cdot r \cdot \left(\sin\varphi + \frac{1}{2} \cdot \lambda \cdot \sin 2\varphi\right), \\ v(t) &= \omega \cdot r \cdot \left(\sin(\omega \cdot t) + \frac{1}{2} \cdot \lambda \cdot \sin(2 \cdot \omega \cdot t)\right) \end{aligned} \tag{8.52}$$

Legea de variație a accelerației se determină derivând viteza în raport cu timpul: $a = \frac{dv}{dt} = \frac{dv}{d\phi} \cdot \frac{d\phi}{dt} = \omega \cdot \frac{dv}{d\phi}$

Rezultă:

$$a(\phi) = \omega^2 \cdot r \cdot (\cos\phi + \lambda \cdot \cos 2\phi)$$
$$a(t) = \omega^2 \cdot r \cdot (\cos(\omega \cdot t) + \lambda \cdot \cos(2 \cdot \omega \cdot t)) \quad (8.53)$$

Studiu cinematic presupune trasarea graficelor deplasării (**x**), a vitezei (**v**), respectiv a acceleraţiei (**a**) în funcție de timp (figura 8.10b).

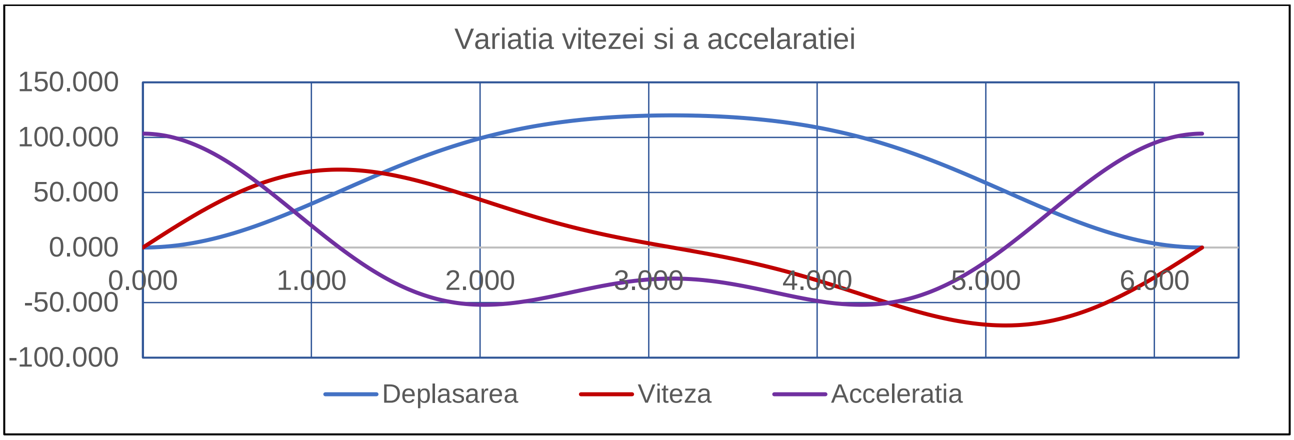


Figura 8.10b. Graficele de variaţie a deplasării, vitezei şi acceleraţiei

În tabelul 8.1 sunt prezentate o parte din valorile unghiului în grade, a unghiului în radiani, a deplasării, vitezei şi acceleraţiei calculate pe baza relaţiilor de mai sus;

Tabelul 8.1

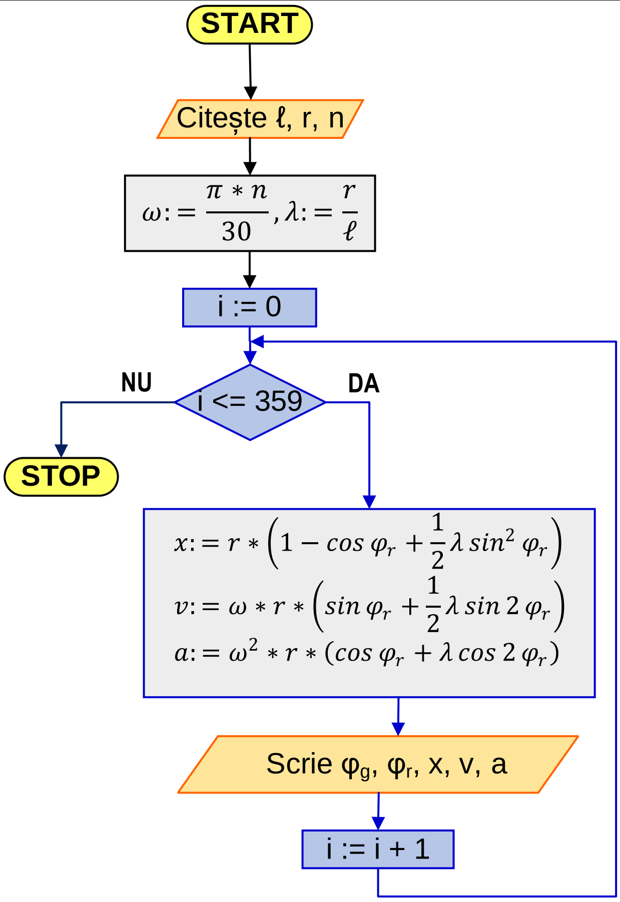
Unghiul în grade φ	Unghiul în radiani φr	Deplasarea x [mm]	Viteza v [mm/s]	Acceleraţia a [mm/s^2]
0	0.000	0.000	0.000	103.396
1	0.017	0.014	1.723	103.363
2	0.035	0.057	3.445	103.264
3	0.052	0.129	5.165	103.100
4	0.070	0.230	6.881	102.870
5	0.087	0.359	8.593	102.574
...	...	...	...	...
355	6.196	0.359	-8.593	102.574
356	6.213	0.230	-6.881	102.870
357	6.231	0.129	-5.165	103.100
358	6.248	0.057	-3.445	103.264
359	6.266	0.014	-1.723	103.363
360	6.283	0.000	0.000	103.396

Algoritmul de calcul pentru valorile deplasării, vitezei şi acceleraţiei presupune parcurgerea următoarelor etape:

- se citesc variabilele de intrare, adică: lungimea bielei – **l** [mm], raza manivelei – **r** [mm], turaţia manivelei – **n** [rot/min];
- cu ajutorul unui ciclu cu contor se dau valori unghiului în grade, în intervalul [0, 360°], cu pasul de **1°**;
- pentru fiecare valoarea a unghiului în grade se calculează unghiul în radiani, valoarea deplasării – **x**, valoarea vitezei – **v**, valoarea acceleraţiei – **a**, pe baza relaţiilor de mai sus şi se afişează aceste valori.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.10c, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.10d.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte l, r, n
  omg ← M_PI * n / 30.0
  lbd ← r / l
  Pentru i = 0, 359
     fig[i] ← i
     fir[i] ← fig[i] * M_PI / 180
     x[i] ← r * ( 1 - cos(fir[i]) + lbd * pow(sin(fir[i]),2) / 2.0)
     v[i] ← omg * r * (sin(fir[i]) + 0.5 * lbd * sin(2 * fir[i])) / 10^3
     a[i] ← omg^2 * r * (cos(fir[i]) + lbd * cos(2 * fir[i])) / 10^6
  Scrie fig[i], fir[i], x, v, a
  Sfârşit pentru
Sfârşit
```

Figura 8.10c. Reprezentarea algoritmului pentru sinteza mecanismului bielă - manivelă

Programul C şi rularea acestuia

```
#include<stdio.h>
#include<math.h>
int main(void)
{  float fig[360], fir[360], x[360], v[360], a[360];
   float l, r, n, omg, lbd;  int i;
   printf( "\n Lungimea bielei [m], l = " ); scanf( "%f",&l );
   printf( "\n Raza manivelei [m], r = " ); scanf( "%f",&r );
   printf( "\n Turatia manivelei [rot/min], n = " ); scanf( "%f",&n );
   omg = M_PI * n / 30.0; lbd = r / l;
   for( i = 0 ; i <= 359 ; i++ )
     {  fig[i] = i; fir[i] = fig[i] * M_PI / 180;
        x[i] = r * (1 - cos(fir[i]) + lbd * pow(sin(fir[i]),2) / 2.0);
        v[i] = omg * r * (sin(fir[i]) + 0.5 * lbd * sin(2 * fir[i])) / 1000;
        a[i] = omg * omg * r * (cos(fir[i]) + lbd * cos(2 * fir[i])) / pow(10.,6);
        printf( "\n %6.3f %6.3f %6.3f %6.3f %6.3f",fig[i],fir[i],x[i],v[i],a[i] );
     }
}
```

Rularea programului:

Lungimea bielei [m], l = **0.2**
Raza manivelei [m], r = **0.075**
Turatia manivelei [rot/min], n = **3000**

