

1. Un tur de orizont în limbajul C.

1.1. Structura unui program C foarte simplu

Un *limbaj de programare* reprezintă o interfață între *problema de rezolvat* și *programul de rezolvare*. Limbajul de programare, prin specificarea unor acțiuni care trebuie executate *eficient* este *apropiat de mașină*. Pe de altă parte, el trebuie să fie *apropiat de problema de rezolvat*, astfel încât soluția problemei să fie exprimată *direct* și *concis*.

Trecerea de la specificarea problemei la program nu este directă, ci presupune parcurgerea mai multor *etape*:

- **analiza și abstractizarea problemei.** În această etapă se identifică *obiectele* implicate în rezolvare și acțiunile de transformare corespunzătoare. Ca rezultat al acestei etape se crează un *univers abstract al problemei* (UP), care evidențiază o mulțime de tipuri de obiecte, relațiile dintre acestea și restricțiile de prelucrare necesare rezolvării problemei.
- **găsirea metodei de rezolvare** acceptabile, precizând operatorii de prelucrare ai obiectelor din UP.
- **elaborarea algoritmului de rezolvare**
- **codificarea algoritmului**

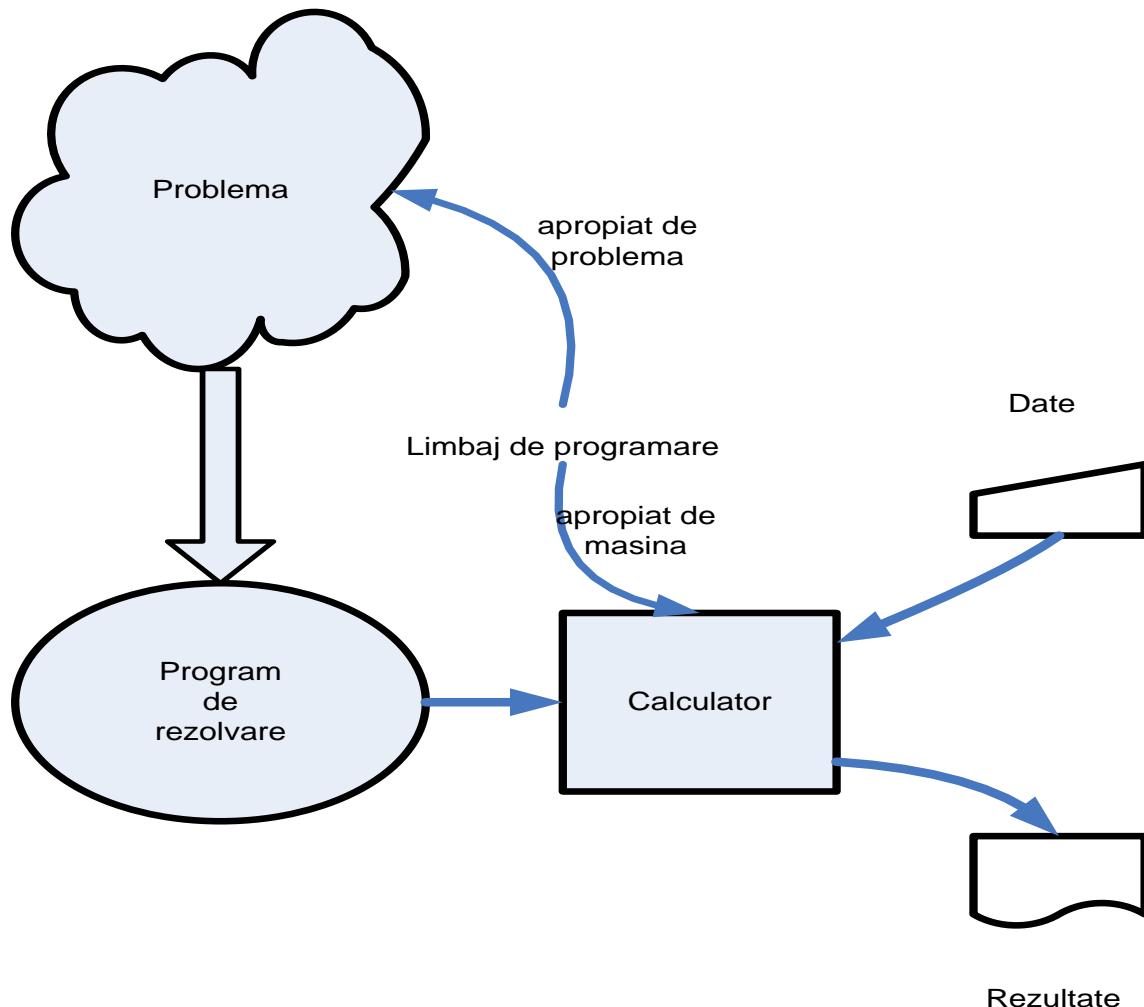


Fig.1.1. Rezolvarea unei probleme folosind calculatorul

Limbajul C s-a impus în elaborarea programelor datorită:

- ușurinței de reprezentare a obiectelor cu caracter nenumeric
- capacitatea de reprezentare a obiectelor dinamice
- capacitatea de exploatare a caracteristicilor mașinii de calcul pentru controlul strict al performanțelor programului
- asigurării unei interfețe transparente cu sistemul de operare al mașinii utilizate.

Limbajul C a fost creat de Dennis Ritchie și Brian Kernighan și implementat pe o mașină DEC PDP 11, cu intenția înlocuirii limbajului de asamblare.. Limbajul are precuitori direcți limbajele BCPL (Richards) și B (Thompson). Limbajul este folosit ca mediu de programare pentru sistemul de operare UNIX. Limbajul a fost standardizat în 1983 și 1989.

Limbajul C++ a fost dezvoltat de Bjarne Stroustrup pornind de la limbajul C, începând din anul 1980.

C++ împrumută din Simula 67 *conceptul de clasă* și din limbajul Algol 68 - *supraîncărcarea operatorilor*.