```
   0.000000  0.000000  0.000000  0.000000  0.010178
  20.000000  0.349066  0.006168  0.010898  0.009082
  40.000000  0.698132  0.023357  0.019496  0.006152
  60.000000  1.047198  0.048047  0.024231  0.002313
  80.000000  1.396263  0.075615  0.024715 -0.001323
 100.000000  1.745329  0.101662  0.021693 -0.003894
 120.000000  2.094395  0.123047  0.016579 -0.005089
 140.000000  2.443461  0.138264  0.010795 -0.005188
 160.000000  2.792527  0.147122  0.005219 -0.004829
 180.000000  3.141593  0.150000 -0.000000 -0.004626
 200.000000  3.490659  0.147122 -0.005219 -0.004829
 220.000000  3.839724  0.138264 -0.010795 -0.005188
 240.000000  4.188790  0.123047 -0.016579 -0.005089
 260.000000  4.537856  0.101662 -0.021693 -0.003894
 280.000000  4.886922  0.075615 -0.024715 -0.001323
 300.000000  5.235988  0.048047 -0.024231  0.002313
 320.000000  5.585053  0.023357 -0.019496  0.006152
 340.000000  5.934119  0.006168 -0.010898  0.009082
```

Figura 8.10d. Programul C şi rularea acestuia pentru sinteza mecanismului bielă - manivelă

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.11. Calculul de dimensionare al capătului de arbore şi alegerea penei

Se consideră un capăt de arbore supus doar solicitării la torsiune. Relația de calcul al diametrului acestuia este:

$$d_p = \sqrt[3]{\frac{16 \cdot M_t}{\pi \cdot \tau_{at}}} \tag{8.54}$$

unde: $\mathbf{M_t}$ reprezintă momentul de torsiune [Nm], $\boldsymbol{\tau_{at}}$ reprezintă rezistența admisibilă la torsiune (pentru oțeluri obișnuite se recomandă valori mici: τ_{at} = 15 … 45 N/mm², valorile mai mari se recomandă în cazul arborilor scurți, respectiv valorile mai mici în cazul arborilor lungi).

Valoarea obținută aplicând relația (8.34) se adoptă din gama de valori standardizate: 10, 11, 12, 14, 16, 18, 19, 20, 22, 24, 25, 28, 30, 32, 35, 38, 40, 42, 45, 48, 50, 55, 60, 65, 70, 71, 75, 80, 85, 90, conform STAS 8724/2-84.

Lungimea tronsonului de arbore se alege în funcție de diametrul acestuia, conform STAS 8724/2-84 (tabelul 8.2).

Pentru montarea roților de curea, a celor dințate sau a cuplajelor pe tronsoanele arborilor, se vor utiliza pene paralele (figura 8.11a). Dimensiunile penei (**b** x **h**) și ale canalului de pană ($\mathbf{t_1}$ și $\mathbf{t_2}$) se aleg în funcție de diametrul tronsonului de arbore pe care se montează roata sau cuplajul, conform (tabelul 8.3).

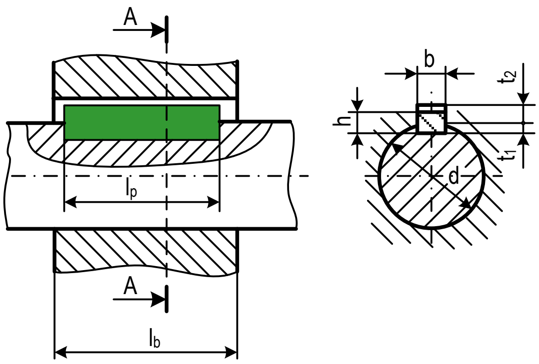


Figura 8.11a

Tabelul 8.2

Valori standardizate ale diametrelor arborilor și a lungimii acestora (serie scurtă)																
Diametrul [mm]	10	11	12	14	16	18	19	20	22	24	25	28	30	32	35	38
Lungimea [mm]	20	20	25	25	28	28	28	36	36	36	42	42	58	58	58	58
Diametrul [mm]	40	42	45	48	50	55	56	60	63	65	70	71	75	80	85	90
Lungimea [mm]	82	82	82	82	82	82	82	105	105	105	105	105	105	130	130	130

Tabelul 8.3

d [mm]		Dimensiunile penei [mm]		Dimensiunile canalului [mm]		Interval de lungimi [mm]
				Adâncimea		
de la	până la	b	h	arbore t_1	butuc t_2	l
10	12	4	4	2,5	1,8	8 … 45
12	17	5	5	3,0	2,3	10 … 56
17	22	6	6	3,5	2,8	14 … 70
22	30	8	7	4,0	3,3	18 … 90
30	38	10	8	5,0	3,3	22 … 110
38	44	12	8	5,0	3,3	28 … 140
44	50	14	9	5,5	3,8	36 … 160
50	58	16	10	6,0	4,3	45 … 180
58	65	18	11	7,0	4,4	50 … 200
65	75	20	12	7,5	4,9	56 … 220
75	85	22	14	9,0	5,4	63 … 250
85	95	25	14	9,0	5,4	70 … 280

Lungimea penei se determină din:

a) condiţia de rezistenţă la strivire, pe baza relaţiei:

$$l_1 \geq \frac{4 \cdot M_t}{d \cdot h \cdot \sigma_{as}} + b \tag{8.55}$$

unde: $\boldsymbol{\sigma_{as}}$ reprezintă rezistenţa admisibilă la strivire, pentru sarcini constante, $\boldsymbol{\sigma_{as}}$ = 100 … 120 [N/mm^2];

b) condiţia de rezistenţă la forfecare, pe baza relaţiei:

$$l_2 \geq \frac{2 \cdot M_t}{d \cdot b \cdot \tau_{af}} + \frac{\pi \cdot b}{4} \tag{8.56}$$

unde: **b** reprezintă lăţimea penei [mm], $\boldsymbol{\tau_{af}}$ reprezintă tensiunea admisibilă la forfecare [N/mm^2] şi are valoarea ≤ 100 N/mm^2;

Lungimile tipizate pentru pene paralele sunt (STAS 1004): 8, 10, 12, 14, 16, 18, 20, 22, 25, 28, 32, 36, 40, 45, 50, 56, 63, 70, 80, 90, 100, 110, 125, 140, 160, 180, 200, 220, 250, 280, 320, 360, 400, 450, 500.

Valoarea maximă rezultată din aplicarea celor două relaţii se va standardiza prin adoptarea unei valori din şirul valorilor standardizate, conform tabelului 8.4.

$$l_{pst} \geq \max\left(l_1, l_2\right) \tag{8.57}$$

Tabelul 8.4. Extras din STAS 1004

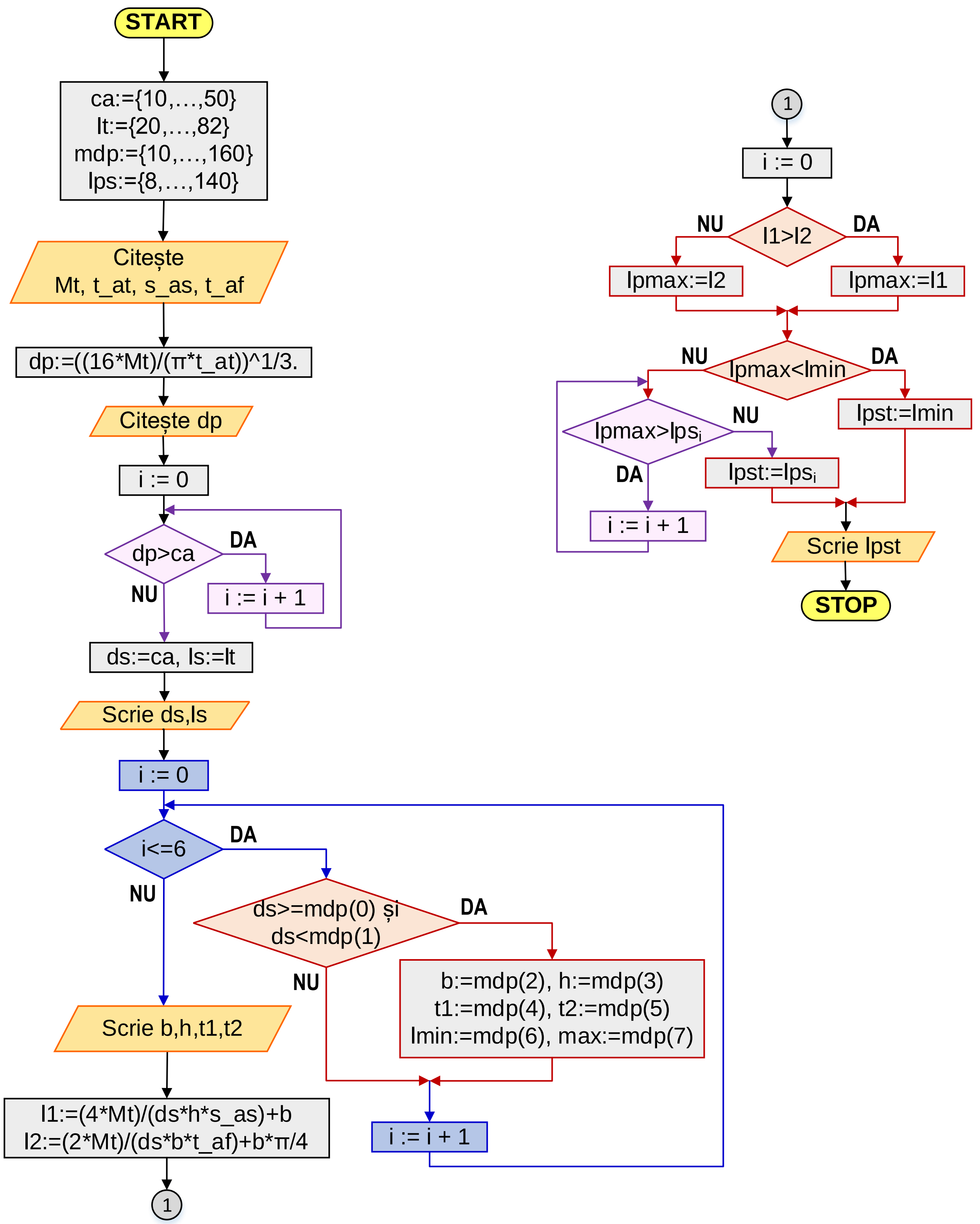
b [mm]	h [mm]	Lungimea STAS [mm]													
6	6	16	18	20	22	25	28	32	36	40	45	50	56	63	70
8	7	20	22	25	28	32	36	40	45	50	56	63	70	80	90
10	8	25	28	32	36	40	45	50	56	63	70	80	90	100	110
12	8		28	32	36	40	45	50	56	63	70	80	90	100	110
14	9				36	40	45	50	56	63	70	80	90	100	110
16	10						45	50	56	63	70	80	90	100	110
18	11							50	56	63	70	80	90	100	110
20	12								56	63	70	80	90	100	110
22	14									63	70	80	90	100	110
25	14										70	80	90	100	110

Algoritmul de rezolvare presupune parcurgerea următoarelor etape:

- citirea de la tastatură a momentului de torsiune la care este supus arborele $\mathbf{M_t}$, în N*mm;
- citirea de la tastatură a rezistenţei admisibile la torsiune $\boldsymbol{\tau_{at}}$, se alege o valoare din intervalul [15, 45], în N/mm^2;
- citirea de la tastatură a valorii rezistenţei admisibile la strivire, $\boldsymbol{\sigma_{as}}$, introducându-se o valoare din intervalul [100, 120] în N/mm^2;
- citirea de la tastatură a valorii tensiunii admisibile la forfecare, $\boldsymbol{\tau_{af}}$, introducându-se o valoare ≤ 100 [N/mm^2];
- se calculează diametrul preliminar **dp**, al capătului de arbore, cu relaţia (8.54);
- cu ajutorul unui ciclu cu test iniţial, se parcurge şirul valorilor standardizate ale diametrelor capetelor de arbore (notat cu **ca**) şi se alege cea mai apropiată valoare din şir, mai mare decât valoarea rezultată din calcul, reţinându-se de asemenea şi valoarea indicelui acesteia. Se afişează valoarea standardizată a diametrului capătului de arbore;
- din şirul valorilor standardizate ale lungimilor tronsoanelor (notat cu **lt**) se alege valoarea al cărui indice este egal cu cel determinat la pasul anterior şi se afişează;
- cu ajutorul unui ciclu cu contor, se parcurge matricea care conţine dimensiunile penei (notată cu **mdp**) şi se determină rândul în care se încadrează diametrul capătului de arbore, se identifică indicele acestui rând şi cu ajutorul acestuia se culeg dimensiunile penei (lăţimea **b** şi înălţimea **h**), adâncimea canalului de pană în arbore **t1**, respectiv în butuc **t2**), precum şi limitele **lmin**, respectiv **lmax** ale lungimii penei;
- se determină lungimile preliminarii ale penei pe baza relaţiilor (8.55) şi (8.56) şi se determină valoarea maximă dintre cele două;
- cu ajutorul unui ciclu cu contor se parcurge şirul valorilor standardizate ale lungimilor penelor **lps**, între limitele stabilite anterior şi se determină cea mai apropiată valoare din şir care este mai mare decât valoarea preliminară. Se afişează valoarea determinată;.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.11b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.11c. Din considerente legate de spaţiu, s-au utilizat valorile standardizate din şiruri corespunzătoare până la diametrul arborelui de maxim 50 mm;

Schema logică



Limbajul pseudocod