Dintre nouătile introduse de C++ menționăm: *moștenirea multiplă, funcțiile membre statice și funcțiile membre constante, sabloanele, tratarea excepțiilor, identificarea tipurilor la execuție, spațiile de nume, etc.*

Deși C++ este considerat o extensie a limbajului C, cele două limbaje se bazează pe *paradigme de programare* diferite. Limbajul C folosește *paradigma programării procedurale și structurate*. Conform acesteia, un program este privit ca o mulțime ierarhică de blocuri și proceduri (funcții). Limbajul C++ folosește *paradigma programării orientate pe obiecte*, potrivit căreia un program este constituit dintr-o mulțime de obiecte care interacționează.

Elementul constructiv al unui program C este *funcția*. Un program este constituit dintr-o mulțime de funcții, declarate pe un singur nivel (fără a se imbrica unele în altele), grupate în *module program*.

O *funcție* este o secțiune de program, identificată printr-un nume și parametrizată, construită folosind declarații, definiții și instrucțiuni de prelucrare. Atunci când este apelată, funcția calculează un anumit rezultat sau realizează un anumit efect.

Funcția **main()** este prezentă în orice program C. Execuția programului începe cu **main()**. Funcția **main()** poate întoarce un rezultat întreg (**int**) sau nici un rezultat (**void**). Numai în C este posibil să nu specificăm tipul rezultatului întors de funcție, acesta fiind considerat în mod implicit **int**.

```
/* program C pentru afisarea unui mesaj */
#include <stdio.h>
main(){
    printf("Acesta este primul program in C /n");
}
```

Programul folosește un *comentariu*, delimitat prin **/*** și ***/** care, prin explicații în limbaj natural, crește claritatea programului. Comentariul este constituit dintr-o linie sau mai multe linii, sau poate apărea în interiorul unei linii. Nu se pot include comentarii în interiorul altor comentarii.

Linia **#include <stdio.h>** anunță compilatorul că trebuie să insereze *fișierul antet stdio.h*. Acest fișier conține prototipurile unei serii de funcții de intrare și ieșire folosite de majoritatea programelor C. Fișierele antet au prin convenție extensia **.h**. Fișierul de inclus este căutat într-o zonă standard de includere, în care sunt memorate fișierele antet ale compilatorului C, dacă numele este încadrat între paranteze unghiulare (**<** și **>**), sau căutarea se face în zona curentă de lucru, dacă

fișierul este încadrat între ghilimele (""). Fișierele antet sunt foarte utile în cazul funcțiilor standard de bibliotecă; fiecare categorie de funcție standard are propriul fișier antet.

Valoarea întoarsă de funcția **main()** este în mod obișnuit 0, având semnificația că nu au fost întâlnite erori, și se asigură prin instrucțiunea **return 0**.

Instrucțiunea **printf()** servește pentru afișarea la terminal (pe ecran) a unor valori *formatate*.

Față de limbajul C, care este considerat un subset, limbajul C++ permite: *abstractizarea datelor, programarea orientată pe obiecte și programarea generică*.

1.2. Câteva elemente necesare scrierii unor programe C foarte simple.

1.2.1. Directiva define.

Directiva **#define nume text** este o *macrodefiniție*. Prelucrarea acesteia, numită *macroexpandare*, înlocuiește fiecare apariție a numelui prin textul asociat.

O aplicație o reprezintă creerea de *constante simbolice*. De exemplu:

```
#define PI      3.14159
#define mesaj   "Bonjour madame"
#define MAX     100
```

O constantă simbolică astfel definită nu poate fi redefinită prin atribuire.

1.2.2. Tipuri.

Fiecărui nume i se asociază un tip, care determină ce operații se pot aplica aceluia nume și cum sunt interpretate acestea. De exemplu:

```
char c='a'; /*c este variabilă caracter initializata cu 'a' */
int f(double); /*f functie de argument real cu rezultat intreg */
```

1.2.3. Definiții și declarații de variabile,

O valoare *constantă* se reprezintă textual (este un *literal*) sau printr-un nume - *constantă simbolică*.

O *variabilă* este un nume (identificator) care desemnează o locație de memorie în care se păstrează o valoare.

O variabilă se caracterizează astăzi prin: *nume (adresă), tip și valoare*, atributul valoare putând fi modificat. De exemplu:

```
int n, p;
char c;
float eps;
```

O variabilă poate fi *initializată* la declararea ei. De exemplu: **float eps=1.0e-6;**

Inițializarea se face numai o dată, înaintea execuției programului.

Variabilele *externe* și *static*e sunt inițializate implicit la 0.

Pentru o variabilă automatică (declarată în interiorul unui bloc), pentru care există inițializare explicită, aceasta este realizată la fiecare intrare în blocul care o conține.

O *definiție* este o construcție textuală care asociază unui nume o zonă de memorie (un obiect) și eventual inițializează conținutul zonei cu o valoare corespunzătoare tipului asociat numelui.

1.2.4. Atribuirea.

Atribuirea simplă este de forma: **variabilă = expresie** și are ca efect modificarea valorii unei variabile.

Atribuirea compusă a op= b reprezintă într-o formă compactă operația **a = a op b**

Atribuirea multiplă este de forma **variabilă1 = variabilă2 = ... = expresie** și inițializează variabilele, pornind de la dreapta spre stânga cu valoarea expresiei.

Operatorii de incrementare folosiți în atribuiră au efecte diferite. Astfel:

a = ++b este echivalentă cu **b=b+1; a=b;** în timp ce:
a = b++ are ca efect **a=b; b=b+1;**

1.2.5. Decizia.

Permite alegerea între două alternative, în funcție de valoarea (diferită de 0 sau 0) a unei expresii:

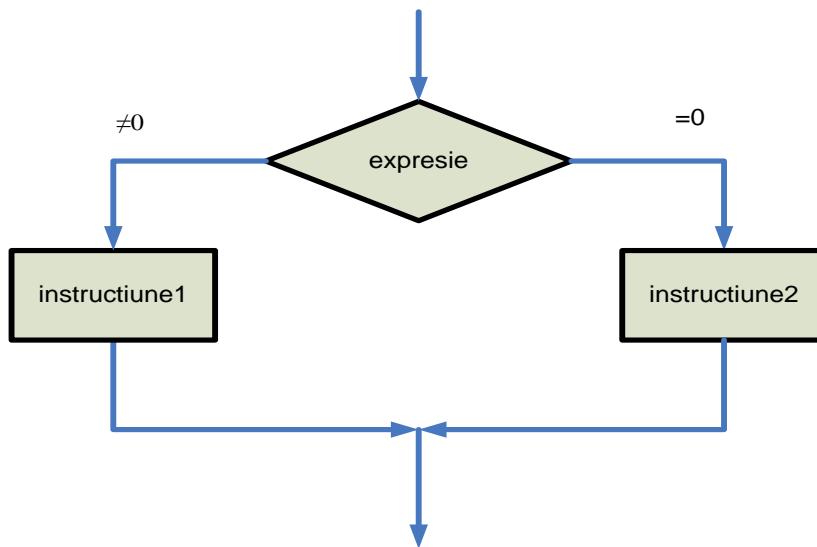


Fig.1.2. Structura de control decizie

```
if (expresie)
    instructiune1;
else
    instructiune2;
```

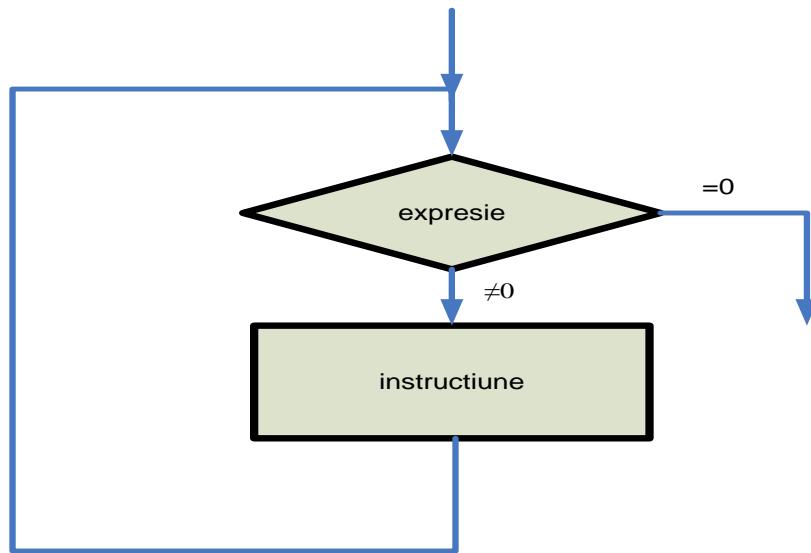
De exemplu:

```
if (a > b)
    max=a;
else
    max=b;
```

1.2.6. Ciclul.

Execută în mod repetat instrucțiunea, cât timp condiția este îndeplinită.

```
while (expresie)
    instructiune;
```

**Fig.1.3. Structura de control iteratie (ciclu)**

De exemplu calculul celui mai mare divizor comun se realizează cu:

```

while (a!=b)
    if (a > b)
        a -=b;
    else
        b -=a;
  
```

1.2.7. Afişarea valorii unei expresii (descriptori),

Pentru afişarea unei valori la terminal, sub controlul unui format se foloseşte funcţia:

```
printf(lista_descriptori, lista_expresii);
```

De exemplu: **printf("pi=% .5f\n", M_PI);**

1.2.8. Citirea valorilor de la terminal.

Pentru citirea unor valori introduse de la terminal, sub controlul unui format se foloseşte funcţia:

```
scanf(lista_descriptori, lista_adrese);
```

De exemplu: **scanf("%d%d/n", &a, &b);**

1.3. Structura unui program.

Un program în C este un ansamblu de funcţii şi variabile, între care există în mod obligatoriu o funcţie cu numele **main()** care se execută întotdeauna prima.

Cea mai simplă structură de program conţine o singură funcţie **main ()**:

```

int main() {
    <declaratii_locale>
    <instructiuni>
}
  
```

2. Elementele fundamentale ale limbajului C .

2.1.Alfabetul limbajului.

Conține setul de caractere ASCII (setul extins 256 caractere), în care un caracter este reprezentat printr-un octet, în care reprezentările caracterelor litere sunt contigue.

2.2. Atomi lexicali.

Există următoarele entități lexicale: identificatori, cuvinte cheie, constante, siruri de caractere, operatori și separatori.

Spațiile albe în C sunt reprezentate de: spațiu liber (blanc), tabular orizontală, tabulară verticală, linie nouă, pagină nouă, comentarii. Spațiile albe separă atomii lexicali vecini.

2.2.1. Identificatori.

Identificatorii servesc pentru numirea constantelor simbolice, variabilelor, tipurilor și funcțiilor.

Sunt formați dintr-o literă, urmată eventual de litere sau cifre. Caracterul de subliniere _ este considerat literă.

Intr-un identificator literele mari și cele mici sunt distințe. Astfel **Abc** și **abc** sunt identificatori diferenți.

Identificatorii pot avea orice lungime, dar numai primele 32 caractere sunt semnificative.

2.2.2. Cuvinte cheie.

Cuvintele cheie sau cuvintele rezervate nu pot fi folosite ca identificatori. Acestea sunt:

Tabel 2.1. Cuvinte cheie

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	if	int	long	register
return	short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void	volatile
while					

2.2.3. Literali.

Un literal este o reprezentare explicită a unei valori. Literalii pot fi de mai multe tipuri: întregi, caractere, reali, enumerări sau siruri de caractere..

2.2.4 Siruri de caractere.

Conțin caractere încadrate între ghilimele. În interiorul unui sir ghilimelele pot fi reprezentate prin \" . Un sir de caractere se poate întinde pe mai multe linii, cu condiția ca fiecare sfârșit de linie să fie precedat de \.

Sirurile de caractere în C se termină cu *caracterul nul '\\"0'*. În cazul literalelor siruri de caractere, compilatorul adaugă în mod automat la sfârșitul sirului caracterul nul.

2.2.5. Comentarii.

Un comentariu începe prin /* , se termină prin */ și se poate întinde pe mai multe linii. Comentariile nu pot fi incluse unele în altele (imbricate).

2.2.6. Terminatorul de instrucție.

Caracterul ; este folosit ca terminator pentru instrucțiuni și declarații, cu o singură excepție – după o instrucție compusă, terminată prin accoladă, nu se pune terminatorul ; .