```
Început
  ca[21] ← {10,11,12,14,16,18,19,20,22,24,25,28,30,32,35,38,40,42,45,48,50}
  lt[21] ← {20,20,25,25,28,28,28,36,36,36,42,42,58,58,58,58,82,82,82,82,82}
  mdp[7][8] ← { {10,12,4,4,2.5,1.8,8,45},
                {12,17,5,5,3.0,2.3,10,56},
                {17,22,6,6,3.5,2.8,14,70},
                {22,30,8,7,4.0,3.3,18,90},
                {30,38,10,8,5.0,3.3,18,90},
                {38,44,12,8,5.0,3.3,28,140},
                {44,50,14,9,5.5,3.8,36,160} }
  lps[25] ← {8,10,12,14,16,18,20,22,25,28,32,36,40,45,50,56,63,70,80,90,100,110,125,140,160}
  Citeşte Mt, t_at, s_as, t_af
  dp ← (16*Mt/(PI*t_at))^1/3.
  i ← 0
  Cât timp dp > ca[i] execută
     i ← i + 1
  Sfârşit cât timp
  ds ← ca[i],   ls ← lt[i]
  Scrie ds, ls
  Pentru i = 0, 6
        Dacă ds >= mdp[i][0] ŞI ds < mdp[i][1] atunci
              b ← mdp[i][2]
              h ← mdp[i][3]
              t1 ← mdp[i][4]
              t2 ← mdp[i][5]
              lmin ← mdp[i][6]
              lmax ← mdp[i][7]
        Sfârşit dacă
  Sfârşit pentru
  Scrie b, h, t1, t2
  l1 ← (4 * Mt) / (ds * h * s_as) + b
  l2 ← (2 * Mt) / (ds * b * t_af) + b * M_PI / 4
  i ← 0
  Dacă l1 >= l2 atunci
        lpmax ← l1
  altfel
        lpmax ← l2
  Sfârşit dacă
  Dacă lpmax < lmin atunci
        lpst ← lmin
  altfel
          Cât timp lpmax > lps[i] execută
               i ← i + 1
          Sfârşit cât timp
          lpst ← lps[i]
  Sfârşit dacă
  Scrie lpst
Sfârşit
```

Figura 8.11b. Reprezentarea algoritmului calculul de dimensionare al capătului de arbore şi alegerea penei

Programul C şi rularea acestuia

```
#include<stdio.h>
#include<math.h>
int main(void)
{  int i, j;
   float ca[21]={10,11,12,14,16,18,19,20,22,24,25,28,30,32,35,38,40,42,45,48,50};
   float lt[21]={20,20,25,25,28,28,28,36,36,36,42,42,58,58,58,58,82,82,82,82,82};
   float mdp[7][8]={{10,12,4,4,2.5,1.8,8,45},{12,17,5,5,3.0,2.3,10,56},{17,22,6,6,3.5,2.8,14,70},
   {22,30,8,7,4.0,3.3,18,90},{30,38,10,8,5.0,3.3,18,90},{38,44,12,8,5.0,3.3,28,140},
   {44,50,14,9,5.5,3.8,36,160}};
   float lps[25]={8,10,12,14,16,18,20,22,25,28,32,36,40,45,50,56,63,70,80,90,100,110,125,140};
   float dp,ds,ls,Mt,t_at,s_as,t_af,b,h,t1,t2,lmin,lmax,l1,l2,lpmax,lpst,l;
   printf( "\n Momentul de torsiune [N*mm], Mt = " );  scanf( "%f",&Mt );
   printf( "\n Rezistenta admisibila la torsiune [15 ... 45 N/mm2], t_at = " );  scanf( "%f",&t_at );
   printf( "\n Rezistenta admisibila la strivire [100 ... 120 N/mm2], s_as = " );  scanf( "%f",&s_as );
   printf( "\n Tensiunea admisibila la forfecare, [< 100 N/mm2] t_af = " );  scanf( "%f",&t_af );
   dp = pow(16 * Mt / M_PI / t_at,1/3.);
   printf( "\n Diametrul primitiv calculat, dp = %7.3f [mm]",dp );
   i = 0;
   while( dp > ca[i] )
     i++;
   ds = ca[i]; ls = lt[i];
   printf( "\n Diametrul standardizat al capatului de arbore, ds = %7.3f [mm]",ds );
   printf( "\n Lungimea standardizata a capatului de arbore, ls = %7.3f [mm]]",ls );
   for( i = 0 ; i <= 6 ; i++ )
     if( ds >= mdp[i][0] && ds < mdp[i][1] )
     { b = mdp[i][2]; h = mdp[i][3]; t1 = mdp[i][4]; t2 = mdp[i][5]; lmin = mdp[i][6]; lmax = mdp[i][7]; }
   printf( "\n Latimea penei, b = %7.3f [mm]",b );
   printf( "\n Inaltimea penei, h = %7.3f [mm]",h );
   printf( "\n Adancimea canalului in arbore, t1 = %7.3f [mm]",t1 );
   printf( "\n Adancimea canalului in butuc, t2 = %7.3f [mm]",t2 );
   l1 = (4 * Mt) / (ds * h * s_as) + b; l2 = (2 * Mt) / (ds * b * t_af) + b * M_PI / 4;
   i = 0;
   if( l1 > l2 ) lpmax = l1;
   else lpmax = l2;
   if( lpmax < lmin )
      lpst = lmin;
   else
     { while( lpmax > lps[i] )
           i++;
       lpst = lps[i];
     }
   printf( "\n Lungimea standardizata a penei, lpst = %7.3f [mm]",lpst );
}
```

Rularea programului:

Momentul de torsiune [N*mm], Mt = **20000**
Rezistenta admisibila la torsiune [15 ... 45 N/mm2], t_at = **25**
Rezistenta admisibila la strivire [100 ... 120 N/mm2], s_as = **110**
Tensiunea admisibila la forfecare, [< 100 N/mm2] t_af = **50**
Diametrul primitiv calculat, dp = 15.972 [mm]
Diametrul standardizat al capatului de arbore, ds = 16.000 [mm]
Lungimea standardizata a capatului de arbore, ls = 28.000 [mm]]
Latimea penei, b = 5.000 [mm]
Inaltimea penei, h = 5.000 [mm]
Adancimea canalului in arbore, t1 = 3.000 [mm]
Adancimea canalului in butuc, t2 = 2.300 [mm]
Lungimea standardizata a penei, lpst = 16.000 [mm]

Figura 8.11c. Programul C şi rularea acestuia pentru calculul de dimensionare al capătului de arbore şi alegerea penei

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

8.12. Calculul reacţiunilor într-o grindă

Se consideră o grindă (figura 8.12a), sprijinită pe două reazeme, unul fix şi unul mobil, asupra căruia pot să acţioneze:

- **forţe punctiform aplicate**, pentru care se cunosc: **modulul**, **direcţia** (dată de unghiul de înclinare al forţei, măsurat faţă de direcţia orizontală, în sens trigonometric), **sensul**, respectiv **punctul de aplicaţie**;
- **forţe distribuite**, pentru care se cunosc: **modulul pe unitate de lungime**, **sensul**, **punctul de început** şi **punctul de sfârşit**. În calcule forţele distribuite se înlocuiesc cu forţe punctiform aplicate al căror punct de aplicaţie se consideră la jumătatea intervalului de acţiune, modulul acestora rezultând din produsul dintre modulul pe unitatea de lungime şi lungimea de acţiune a forţei distribuite;
- **momente**, care acţionează asupra grinzii şi pentru care se cunosc: **punctul de aplicaţie**, **modulul** şi **sensul**.

În reazemul fix apar două reacţiuni:

- **reacţiunea orizontală**, notată cu **H**, care acţionează în lungul grinzii;
- **reacţiunea verticală**, notată cu **V**, care acţionează perpendicular pe grindă.

În reazemul mobil apare o **reacţiune normală**, notată cu **N**, care acţionează perpendicular pe grindă.

În calcule, pentru forţe se consideră că acestea au valori **pozitive** dacă acţionează în sus, respectiv au valori **negative** dacă acţionează în jos (figura 8.12a). Pentru momente, se consideră că acestea au valori **pozitive** dacă acţionează în sens trigonometric (sens antiorar), respectiv au valori **negative** dacă acţionează în sens orar.

Pentru determinarea reacţiunilor este necesar să se rezolve un sistem format din trei ecuaţii cu trei necunoscute:

- o ecuaţie de forţe, scrisă pe direcţie orizontală, în care avem o singură necunoscută: **reacţiunea orizontală H**;
- o ecuaţie de forţe, scrisă pe direcţie verticală, în care apar două necunoscute: **reacţiunea verticală V** şi **reacţiunea normală N**;
- o ecuaţie de momente, scrisă în raport cu punctul de aplicaţie al reazemului fix, în care apare o singură necunoscută: **reacţiunea normală N**.

Se observă că din prima ecuaţie se poate determina direct **reacţiunea orizontală H**. De asemenea, din ultima ecuaţie se poate determina direct **reacţiunea normală N**. Cunoscând valoarea **reacţiunii normale N**, din cea de a doua ecuaţie se determină **reacţiunea verticală V**.

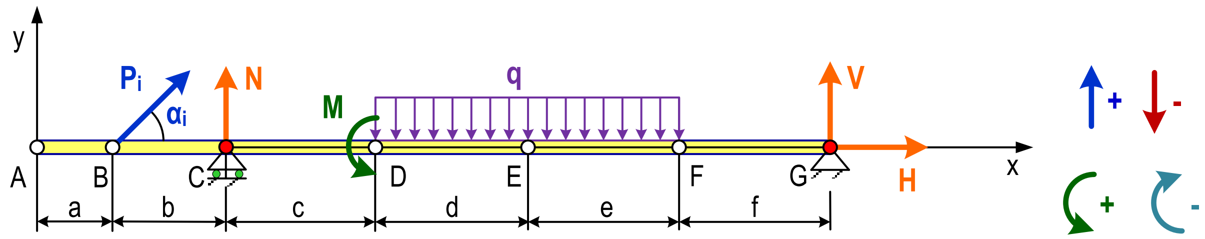


Figura 8.12a. Grindă cu forţe

Algoritmul de calcul al reacţiunilor din reazeme presupune parcurgerea următorilor paşi:

A. Se citeşte de la tastatură lungimea grinzii;
B. Se citeşte de la tastatură poziţia reazemului fix;
C. Se citeşte de la tastatură poziţia reazemului mobil;
D. Se citeşte de la tastatură numărul de forţe punctiform aplicate;
E. Utilizând un ciclu cu contor, se citeşte de la tastatură pentru fiecare forţă aplicată punctiform: **modulul**, **direcţia** (unghiul pe care îl face forţa cu orizontala, măsurat în sens trigonometric), respectiv coordonata **x** a punctului de aplicaţie al forţei;
F. Se citeşte de la tastatură numărul de forţe distribuite;
G. Utilizând un ciclu cu contor, se citeşte pentru fiecare forţă distribuită: **modulul** (dacă forţa acţionează în sus se introduce o valoare pozitivă, iar dacă forţa acţionează în jos se introduce o valoare negativă), coordonata **x** a punctului de unde începe acţiunea forţei distribuite, respectiv coordonata **x** a punctului unde se sfârşeşte acţiunea forţei distribuite;

H. Se citeşte de la tastatură numărul de momente;

I. Utilizând un ciclu cu contor, pentru fiecare moment se introduce **modulul** (dacă momentul acţionează în sens trigonometric – antiorar, se introduce o valoare pozitivă, iar dacă momentul acţionează în sens orar se introduce o valoare negativă);

J. Se iniţializează valoarea reacţiunii orizontale (notată cu **H** în program) cu zero (**H = 0**);

K. Utilizând un ciclu cu contor, se calculează valoarea reacţiunii orizontale, scăzând din valoarea acesteia valoarea componentei orizontale ale fiecărei forţe punctiform aplicate;

L. Se iniţializează valoarea momentului total (notat cu **MT** în program) cu zero (**MT = 0**);

M. Utilizând un ciclu cu contor, se adaugă la valoarea momentului total valoarea fiecărui moment care acţionează asupra grinzii;

N. Utilizând un ciclu cu contor, se adaugă la valoarea momentului total valoarea momentelor date de fiecare forţă punctiform aplicată în raport cu punctul în care se găseşte reazemul fix;

O. Utilizând un ciclu cu contor se adaugă la momentul total valoarea momentelor date de forţele distribuite, în raport cu punctul în care se găseşte reazemul fix. Pentru a calcula aceste momente se înlocuiesc forţele distribuite cu forţe punctiform aplicate al căror modul este dat de produsul dintre modulul forţei distribuite şi lungimea pe care acţionează aceasta, iar punctul de aplicaţie se află la jumătatea distanţei pe care acţionează forţa distribuită. Momentul dat de o forţă distribuită este dat de produsul dintre modulul forţei punctiform aplicată care înlocuieşte forţa distribuită şi distanţa dintre poziţia reazemului fix şi cea a punctului de aplicaţie a forţei punctiform aplicată care înocuieşte forţa distribuită;

P. Se calculează reacţiunea normală **N** prin împărţirea momentului total calculat anterior la distanţa dintre cele două reazeme;

R. Se iniţializează valoarea variabilei **SFV** (suma forţelor verticale) cu zero (**SFV = 0**);

S. Utilizând un ciclu cu contor, se adaugă variabilei **SFV** componenta verticală a fiecărei forţe punctiform aplicate;

T. Utilizând un ciclu cu contor se adaugă variabilei **SFV** valoarea forţei punctiform aplicate care înlocuieşte forţa distribuită, pentru fiecare forţă distribuită în parte;

U. Se calculează reacţiunea verticală **V**;

V. Se afişează cele trei reacţiuni calculate: **orizontală H**, **verticală V** şi **normală N**;

În cadrul programului s-au făcut următoarele notaţii:

nfpa – numărul de forţe punctiform aplicate;

nfd – numărul de forţe distribuite;

nm – numărul de momente aplicate;

lg – lungimea grinzii [cm];

prfix – poziţia reazemului fix, adică coordonata x a punctului unde este plasat reazemul fix, considerând ca origine extremitatea stângă a grinzii [cm];

prmob – poziţia reazemului mobil, adică coordonata x a punctului unde este plasat reazemul mobil, considerând ca origine extremitatea stângă a grinzii [cm];

MFPA – modulul forţei punctiform aplicate [N];

UFPAg – unghiul pe care îl face forţa punctiform aplicată cu direcţia orizontală, măsurat în sens trigonometric [grade];

UFPAr - unghiul pe care îl face forţa punctiform aplicată cu direcţia orizontală, măsurat în sens trigonometric [radiani];

PFPA – poziţia punctului în care acţionează forţa punctiform aplicată [cm];

MFD – modulul forţei distribuite [N/cm];

PINFD – poziţia punctului de început a lungimii pe care acţionează forţa distribuită [cm];

PSFFD – poziţia punctului de sfârşit a lungimii pe care acţionează forţa distribuită [cm];

MFDP – modulul forţei punctiform aplicate, care înlocuieşte forţa distribuită [N];

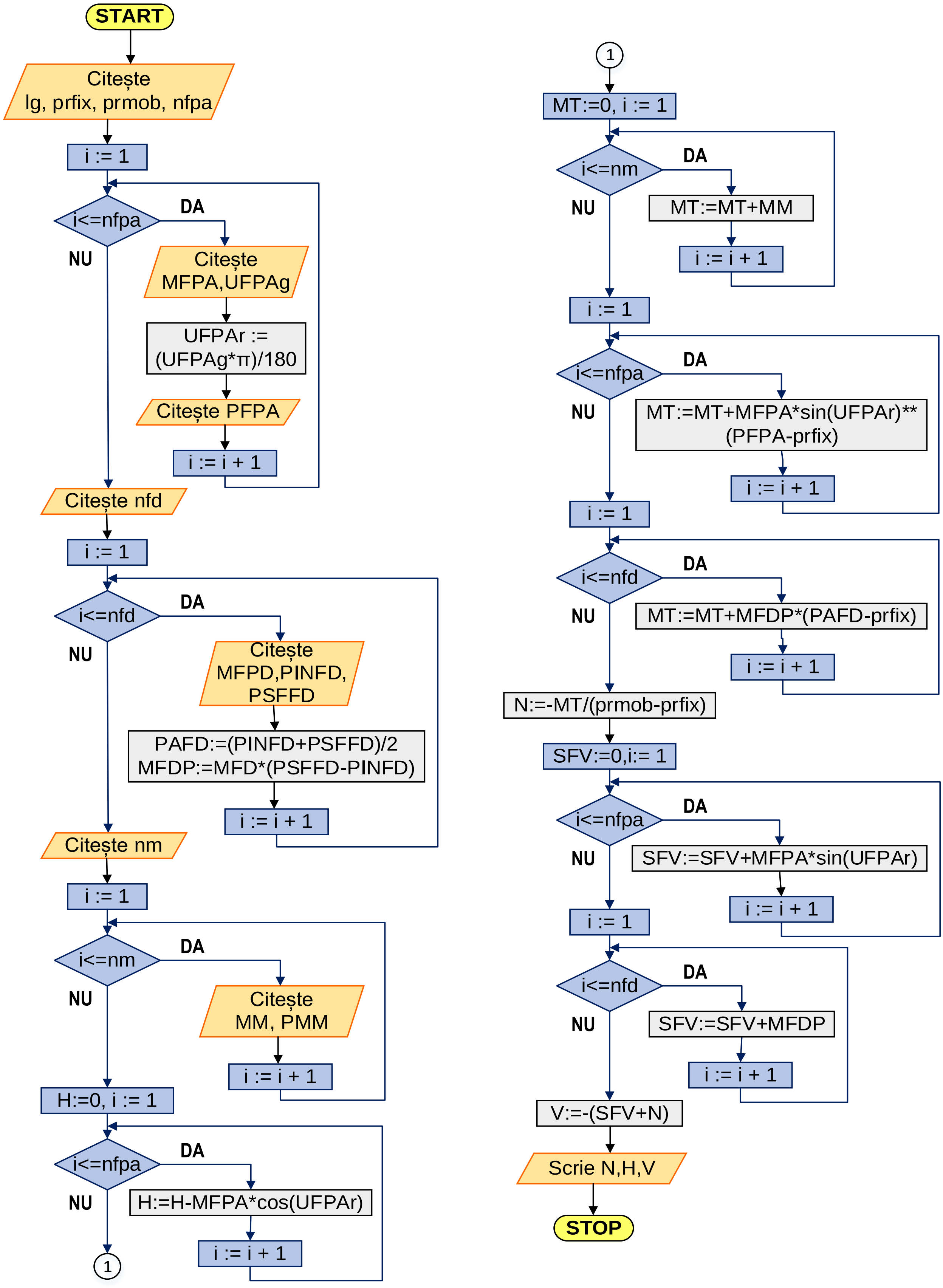
PAFD – poziţia punctului în care acţionează forţa punctiform aplicată care înlocuieşte forţa ditribuită [cm];

MM – modulul unui moment aplicat grinzii [N*cm];

PMM – poziţia punctului unde este aplicat momentul [cm];

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.12b, iar programul în limbajul C şi rularea acestuia sunt prezentate în figura 8.12c.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte lg, prfix, prmob
  Citeşte nfpa
  Pentru i = 1, nfpa
     Citeşte MFPA[i], UFPAg[i], PFPA[i]
     UFPAr[i] ← UFPAg[i] * PI/180
  Sfârşit pentru
  Citeşte nfd
  Pentru i = 1, nfd
     Citeşte MFD[i], PINFD[i], PSFFD[i]
     PAFD[i] ← (PINFD[i]+PSFFD[i]) / 2
     MFDP[i] ← MFD[i] * (PSFFD[i] - PINFD[i])
  Sfârşit pentru
  Citeşte nm
  Pentru i = 1, nm
     Citeşte MM[i], PMM[i]
  Sfârşit pentru
  H ← 0
  Pentru i = 1, nfpa
     H ← H - MFPA[i] * cos(UFPAr[i])
  Sfârşit pentru
  MT ← 0
  Pentru i = 1, nm
     MT ← MT + MM[i]
  Sfârşit pentru
  Pentru i = 1, nfpa
     MT ← MT + MFPA[i] * sin(UFPAr[i]) * (PFPA[i] - prfix)
  Sfârşit pentru
  Pentru i = 1, nfd
     MT ← MT + MFDP[i] * (PAFD[i] - prfix)
  Sfârşit pentru
  N ← - MT / (prmob - prfix)
  SFV ← 0
  Pentru i = 1, nfpa
     SFV ← SFV + MFPA[i] * sin(UFPAr[i])
  Sfârşit pentru
  Pentru i = 1, nfd
     SFV ← SFV + MFDP[i]
  Sfârşit pentru
  V ← -(SFV + N)
  Scrie N, H, V
Sfârşit
```