2.2.7. Constante.

Constantele identifieri se obțin folosind directiva `#define` a preprocesorului:

```
#define constantă-identifier literal sau constantă-
identifier
#define constantă-identifier (expresie-constantă)
```

2.3. Ciclul de dezvoltare al unui program

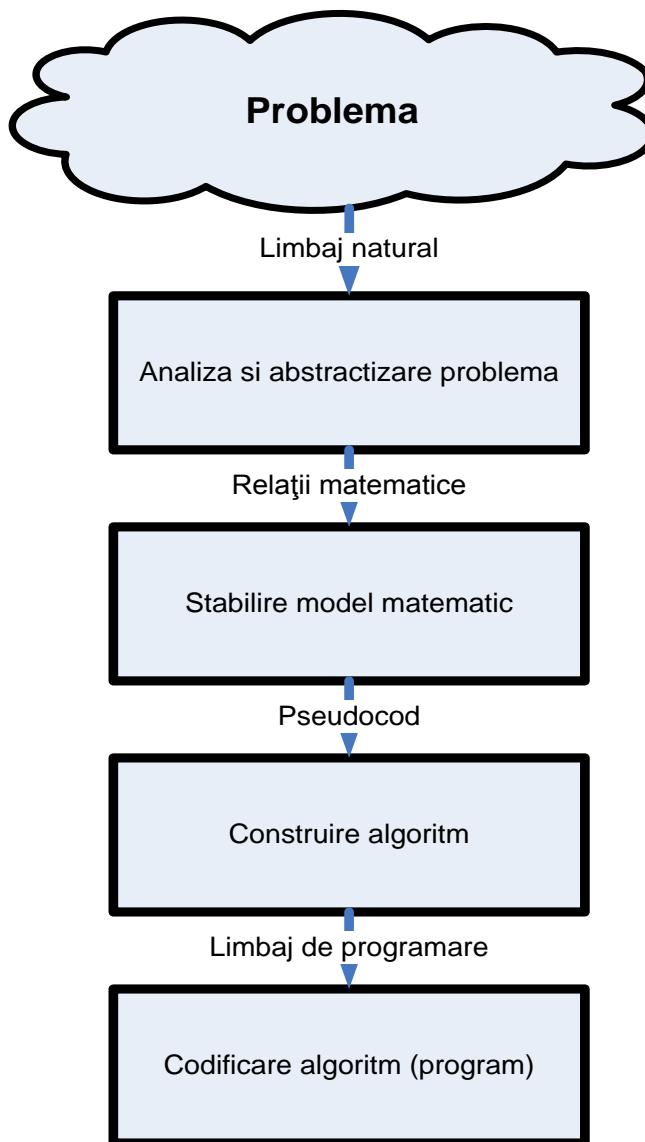


Fig.2.1. Trecerea de la problemă la programul de rezolvare

Conține următoarele etape:

1. Definirea problemei de rezolvat.

Analiza problemei cuprinde în afara formulării problemei în limbaj natural, o precizare riguroasă a intrărilor (datelor problemei) și a ieșirilor (rezultatelor).

De exemplu ne propunem să rezolvăm ecuația de gradul 2: $ax^2+bx+c=0$

Datele de intrare sunt cei 3 coeficienți **a**, **b**, **c** ai ecuației, care precizează o anumită ecuație de grad 2.

Rezultatele sunt cele două rădăcini (reale sau complexe) sau celelalte situații particulare care pot apărea.

2. Identificarea pașilor necesari pentru rezolvarea problemei începe cu formularea **modelului matematic**.

Ca model matematic vom folosi formula:

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

Formula nu este întotdeauna aplicabilă. Vom distinge următoarele situații:

1. **a=0**, caz în care nu avem de a face cu o ecuație de gradul 2, ci
 - 1.1. este posibil să avem ecuația de gradul 1: **bx+c=0**, cu soluția **x= - c/b**, dacă **b ≠ 0**.
 - 1.2. dacă și **b=0**, atunci
 - 1.2.1. dacă **c≠0**, atunci nu avem nici o soluție, în timp ce
 - 1.2.2. dacă și **c=0**, atunci avem o infinitate de soluții.
2. **a≠0** corespunde ecuației de gradul 2. În acest caz avem alte două situații:
 - 2.1. Formula este aplicabilă pentru rădăcini reale (discriminant pozitiv)
 - 2.2. Pentru discriminant negativ, încrucișat nu dispunem de aritmetică complexă, va trebui să efectuăm separat calculele pentru partea reală și cea imaginară.

3. Proiectarea algoritmului folosind ca instrument **pseudocodul**.

Pentru simplificare, în pseudocodul utilizat s-a evitat folosirea vreunei sintaxe. Ierarhizarea acțiunilor în structurile de control introduse a fost făcută numai prin indentare (aliniere). Acțiunile componente (subordonate) dintr-o structură de control au fost scrise decalat spre dreapta, în timp ce acțiunile independente au aceeași aliniere. Pseudocodul utilizat cuprinde:

1. operații de intrare / ieșire:

```
citește var1, var2,...
scrie expresie1, expresie2,...
```

2. structuri de control:

- 2.1. decizia:

```
dacă expresie atunci
    instructiune1
altfel
    instructiune2
```

- 2.2. ciclul:

```
cât timp expresie repetă
    instructiune
```

- 2.3. secvența:

```
instructiune1
    ...
instructiunen
```

Algoritmul dezvoltat pe baza acestui pseudocod este:

```
reali a,b,c,delta,x1,x2,xr,xi
citește a,b,c
dacă a=0 atunci
```

```

dacă b≠0 atunci
    scrie -c/b
altfel
    dacă c≠0 atunci
        scrie "nu avem nici o solutie"
    altfel
        scrie "o infinitate de solutii"
altfel
    delta=b*b-4*a*c
    dacă delta >= 0 atunci
        x1=(-b-sqrt(delta))/(2*a)
        x2=(-b+sqrt(delta))/(2*a)
        scrie x1,x2
    altfel
        xr=-b/(2*a)
        xi=sqrt(-delta)/(2*a)
        scrie xr,xi;

```

4. Scrierea programului folosind un limbaj de programare.

Vom codifica algoritmul descris mai sus folosind limbajul C:

```

#include <stdio.h>
#include <math.h>
int main()
{ double a,b,c,delta,x1,x2,xr,xi;
  scanf("%f %f %f",&a,&b,&c);
  if (a==0)
    if (b!=0)
      printf("o singura radacina x=%6.2f\n",-c/b);
    else
      if (c!=0)
        printf("nici o solutie\n");
      else
        printf("o infinitate de solutii\n");
  else
    { delta=b*b-4*a*c;
      if(delta >= 0)
        { x1=(-b-sqrt(delta))/2/a;
          x2=(-b+sqrt(delta))/2/a;
          printf("x1=%5.2f\tx2=%5.2f\n",x1,x2);
        }
      else
        { xr=-b/2/a;
          xi=sqrt(-delta)/2/a;
          printf("x1=%5.2f+i*%5.2f\nx2=%5.2f-i*%5.2f\n",
                 xr,xi,xr,xi);
        }
    }
}

```

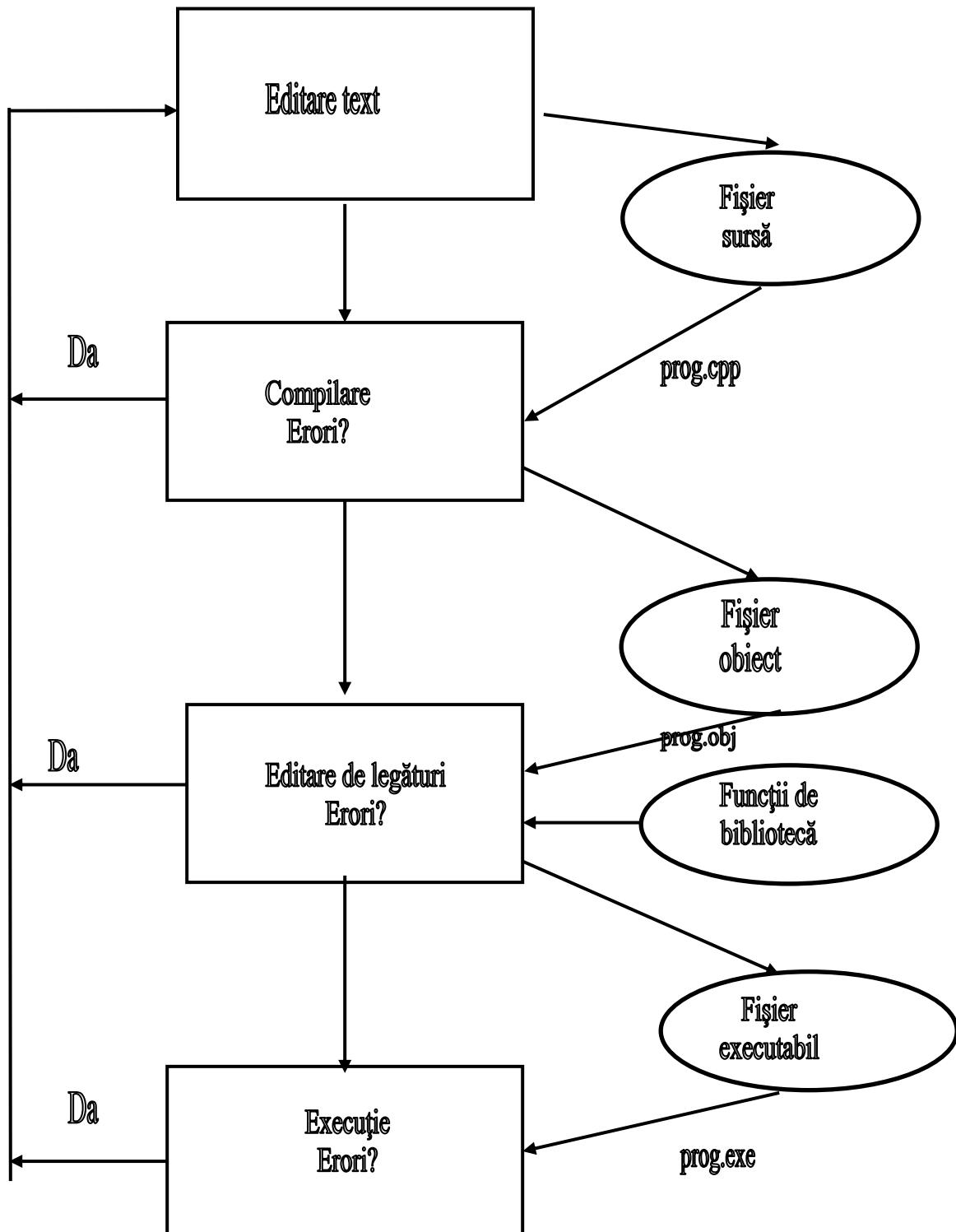



Fig.2.2. Etapele rezolvării unei probleme folosind calculatorul

5. Implementarea programului: editare, compilare, editare de legături, execuție.

Un mediu de programare C conține:

- I. *Editor de text* – folosit pentru creearea și modificarea codului sursă C
- II. *Compilator* – pentru a converti programul C în cod înțeles de calculator. Procesul de compilare cuprinde:
 - o fază de *precompilare (macroprocesare)* în care are loc *expandarea macrodefinițiilor si compilarea condițională si includerea fisierelor*
 - *compilarea propriu-zisă* în urma careia se generează cod obiect

Compilarea din linia de comandă în GCC se face cu:

```
gcc -c prog.c           prog.c → prog.o
```

- III. *Editor de legături* – leagă la codul obiect anumite *funcții de bibliotecă* (de exemplu intrări/ieșiri) extrase dintr-o bibliotecă de funcții, creindu-se un *program executabil*.

```
gcc -o prog prog.o       prog.o → prog.exe
```

Compilarea și editarea de legături se poate face printr-o singură comandă:

```
gcc -o prog prog.c       prog.c → prog.exe
```

Opoziția **-o** permite specificarea fișierului de ieșire. Dacă acesta lipsește, se consideră în mod implicit ca nume al fișierului executabil **a.out**. Așadar:

```
gcc prog.c               prog.c → a.out
```

- IV. *Fișiere biblioteci de funcții* – sunt fișiere de funcții gata compilate care se adaugă (se leagă) la program.

Există biblioteci pentru: funcții matematice, siruri de caractere, intrări/ieșiri.