Figura 8.12b. Reprezentarea algoritmului calculul reacţiunilor din reazeme pentru o grindă încărcată cu sarcini

Programul C şi rularea acestuia

```
#include<stdio.h>
#include<math.h>
int main(void)
{ int i, nfpa, nfd, nm;
  float lg, prfix, prmob,MFPA[10],UFPAg[10],UFPAr[10],PFPA[10], MFD[10],PINFD[10];
  float PSFFD[10],PAFD[10],MFDP[10], MM[10], PMM[10], MT, SFV, H, V, N;
  printf( "\n Lungimea grinzii [cm], lg = " ); scanf( "%f",&lg );
  printf( " Pozitia reazemului fix [cm], prfix = " ); scanf( "%f",&prfix );
  printf( " Pozitia reazemului mobil [cm], prmob = " ); scanf( "%f",&prmob );
  printf( "\n Numarul de forte punctiform aplicate, nfpa = " ); scanf( "%d",&nfpa );
  for( i = 1 ; i <= nfpa ; i++ )
    {  printf( " Modulul fortei pct. aplic. [N], MFPA[%d] = ",i ); scanf( "%f",&MFPA[i] );
       printf( " Unghiul fortei pct. aplic. [grade], UFPA[%d] = ",i ); scanf( "%f",&UFPAg[i] );
       UFPAr[i] = UFPAg[i] * M_PI/180.0;
       printf( " Pozitia fortei pct. aplic. [cm], PFPA[%d] = ",i ); scanf( "%f",&PFPA[i] );  }
  printf( "\n Numarul de forte distribuite, nfd = " ); scanf( "%d",&nfd );
  for( i = 1 ; i <= nfd ; i++ )
    { printf( " Modulul fortei distribuite [N/cm], MFD[%d] = ",i ); scanf( "%f",&MFD[i] );
      printf( " Punctul de start al fortei distrib. [cm], PINFD[%d] = ",i ); scanf( "%f",&PINFD[i] );
      printf( " Punctul final al fortei distrib. [cm], PSFFD[%d] = ",i ); scanf( "%f",&PSFFD[i] );
      PAFD[i]=(PINFD[i]+PSFFD[i])/2;  MFDP[i]=MFD[i]*(PSFFD[i]-PINFD[i]);  }
  printf( "\n Numarul de momente, nm = " ); scanf( "%d",&nm );
  for( i = 1 ; i <= nm ; i++ )
   { printf( " Modulul momentului [N*cm], MM[%d] = ",i ); scanf( "%f",&MM[i] );
     printf( " Punctul unde se aplica momentului [cm], PMM[%d] = ",i ); scanf( "%f",&PMM[i] );  }
// Ecuatia de forte pe directie orizontala
  for( H = 0, i = 1 ; i <= nfpa ; i++ )
    H = H - MFPA[i] * cos(UFPAr[i]);
// Ecuatia de momente
  for( MT = 0, i = 1 ; i <= nm ; i++ )
    MT = MT + MM[i];
  for( i = 1 ; i <= nfpa ; i++ )
    MT = MT + MFPA[i] * sin(UFPAr[i]) * (PFPA[i] - prfix);
  for( i = 1 ; i <= nfd ; i++ )
    MT = MT + MFDP[i] * (PAFD[i] - prfix);
  N = - MT / (prmob - prfix);
// Ecuatia de forte pe directie verticala
  for( SFV = 0, i = 1 ; i <= nfpa ; i++ )
    SFV = SFV + MFPA[i] * sin(UFPAr[i]);
  for( i = 1 ; i <= nfd ; i++ )
    SFV = SFV + MFDP[i];
  V = - (SFV + N);
  printf( "\n Reactiunea normala, N = %9.3f [N]",N );
  printf( "\n Reactiunea orizontala, H = %9.3f [N]",H );
  printf( "\n Reactiunea verticala, V = %9.3f [N]",V );
}
```