Biblioteca standard C la execuție (run-time) este adăugată în mod automat la fiecare program; celelalte biblioteci trebuie legate în mod explicit.

De exemplu, în GNU C++ pentru a include biblioteca matematică **libm.so** se folosește opțiunea – **lm**:

```
gcc -o prog prog.c -lm
```

Bibliotecile pot fi:

- *static* (.lib în BorlandC, .a în GCC) – codul întregii biblioteci este atașat programului
- *dinamice* (.dll în BorlandC, .so în GCC) – programului își atașează numai funcțiile pe care acesta le solicită din bibliotecă.

În GCC se leagă în mod implicit versiunea dinamică a bibliotecii; pentru a lege versiunea statică se folosește opțiunea **-static**.

- V. *Fișiere antet (header files)* – datele și funcțiile conținute în biblioteci sunt declarate în fișiere antet asociate bibliotecilor. Prin includerea unui fișier antet compilatorul poate verifica corectitudinea apelurilor de funcții din biblioteca de funcții asociată (fără ca aceste funcții să fie disponibile în momentul compilării).

Tabel 2.1. Fișiere antet

Bibliotecă	Fișier antet
matematică	math.h
șiruri de caractere	string.h
intrări/ieșiri	stdio.h

Un fișier antet este inclus printr-o directivă cu sintaxa:

```
#include <nume.h>
```

care caută fișierul antet într-un director special de fișiere incluse (include), în timp ce directive:

```
#include "nume.h"
```

caută fișierul antet în directorul current.

Încărcarea și execuția programului (fișierului executabil .exe) se va face în GCC prin:

```
./ prog
```

6. Depanarea programului (debugging).

Depanarea unui program reprezintă localizarea și înlăturarea erorilor acestuia.

Cele mai frecvente *erori de sintaxă* se referă la: lipsa terminatorului de instrucțiune , neechilibrarea parantezelor, neînchiderea șirurilor de caractere, etc.

Erorile detectate de către *editorul de legături* sunt *referințele nerezolvate* (apelarea unor funcții care nu au fost definite, sau care se află în biblioteci care nu au fost incluse).

Cele mai des întâlnite erori la execuție provin din:

1. confuzia între = și ==
2. depășirea limitelor memoriei allocate tablourilor
3. variabile neinitializate
4. erori matematice: împărțire prin 0, depășire, radical dintr-un număr negativ, etc.

Dacă s-a depășit faza erorilor la execuție, vom testa programul, furnizându-i date de intrare pentru care cunoaștem ieșirile. Apariția unor neconcordanțe indică prezența unor *erori logice*. Dacă însă, rezultatele sunt corecte, nu avem certitudinea că programul funcționează corect în toate situațiile. Prin testare putem constata prezența erorilor, nu însă și absența lor.

Desfășurarea calculelor poate fi controlată *prin execuție pas cu pas*, sau prin asigurarea unor *puncte de intrerupere* în care să inspectăm starea programului, folosind în acest scop un *depanator* (debugger).

3. Tipuri și variabile.

3.1. Introducere

Mulțimile din matematică (**N**, **Z**, **R**, etc) sunt mulțimi infinite, cărora le sunt specifice anumite operații. De exemplu, cu elemente din **Z** se pot efectua operații precum: adunarea, scăderea, înmulțirea, împărțirea întreagă și restul împărțirii întregi. În cazul împărțirii există *restricția* ca împărțitorul să fie diferit de 0.

Tipurile de date ale limbajelor de programare se referă la mulțimi finite, cu operații și restricții specifice.

Un *tip de date* este precizat prin:

- o mulțime finită de *valori* corespunzătoare tipului (constantele tipului)
- o mulțime de *operatori* prin care se prelucrează valorile tipului
- o mulțime de *restricții* de utilizare a operatorilor.

De exemplu tipul întreg (**int**) este definit prin:

- mulțimea valorilor reprezentând numere întregi (între **-32768** și **32767**)
- mulțimea operatorilor : **+**, **-**, *****, **/**, **%**
- mulțimea restricțiilor: pentru operatorul / împărțitorul nu poate fi 0, etc.

Tipurile pot fi *tipuri fundamentale* și *tipuri derivate*.

3.2. Tipuri fundamentale.

Calculatoarele pot lucra în mod direct cu caractere, întregi și reali. Acestea sunt *tipuri fundamentale (predefinite)*.

Tipurile fundamentale (predefinite sau de bază sunt:

- tipul caracter (**char**)
- tipurile întregi (**int**, **short**, **long**)
- tipurile reale (**float**, **double**, **long double**)
- tipul vid (**void**)
- tipurile enumerate (**enum**)

Tipurile caracter, întreg, real și tipurile enumerate sunt *tipuri aritmetice*, deoarece valorile lor pot fi interpretate ca numere.

Tipurile caracter, întreg și enumerările sunt *tipuri întregi*.

În cele ce urmează, vom înțelege prin *obiect*, o zonă de memorie.

Declararea unui obiect specifică numai proprietățile obiectului, fără a aloca memorie pentru acesta.

Definirea unui obiect specifică proprietățile obiectului și alocă memorie pentru obiect.

Declararea obiectelor ne permite referirea la obiecte care vor fi definite ulterior în același fișier sau în alte fișiere care conțin părți ale programului.

3.2.1. Caracterele (tipul **char**)

Valorile asociate tipului caracter (**char**) sunt elemente din mulțimea caracterelor – setul de caractere ASCII. Drept consecință, caracterele pot fi tratate ca întregi, și invers.

Afișarea sau citirea unei variabile de tip **char** la terminal se face cu descriptorul de format **%c**.

Fiecarei constante caracter i se asociază o valoare întreagă, valoarea caracterului în setul de caractere ASCII. Pentru reprezentarea de caractere în alt set de caractere (de exemplu Unicode) se folosește tipul **wchar_t**.

Un literal caracter se reprezintă prin caracterul respectiv inclus între apostrofi (dacă este tipăribil).

Caracterele netipăribile se reprezintă prin mai multe caractere speciale numite *secvențe escape*.

Acestea sunt:

- \n** sfârșit de linie (LF)
- \t** tabulare orizontală (HT)
- \v** tabulare verticală (VT)
- \b** revenire la caracterul precedent (BS)
- \r** revenire la început de linie (CR)
- \f** avans la pagină nouă (FF)
- \a** alarmă (BEL)
- ** caracterul \
- \?** caracterul ?
- \'** caracterul '
- \"** caracterul "
- \ooo** caracterul cu codul octal **ooo**
- \xhh** caracterul cu codul hexazecimal **hh**

3.2.2. Întregii (tipul **int**).

Întregii pot avea 3 dimensiuni: **int**, **short int** (sau **short**) și **long int** (sau **long**).

Constantele intregi pot fi date în bazele:

```
10: 257, -65, +4928
 8: 0125, 0177
16: 0x1ac, 0XBF3
```

Afișarea unei variabile de tip **int** în baza 10 la terminal se face cu descriptorul de format **%d** sau **%i**, în baza 8 cu **%o**, iar în baza 16 cu **%x**.

Calificatorii long, short și unsigned

Calificatorul **long** situat înaintea tipului **int** extinde domeniul tipului întreg de la $(-2^{15}, 2^{15}-1)$ la $(-2^{31}, 2^{31}-1)$.

Constantele intregi lungi se scriu cu sufixul **L**: **125436L**.

Afișarea sau citirea unei variabile de tip **long int** la terminal se face cu descriptorul de format **%ld** sau **%li** în baza 10, **%lo** în baza 8 și **%lx** în baza 16.

Calificatorul **short** situat înaintea tipului **int** restrânge domeniul întregilor.

Afișarea sau citirea unei variabile de tip **short int** la terminal se face cu descriptorul de format **%hd** sau **%hi** în baza 10, **%ho** în baza 8 și **%hx** în baza 16.

Calificatorul **unsigned** înainte de **int** deplasează domeniul întregilor

$(-2^{15}, 2^{15}-1)$ la $(0, 2^{16}-1)$.

Afișarea sau citirea unei variabile de tip **unsigned int** la terminal se face cu descriptorul de format **%ud** sau **%ui** în baza 10, **%uo** în baza 8 și **%ux** în baza 16.

Constantele fără semn se specifică cu sufixul **U** sau **u**.

Există 6 tipuri întregi, formate folosind calificatorii:

```
{ [ signed | unsigned ] } { [ short | long ] } int
```

Avem următoarele echivalențe:

```
int = signed int
short = short int = signed short int
long = long int
unsigned = unsigned int
unsigned short = unsigned short int
unsigned long = unsigned long int
```

Literali întregi fi scriși în:

- *zecimal* - un sir de cifre zecimale, dintre care prima nu este 0.
- *octal* - un sir de cifre octale care începe cu cifra 0
- *hexazecimal* - un sir de cifre hexa care începe prin 0x.

Un număr negativ este precedat de semnul -. Există și semnul + unar.

Exemple: 125 01473 0x2AFC +645 -8359

Literalii întregi se pot termina prin **u** sau **U** (fără semn) sau **l** sau **L** (de tip lung).

Dacă constanta nu are sufix, atunci ea va apartine primului tip din succesiunea: **int**, **long int**, **unsigned long int** care permite reprezentarea valorii.

3.2.3. Realii (tipurile **float** și **double**).

Partea întreagă sau cea fracționară din constantă reală poate lipsi:

intreg.fractie sau **intreg.** sau **.fractie**

Exemple: 2.25, 1., -.5, +234.5

Constantele reale pot fi exprimate cu *mantisă* și *exponent* (notația științifică):

mantisaEexponent = **mantisa** 10^{exponent}

Exemple: 1.5e-3, 0.5E6.

Tipul **float** asigură o precizie de 7 cifre zecimale semnificative și exponentul maxim 38. Reprezentarea se face pe 4 octeți.

Afișarea unei variabile de tip **float** la terminal se face cu descriptorii de format **%f** sau **%e**.

Tipul **double** este foarte asemănător tipului **float**, cu deosebirea că se asigură o precizie de 16 cifre zecimale semnificative și exponentul maxim 306. Reprezentarea se face pe 8 octeți.

Afișarea sau citirea unei variabile de tip **double** la terminal se face cu descriptorul de format **%lf**, iar a unei variabile de tip **long double** cu **%Lf**. Reprezentarea internă pentru **long double** se face pe 10 octeți.

Numerele reale conțin punct zecimal și/sau exponent, având forma:

[<partea întreagă>] [<partea fractionară>] [E<exponent>]

Partea întreagă și partea fractionară pot lipsi, dar nu simultan. Punctul zecimal și exponentul sunt opționale, dar nu simultan.

Constanta poate avea un sufix:

- **f** sau **F** precizează o constantă de tip **float**

- **l** sau **L** precizează o constantă de tip **long double**.

Exemplu: **.25 -7.628 15E-3**

3.2.4. Definiri de tip cu **typedef**.

Un tip de date poate avea și un alt nume, prin folosirea declarației **typedef**. De exemplu:

```
typedef int intreg
```

Același efect se obține cu directiva **define**:

```
#define intreg int
```

3.2.5. Tipuri enumerate.

Folosirea tipului enumerare poate crește claritatea programului, încărcând numele să fie mai semnificative decât valorile care se ascund în spatele lor.

Astfel este mai naturală folosirea valorilor **ROSU**, **ALB**, **NEGRU**, **VERDE** pentru a desemna niște culori, decât a valorilor **0**, **1**, **2**, **3**.

Constantele simbolice pot fi introduse cu macrodefiniții **#define**.

```
#define FALSE 0
#define TRUE 1
```

Un *tip enumerat* folosește în locul valorilor tipului **0, 1, 2, ...** nume simbolice:

```
enum CULORI {ROSU, VERDE, GALBEN, ALBASTRU, NEGRU};
enum boolean {FALSE, TRUE};
```

Este posibil să forțăm pentru numele simbolice alte valori întregi decât **0, 1, 2, ...**

```
enum ZILE {LUNI=1, MARTI, MIERC, JOI, VIN, SAMB, DUM};
enum escapes
```

```
{BELL='\'a', BACKSPACE='\'b', TAB='\'t',
NEWLINE='\'n', VTAB='\'v', RETURN='\'r' };
```

Folosind **typedef** putem defini tipuri enumerative:

```
typedef enum {ROSU, GALBEN, ALBASTRU} culori;
typedef enum { lu, ma, mi, jo, vi, si, du } zile;
```

3.2.6. Tipul vid (**void**).

Tipul **void** precizează o mulțime vidă de valori. Aceasta se folosește pentru a specifica tipul unei funcții care nu întoarce nici un rezultat, sau un pointer generic.