Rularea programului:

Cazul 1:

Lungimea grinzii [cm], lg = **100**
Pozitia reazemului fix [cm], prfix = **10**
Pozitia reazemului mobil [cm], prmob = **90**
Numarul de forte punctiform aplicate, nfpa = **2**
Modulul fortei pct. aplic. [N], MFPA[1] = **300**
Unghiul fortei pct. aplic. [grade], UFPA[1] = **270**
Pozitia fortei pct. aplic. [cm], PFPA[1] = **0**
Modulul fortei pct. aplic. [N], MFPA[2] = **500**
Unghiul fortei pct. aplic. [grade], UFPA[2] = **60**
Pozitia fortei pct. aplic. [cm], PFPA[2] = **50**
Numarul de forte distribuite, nfd = **1**
Modulul fortei distribuite [N/cm], MFD[1] = **-10**
Punctul de start al fortei distrib. [cm], PINFD[1] = **30**
Punctul final al fortei distrib. [cm], PSFFD[1] = **90**
Numarul de momente, nm = **1**
Modulul momentului [N*cm], MM[1] = **2000**
Punctul unde se aplica momentului [cm], PMM[1] = **10**

Reactiunea normala,   N =   95.994 [N]
Reactiunea orizontala, H = -250.000 [N]
Reactiunea verticala,   V =   370.994 [N]

Cazul 2:

Lungimea grinzii [cm], lg = **100**
Pozitia reazemului fix [cm], prfix = **90**
Pozitia reazemului mobil [cm], prmob = **10**
Numarul de forte punctiform aplicate, nfpa = **2**
Modulul fortei pct. aplic. [N], MFPA[1] = **400**
Unghiul fortei pct. aplic. [grade], UFPA[1] = **90**
Pozitia fortei pct. aplic. [cm], PFPA[1] = **100**
Modulul fortei pct. aplic. [N], MFPA[2] = **500**
Unghiul fortei pct. aplic. [grade], UFPA[2] = **240**
Pozitia fortei pct. aplic. [cm], PFPA[2] = **20**
Numarul de forte distribuite, nfd = **1**
Modulul fortei distribuite [N/cm], MFD[1] = **-15**
Punctul de start al fortei distrib. [cm], PINFD[1] = **20**
Punctul final al fortei distrib. [cm], PSFFD[1] = **60**
Numarul de momente, nm = **1**
Modulul momentului [N*cm], MM[1] = **-3000**
Punctul unde se aplica momentului [cm], PMM[1] = **90**

Reactiunea normala,   N =   766.386 [N]
Reactiunea orizontala, H =   250.000 [N]
Reactiunea verticala,   V = -133.373 [N]

Figura 8.12c. Programul C şi rularea acestuia pentru calculul reacţiunilor din reazeme pentru o grindă încărcată cu sarcini

Observaţie: valorile **bolduite** de la rularea programului corespund datelor introduse de utilizator de la tastatură.

Cazul 1: Se consideră grinda din figura 8.12d, pentru care avem:
- lungimea grinzii: **100** [cm];
- poziţia reazemului fix: **10** [cm];
- poziţia reazemului mobil: **90** [cm];
- o forţă punctiform aplicată $\mathbf{P_1}$, având: poziţia punctului de aplicaţie **0** [cm], modulul **300** [N], unghiul de orientare **270** [°];
- o forţă punctiform aplicată $\mathbf{P_2}$ având: poziţia punctului de aplicaţie **50** [cm], modulul **500** [N], unghiul de orientare **60** [°];
- o forţă distribuită **q** având: modulul pe unitatea de lungime **-10** [N/cm], poziţia punctului de început al forţei **30** [cm], poziţia punctului final al forţei **90** [cm]. În calcule această forţă se înlocuieşte cu o forţă punctiform aplicată, notată **Q**, al cărei modul este dat de produsul dintre modulul pe unitatea de lungime şi lungimea ocupată pe grindă, adică **-600** [N], punctul de aplicaţie al acestei forţe fiind la jumătatea distanţei ocupate de forţa distribuită, adică **60** [cm];
- un moment **M** având: modulul de **2000** [N*cm] şi punctul de aplicaţie **10** [cm];

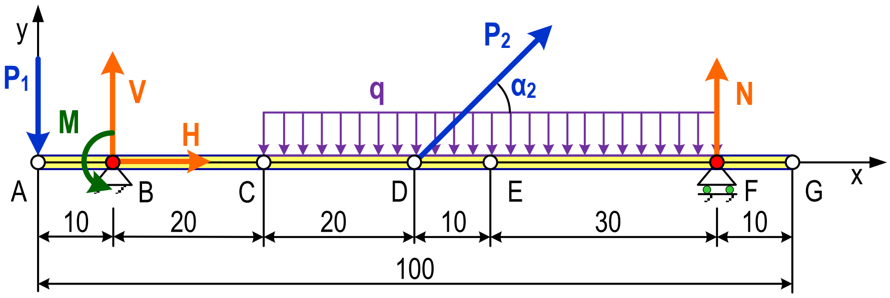


Figura 8.12d. Grindă încărcată – **cazul 1**

Rezolvare:

A. Se scrie mai întâi ecuaţia de forţe pe direcţie orizontală, cu ajutorul căreia se determină **reacţiunea orizontală H**:

$$\boldsymbol{H} + P_2 \cdot cos(\alpha_2) = 0$$

de unde rezultă:

$$\boldsymbol{H} = -P_2 \cdot cos(\alpha_2) = -500 \cdot cos(60) = \mathbf{-250.000} \text{ [N]}$$

B. Se scrie ecuaţia de momente în raport cu punctul de aplicaţie al reazemului fix, rezultând **reacţiunea normală N**:

$$P_1 \cdot AB + M + P_2 \cdot sin(\alpha_2) \cdot BD - q \cdot CF \cdot BE + \boldsymbol{N} \cdot BF = 0$$

de unde rezultă:

$$\boldsymbol{N} = \frac{-P_1 \cdot AB - M - P_2 \cdot sin(\alpha_2) \cdot BD + q \cdot CF \cdot BE}{BF}$$

$$\boldsymbol{N} = \frac{-300 \cdot 10 - 2000 - 500 \cdot 0.866025 \cdot 40 + 10 \cdot 60 \cdot 50}{80} = \frac{-3000 - 2000 - 17320.508 + 30000}{80}$$

$$\boldsymbol{N} = \frac{-3000 - 2000 - 17320.508 + 30000}{80} = \frac{7679.492}{80} = \mathbf{95.994} \text{ [N]}$$

C. Se scrie ecuaţia de forţe pe direcţie verticală, cu ajutorul căreia se determină **reacţiunea verticală V**:

$$-P_1 + \boldsymbol{V} + q * CF + P_2 \cdot sin(60) + N = 0$$

de unde rezultă:

$$\boldsymbol{V} = P_1 - q * CF - P_2 \cdot sin(60) - N = 300 - (-10) \cdot 60 - 500 \cdot 0.866025 - 95.994 = \mathbf{370.994} \text{ [N]}$$

Cazul 2: Se consideră grinda din figura 8.12e, pentru care avem:
- lungimea grinzii: **100** [cm];
- poziţia reazemului fix: **90** [cm];
- poziţia reazemului mobil: **10** [cm];
- o forţă punctiform aplicată P_1, având: poziţia punctului de aplicaţie **0** [cm], modulul **400** [N], unghiul de orientare **90** [°];
- o forţă punctiform aplicată P_2 având: poziţia punctului de aplicaţie **20** [cm], modulul **500** [N], unghiul de orientare **240** [°];
- o forţă distribuită **q** având: modulul pe unitatea de lungime **-15** [N/cm], poziţia punctului de început al forţei **20** [cm], poziţia punctului final al forţei **60** [cm]. În calcule această forţă se înlocuieşte cu o forţă punctiform aplicată, notată **Q**, al cărei modul este dat de produsul dintre modulul pe unitatea de lungime şi lungimea ocupată pe grindă, adică **-600** [N], punctul de aplicaţie al acestei forţe fiind la jumătatea distanţei ocupate de forţa distribuită, adică **40** [cm];
- un moment **M** având: modulul de **-3000** [N*cm] şi punctul de aplicaţie **90** [cm];

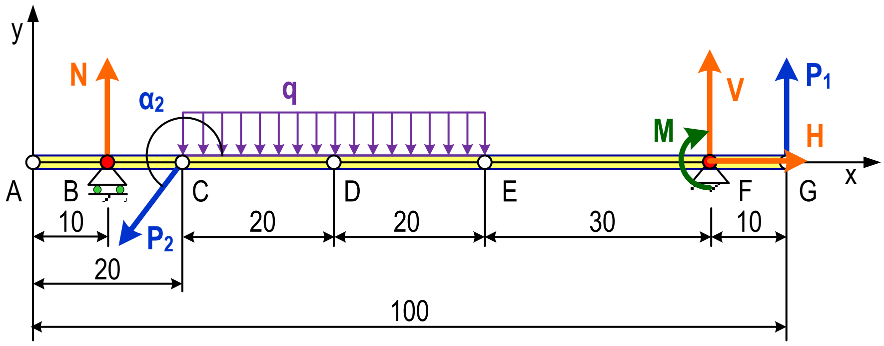


Figura 8.12e. Grindă încărcată – **cazul 2**

Rezolvare:

A. Se scrie mai întâi ecuaţia de forţe pe direcţie orizontală, cu ajutorul căreia se determină **reacţiunea orizontală H**:

$$\boldsymbol{H} + P_2 \cdot cos(\alpha_2) = 0$$

de unde rezultă:

$$\boldsymbol{H} = -P_2 \cdot cos(\alpha_2) = -500 \cdot cos(240) = \mathbf{250.000}\ [\mathrm{N}]$$

B. Se scrie ecuaţia de momente în raport cu punctul de aplicaţie al reazemului fix, rezultând **reacţiunea normală N**:

$$-\boldsymbol{N} \cdot BF - P_2 \cdot sin(\alpha_2) \cdot CF - q \cdot CE \cdot DF + M + P_1 \cdot FG = 0$$

de unde rezultă:

$$\boldsymbol{N} = \frac{-P_2 \cdot sin(\alpha_2) \cdot CF - q \cdot CE \cdot DF + M + P_1 \cdot FG}{BF}$$

$$\boldsymbol{N} = \frac{-500 \cdot (-0.866025) \cdot 70 - (-15) \cdot 40 \cdot 50 - 3000 + 400 \cdot 10}{80}$$

$$\boldsymbol{N} = \frac{30310.875 + 30000 - 3000 + 4000}{80} = \frac{61310.875}{80} = \mathbf{766.386}\ [\mathrm{N}]$$

C. Se scrie ecuaţia de forţe pe direcţie verticală, cu ajutorul căreia se determină **reacţiunea verticală V**:

$$N + P_2 \cdot sin(\alpha_2) + q * CE + \boldsymbol{V} + P_1 = 0$$

de unde rezultă:

$$\boldsymbol{V} = -P_1 - q * CE - P_2 \cdot sin(\alpha_2) - N = -400 - (-15) \cdot 40 - 500 \cdot (-0.866025) - 766.386$$

$$\boldsymbol{V} = \mathbf{-133.373}\ [\mathrm{N}]$$

Bibliografie

1. Antal, T.A. - *The C ANSI programming language*, Editura RISOPRINT, 2001, ISBN 973-656-065-1;
2. Arghir, M., Deteşan, O. A., *Bazele informaticii,* Editura Todesco, Cluj-Napoca, 2000, 180 pag., ISBN 973-99779-3-6;
3. Arghir, M., Deteşan, O. A., Şoancă, A., *Limbajul C – îndrumător de lucrări*, Editura Quo Vadis, Cluj-Napoca, 2001, 118 pag., ISBN 973-8312-00-0;
4. Arghir, Mariana, Deteşan, Ovidiu A., *Utilizarea calculatorului şi programarea în limbajul C*, Editura U.T. Pres, Cluj-Napoca, 2005, 332 pag., ISBN 973-662-198-7;
5. Barbu, Gh., Bănică, L., Păun, V. – *Calculatoare personale – Arhitectura, funcţionare şi interconectare*, Editura MATRIX ROM, Bucureşti, 2011, ISBN 978-973-755-739-1;
6. Damian, C. – *Iniţiere în limbajul C*, Editura Teora, Bucureşti, 1996;
7. Deshpande, P.S., Kakde, O.G. - *C & Data Structures*, Charles River Media, 2004 (700 pages);
8. Kernigham, B., Ritchie, D. – *The C Programming Language*, Prentice – Hall, Inc, Englewood Cliffs, New Jersey, 1978;
9. Laslo, E., Ionescu, V.S. – *Algoritmică C++*, Editura MATRIX ROM, Bucureşti, 2010, ISBN 978-973-755-640-0;
10. Lupea, I., Lupea, M. – *Limbajul C. Teorie şi aplicaţii*. Editura Casa Cărţii de Ştiinţă, Cluj-Napoca, 1998;
11. Muşlea, I. – *Iniţiere în C++*, Editura MicroInformatica, Cluj-Napoca, 1993;
12. Negrescu, L. – *Limbajele C şi C++ pentru începători, Vol. I. – Limbajul C*, Editura Albastră, Cluj-Napoca, 1996;
13. Perry, G. - *C by Example*, Que Corporation, 2000, ISBN 0-7897-2239-9;
14. Petrovici, V., Goicea, F. – *Programarea în limbajul C*, Editura Tehnică, Bucureşti, 1993;
15. Popescu, D.I. – *Programarea în limbajul C*, Editura DSG Press, Dej, 1999, ISBN 973-98621-4-4;
16. Schildt, H. - *C/C++ Programmer's Reference*, Third Edition, McGraw-Hill/Osborne, 2003 (358 pages);
17. Sedgewick, R. – *Algorithms in C*, Addison – Wesley, 1990 ISBN 0-201-51425-7;
18. Smiley, J. – *Learn to program with C++*, MacGraw-Hill, 2003, ISBN-13: 978-0072225358;
19. Stroustrup, B. – *The C++ Programming*, Addison – Wesley, 1997, ISBN 0-201-88954-4;
20. Ursu-Fischer, N., Ursu, M. – *Programarea cu C în inginerie*, Editura Casa Cărţii de Ştiinţă, Cluj-Napoca, 2001, ISBN 973-686-227-5;

Anexa A. Mediul de programare Dev C++

Mediul de programare Dev C++ permite scrierea şi compilarea unor programe al căror cod sursă este scris în limbajul C/ C++. Cu ajutorul mediului de programare Dev C++ se pot crea aplicaţii Windows.

A.1. Instalarea mediului de programare

Pentru a realiza instalarea mediului de programare Dev C++ pe calculatorul dvs. trebuie să parcurgeţi următoarele etape:

A. Obţinerea fişierului **devcpp4.9.9.2_setup.exe** care conţine kit-ul de instalare, de pe unul din site-urile: http://www.soft32.com/, http://www.bloodshed.net/dev/devcpp.html, dev-c.ro.malavida.com, http://sourceforge.net

B. Se lansează în execuţie fişierul **devcpp4.9.9.2_setup.exe**. Pe ecran se deschide fereastra din figura A.1.

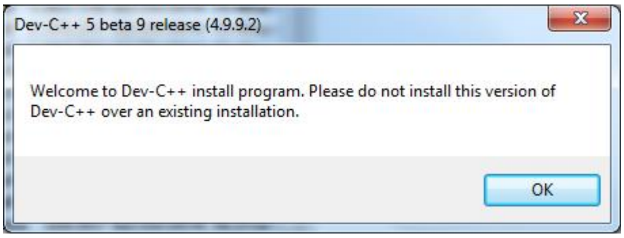


Figura A.1.

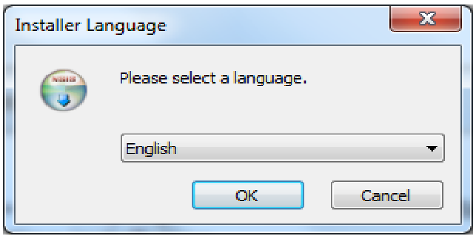


Figura A.2.

Se acţionează butonul **Ok**. Pe ecran se deschide fereastra ***Installer Language*** (figura A.2). Se selectează limba utilizată în etapele de instalare (**selectaţi limba română**) şi se acţionează butonul **Ok**.

C. **Acceptarea contractului de licenţă**. După parcurgerea etapei anterioare, pe ecran se deschide fereastra ***Contract de licenţă*** (figura A.3). Se acţionează butonul **De acord.**

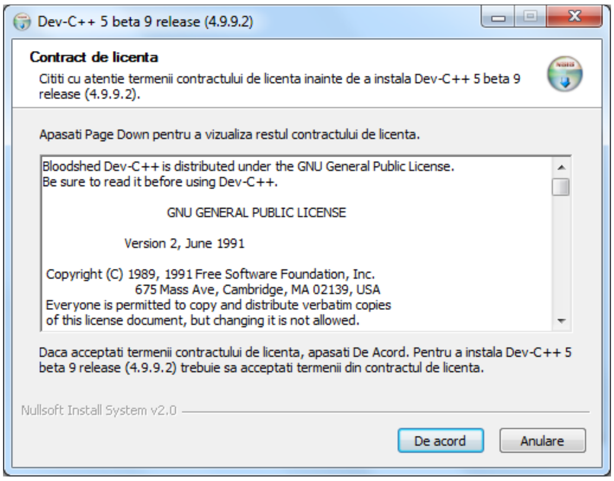


Figura A.3.

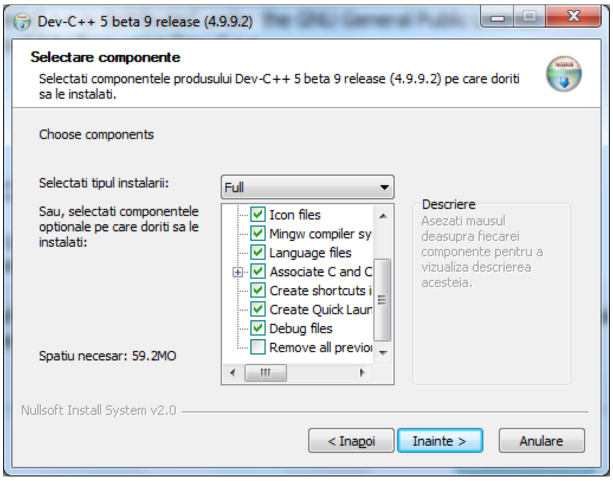


Figura A.4.

D. **Selectarea componentelor pe care dorim să le instalăm**. După acceptarea contractului de licenţă pe ecran se deschide fereastra ***Selectare componente*** (figura A.4). **NU SE MODIFICĂ** configuraţia implicită şi se acţionează butonul **Înainte**.

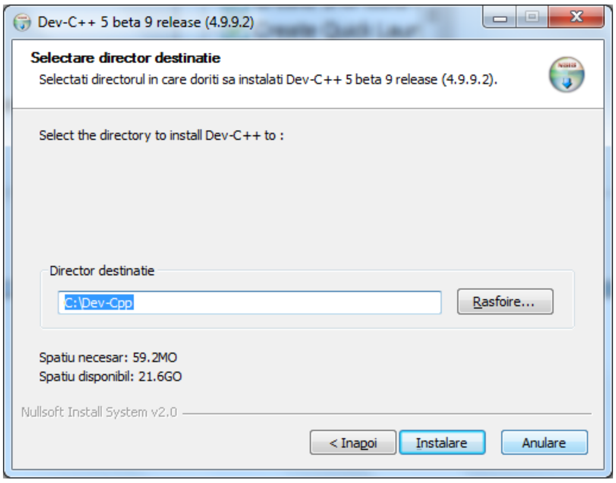


Figura A.5.

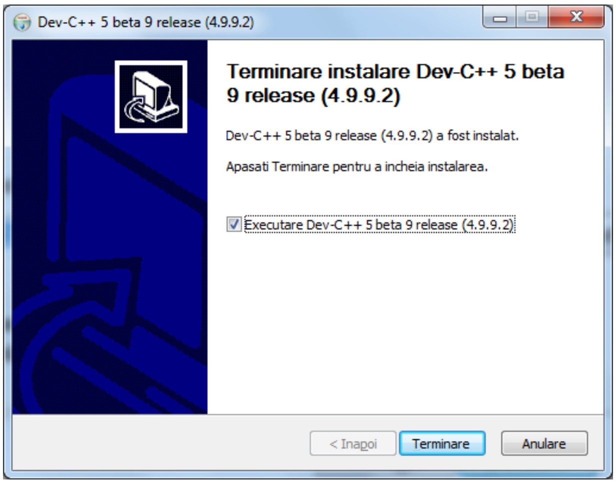


Figura A.6.

E. Selectarea directorului destinaţie. După selectarea componentelor ce urmează a fi instalate, pe ecran se deschide fereastra ***Selectare director destinaţie*** (figura A.5). Dacă se doreşte, se poate modifica directorul unde să fie instalate fişierele. Se acţionează butonul **Instalare**, în câteva secunde se realizează instalarea mediului de programare.

F. Finalizarea instalării. După instalarea fişierelor pe calculatorul dvs. în directorul specificat, pe ecran se deschide fereastra prezentată în figura A.6. Pentru finalizarea instalării se acţionează butonul **Terminare**.

G. Lansarea în execuţie a aplicaţiei. După finalizarea instalării, dacă este activată opţiunea ***Executare Dev-C++ 5 beta 9 release (4.9.9.2)*** pe ecran se deschide aplicaţia ***Dev-C++***, a cărei interfaţă este prezentată în figura A.7.

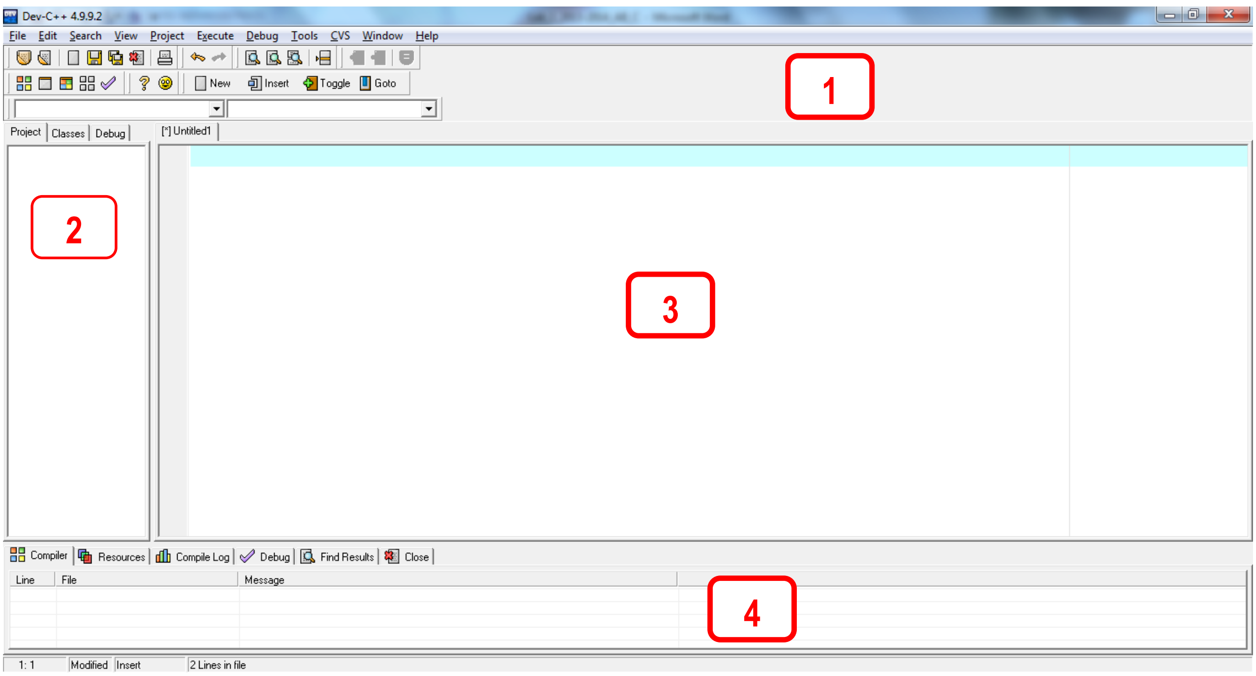


Figura A.7. Interfaţa mediului de programare Dev – C++

A.2. Prezentarea mediului de programare

Interfaţa mediului de programare este prezentată în figura A.7. Elementele componente ale acesteia sunt:
1 – meniul principal şi barele cu unelte;
2 – fereastra în care sunt afişate numele proiectului şi a fişierelor utilizate – **file explorer**;
3 – editorul de texte;
4 – zona de afişare a mesajelor şi rezultatelor;

Meniul principal conţine 11 meniuri, în continuare fiind prezentare cele mai importante.

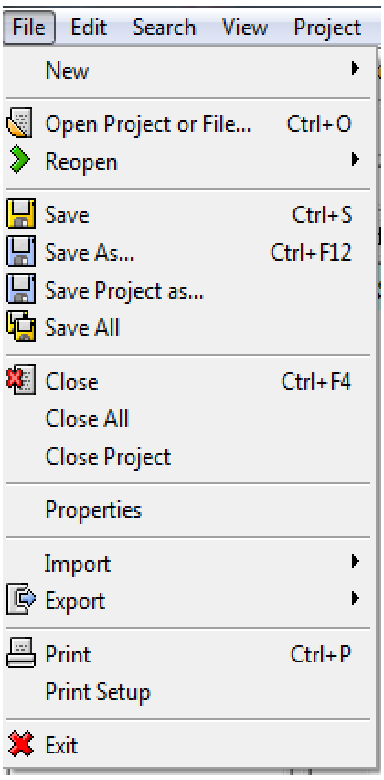


Figura A.8. Meniul **File**

Meniul **File** (figura A.8): se utilizează pentru gestionarea fişierelor, având opţiunile:
New: permite crearea unui element nou (fişier sursă, proiect, fişier de resurse, şablon);
Open Project or File … : deschide un fişier sau proiect existent;
Reopen: afişează o listă cu ultimele fişiere deschise (istoric) şi permite deschiderea acestora;
Save: salvarea fişierului / proiectului curent;
Save as …: salvarea fişierului curent sub un alt nume sau un alt format;
Save Project as …: salvarea proiectului sub un alt nume;
Save All: salvarea tuturor fişierelor deschise;
Close: închiderea fişierului curent;
Close All: închiderea tuturor fişierelor deschise;
Close Project: închiderea proiectului curent;
Properties: afişează fereastra Properties;
Import: importarea unui proiect creat cu MS Visual Studio;
Export: conversia fişierului (proiectului) curent într-un alt din format (HTML, RTF);
Print: permite tipărirea fişierului / proiectului curent;
Print Setup: stabilirea setărilor referitoare la imprimanta utilizată;
Exit: părăsirea mediului de programare;

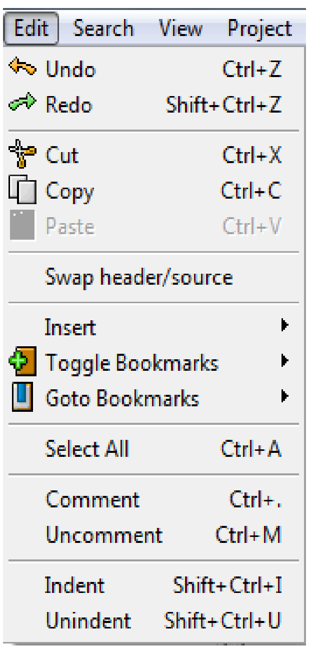


Figura A.9. Meniul **Edit**

Meniul **Edit**: se utilizează pentru operaţii în cadrul editorului de texte (figura A.9):
Undo: anulează ultima comandă de editare;
Redo: anulează efectul ultimei comenzi Undo;
Cut: decupează zona de text selectată şi o plasează în memoria tampon;
Copy: copiază zona de text selectată şi o plasează în memoria tampon;
Paste: inserează în poziţia curentă a cursorului, conţinutul memoriei tampon;
Swap header / source: permite selectarea alternativă a fişierelor sursă şi header;
Insert: permite inserarea datei şi a orei (Date / Time) sau a unor câmpuri de comentariu în poziţia curentă a cursorului;
Toggle Bookmarks: inserează un semn de carte;
Goto Bookmarks: realizează deplasarea la semnul de carte ales;
Select All: realizează selectarea întregului conţinut al fişierului curent;
Comment: transformă linia curentă în linie de comentariu;
Uncomment: transformă linia curentă din linie de comentariu într-o linie de program;
Indent: permite deplasarea la dreapta cu un număr de poziţii a textului selectat;
Unindent: permite realinierea textului deplasat la dreapta, adică anularea comenzii **Indent**;

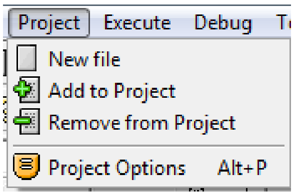


Figura A.10. Meniul **Project**

Meniul **Project**: se utilizează pentru gestionarea proiectelor (figura A.10):
New file: generează un fişier nou;
Add to Project: adaugă fişierul curent unui proiect existent;
Remove from Project: elimină un fişier din proiectul curent;
Project Options: permite stabilirea setărilor pentru proiectul curent;

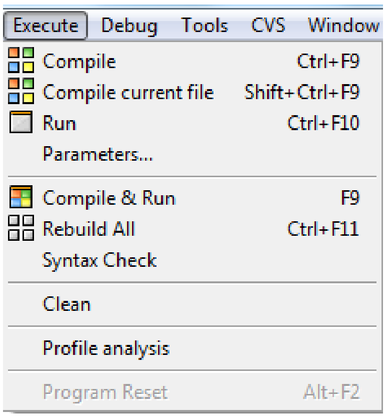


Figura A.11. Meniul **Execute**

Meniul **Execute**: permite generarea fişierelor obiect / executabile (figura A.11):
Compile: realizează compilarea fişierelor deschise;
Compile current file: realizează compilarea fişierului curent;
Run: lansează în execuţie programul compilat;
Parameters…: permite urmărirea parametrilor programului;
Compile & Run: realizează succesiv etapele de compilare şi lansare în execuţie;
Rebuild All: reconstruieşte toate fişierele;
Syntax Check: verifică fişierul din punct de vedere al sintaxei;
Clean: curăţă fişierul curent;
Profile analysis: realizează o analiză a proiectului / fişierului;
Program reset: realizează resetarea programului;

Anexa B. Mediul de programare Code::Blocks (cu compilator)

Code::Blocks este un mediu de programare sau un mediu integrat de dezvoltare (**IDE**) gratuit ce conține facilitățile necesare de care are nevoie un programator pentru a crea aplicații:
- **editor de text**: necesar pentru scrierea programelor sau a codurilor sursă – fișiere cu extensia **.c** (pentru limbajul C) și **.cpp** (pentru limbajul C++).
- **compilatoare**: necesare pentru traducerea codurilor sursă în instrucțiuni pe care procesorul să le înțeleagă și crearea unui program executabil sau a unor fișiere compilate. Există mai multe compilatoare pentru limbajele C/C++, cel mai utilizat fiind **gcc** (GNU C Compiler).
- **biblioteci** – fișiere antet (header) care conțin descrierea anumitor operații.

Orice fel de funcționalitate suplimentară poate fi introdusă prin instalarea de compilatoare.

Pentru a instala mediul de programare **Code::Blocks** parcurgeți următorii pași:

A. Descărcați fișierul **codeblocks-20.03mingw-setup.exe** care conţine kit-ul de instalare de la adresa: https://www.codeblocks.org/downloads/binaries/#imagesoswindows48pnglogo-microsoft-windows (figura B.1).

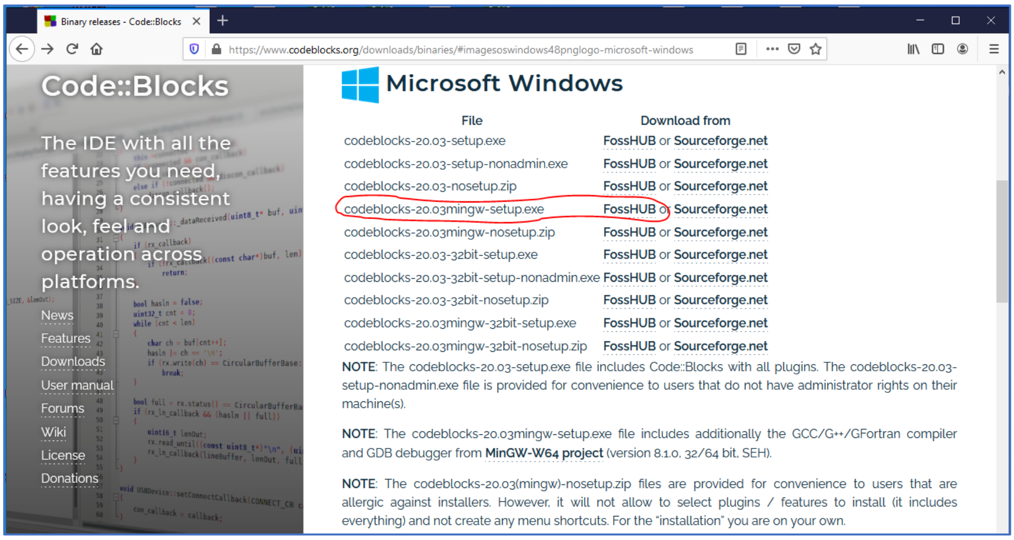


Figura B.1. Pagina de descărcare pentru Code::Blocks

B. Rulați fișierul descărcat. Se va deschide pe ecran fereastra din figura B.2. Apăsați **Next**.

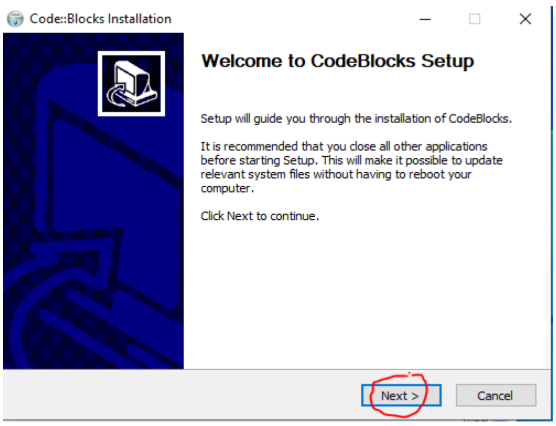


Figura B.2.

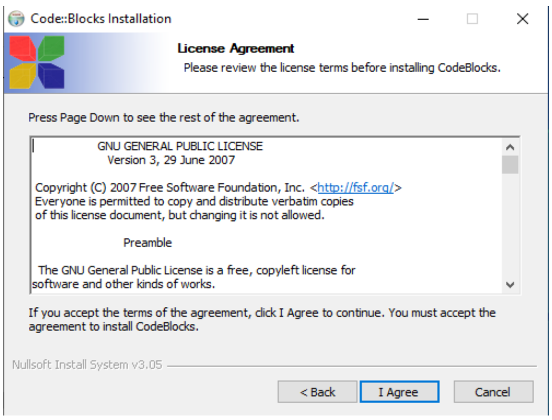


Figura B.3.

C. După completarea pasului anterior, este necesar să vă exprimați acordul în legătură cu termenii și condițiile de utilizare (figura B.3). Apăsați butonul **I Agree**.

D. Selectați componentele pe care doriți să le instalați (figura B.4).Apoi apăsați butonul **Next**.

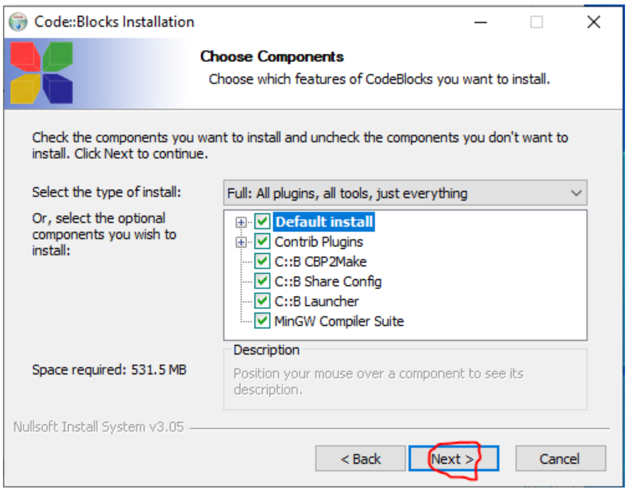


Figura B.4.

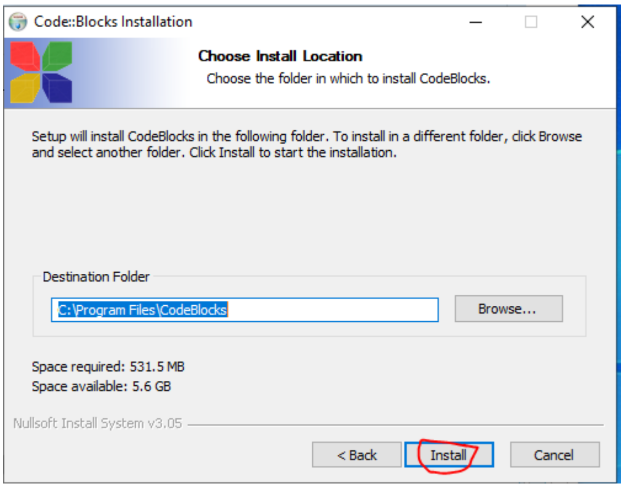


Figura B.5.

E. După selectarea componentelor dorite, alegeți locația unde doriți să se facă instalarea (figura B.5). Dacă doriți modificați directorul de destinație implicit. Apăsați butonul **Install**.

F. Spre finalul instalării, veți fi întrebați dacă doriți să rulați **Code::Blocks** (figura B.6). Alegeți **Yes**. Apoi apăsați **Finish** (figura B.7).

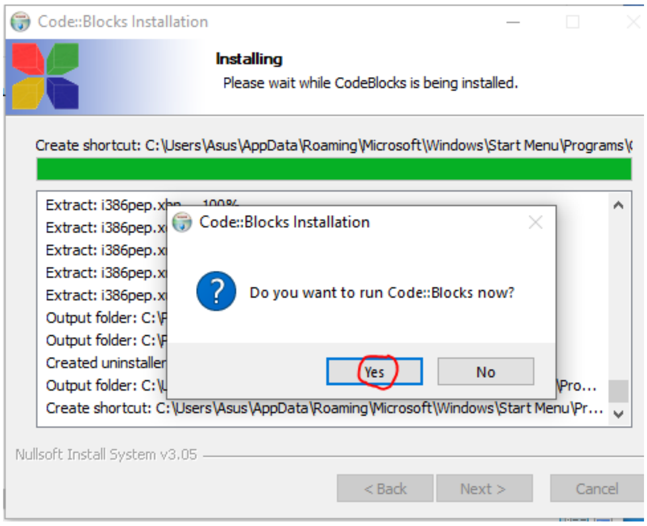


Figura B.6.

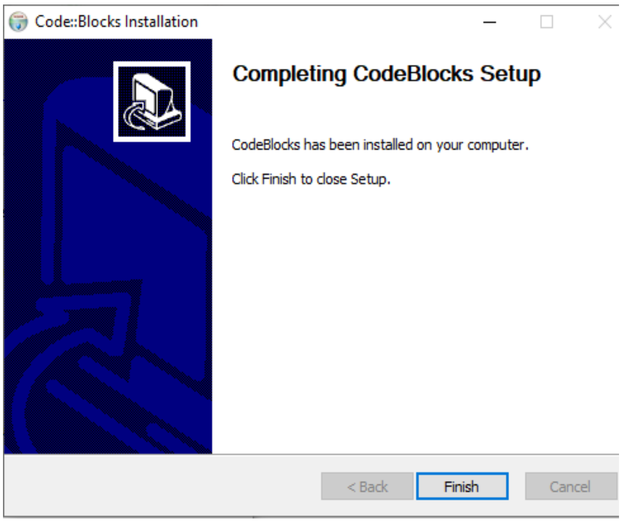
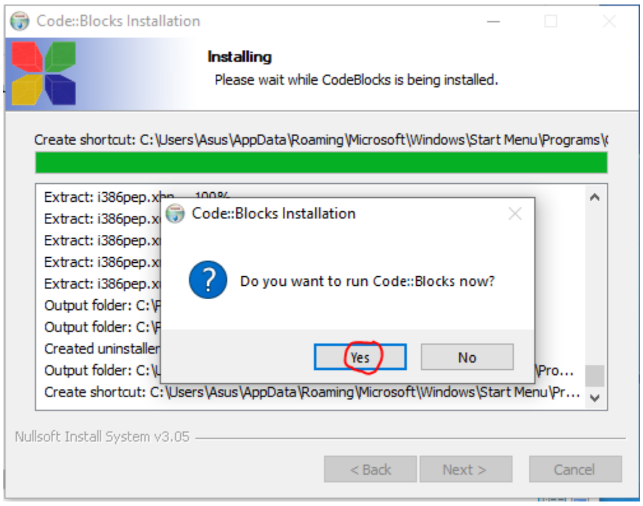


Figura B.7.

G. Code::Blocks se lansează în execuție și detectează automat compilatorul necesar pentru rularea programelor C/C++ (figura B.8). Apăsați **OK**.

H. Puteți alege asocierea programului Code::Blocks cu fișiere de tip C/C++, alegând a treia opțiune (***Yes, associate Code::Blocks with C/C++ file types***) prezentată în figura B.9. Apăsați **OK**.

I. Instalarea programului este finalizată. Interfața programului este prezentată în figura B.10.



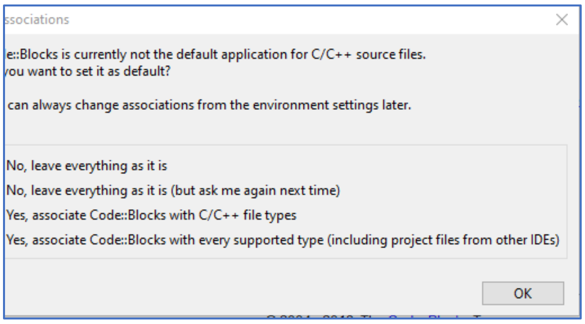


Figura B.8. **Figura B.9.**

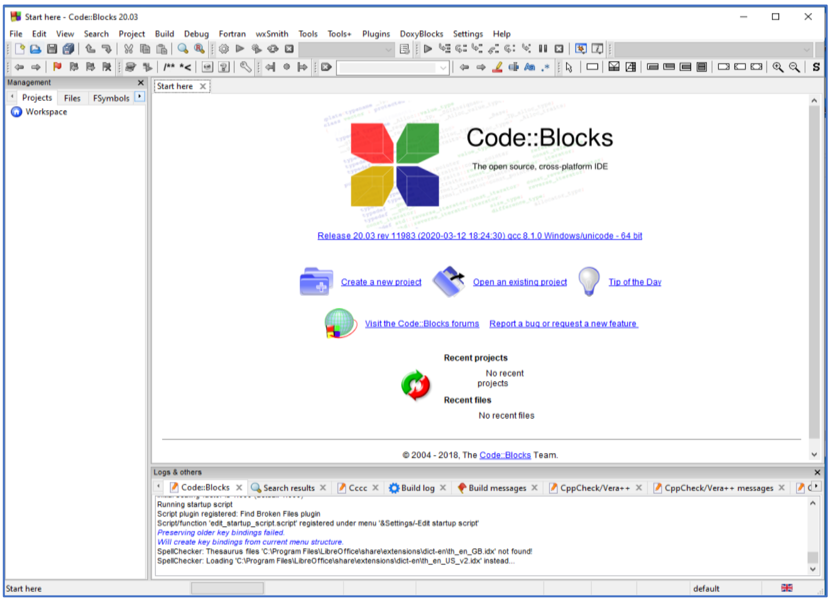


Figura B.10.