3.3. Tipuri derivate.

Tipurile derivate sunt construite pornind de la tipurile fundamentale. Tipurile derivate sunt:

1. tablourile
2. funcțiile
3. pointerii
4. structurile (sau înregistrările)
5. uniunile (înregistrările cu variante)

Pe baza tipurilor fundamentale se pot construi tipuri derivate ca:

- tablouri de obiecte de un anumit tip

- funcții care întorc obiecte de un anumit tip
- pointeri care conțin adrese ale unor obiecte de un anumit tip
- structuri care conțin obiecte de tipuri diferite
- uniuni care conțin obiecte de tipuri diferite

3.4. Declararea variabilelor.

O variabilă constă din două componente: obiectul și numele obiectului. Numele pot fi identificatori sau expresii. Definirea sau declararea unei variabile are forma:

clasa-memorie tip declaratori;

Clasa de memorie poate fi omisă. Declaratorii sunt identificatori.

Fiecare declarator poate fi urmat de un *initializator*, care specifică valoarea inițială asociată identificatorului declarator.

Asupra claselor de memorie vom reveni mai târziu, așa că în exemplele curente le vom omite:

```
int i, j=0;
char c;
float x=1.5, y;
enum CULORI s1, s2=ROSU;
enum BOOLEAN p=TRUE;
culori c1, c2;
zile z, d;
```

3.5. Echivalența tipurilor.

Două tipuri se consideră echivalente în următoarele situații:

- echivalență structurală a tipurilor
- echivalență numelui tipului

Două obiecte sunt de tipuri structural echivalente, dacă au același tip de componente.

Două obiecte sunt de tipuri echivalente după nume, dacă au fost definite folosind același nume de tip.
Exemplu:

```
typedef int integer;
int x, y; /* x și y echivalente după numele tipului*/
integer z; /* x și z de tipuri structural echivalente */
```

4. Operatori și expresii.

Un *operator* este un simbol care arată ce operații se execută asupra unor operanzi (termeni).

Un *operand* este o constantă, o variabilă, un nume de funcție sau o subexpresie a cărei valoare este prelucrată direct de operator sau suportă în prealabil o conversie de tip.

Operatorii, după numărul de operanzi asupra cărora se aplică pot fi: *unari*, *binari* și *ternari*.

În C există 45 de operatori diferenți disponibili pe 15 *niveluri de prioritate*.

În funcție de tipul operanzilor asupra cărora se aplică, operatorii pot fi: aritmetici, relaționali, binari, logici, etc.

Operatorii sunt împărțiți în *clase de precedență* (sau de *prioritate*). În fiecare clasă de precedență este stabilită o *regulă de asociativitate*, care indică ordinea de aplicare a operatorilor din clasa respectivă: de la stânga la dreapta sau de la dreapta la stânga.

O expresie este o combinație de operanzi, separați între ei prin operatori; prin *evaluarea* unei expresii se obține o *valoare rezultat*. Tipul valorii rezultat depinde de tipul operanzilor și a operatorilor folosiți.

Evaluarea unei expresii poate avea *efekte laterale*, manifestate prin modificarea valorii unor variabile.

4.1. Conversii de tip.

Valorile pot fi convertite de la un tip la altul. Conversia poate fi implicită sau realizată în mod explicit de către programator.

4.1.1. Conversii implicite de tip.

Conversiile implicite au loc atunci când este necesar ca operatorii și argumentele funcțiilor să corespundă cu valorile așteptate pentru acestea.

Acestea pot fi sintetizate prin tabelul:

Tabel 4.1. Conversii implicite de tip

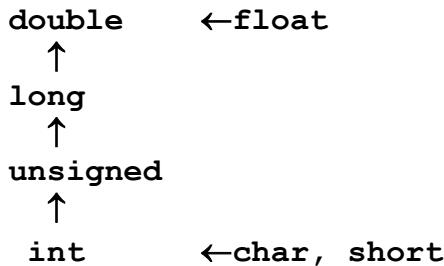
Tip	Tip la care se convertește implicit
char	int, short int, long int
int	char (cu trunchiere) short int (cu trunchiere) long int (cu extensia semnului)
short int	ca și int
long int	ca și int
float	double int, short int, long int
double	float int, short int, long int

4.1.2. Conversii aritmetice.

Când un operator binar se aplică între doi operanzi de tip diferit, are loc o *conversie implicită* a tipului unuia dintre ei, și anume, operandul de tip “mai restrâns” este convertit la tipul “mai larg” al celuilalt operand. Astfel în expresia **f + i**, operandul **int** este convertit în **float**.

Operatorii aritmetici convertesc automat operanzi la un anumit tip, dacă operanzi sunt de tip diferit. Se aplică următoarele reguli:

- operanții **char** și **short int** se convertesc în **int**; operanții **float** se convertesc în **double**.
- dacă unul din operanți este **double** restul operanților se convertesc în **double** iar rezultatul este tot **double**.
- dacă unul din operanți este **long** restul operanților se convertesc în **long**, iar rezultatul este tot **long**.
- dacă unul din operanți este **unsigned** restul operanților se convertesc în **unsigned**, iar rezultatul este tot **unsigned**.
- dacă nu se aplică ultimele 3 reguli, atunci operanții vor fi de tip **int** și rezultatul de asemenea de tip **int**.



Astfel **n = c - '0'** în care **c** reprezintă un caracter cifră calculează valoarea întreagă a acestui caracter.

Conversii implicate se produc și în cazul operației de atribuire, în sensul că valoarea din partea dreaptă este convertită la tipul variabilei acceptoare din stânga.

Astfel pentru declarațiile:

```

int i;
float f;
double d;
char c;

```

sunt permise atribuirile:

```

i=f; /* cu trunchierea partii fractionare */
f=i;
d=f;
f=d;
c=i;
i=c;

```

4.1.3. Conversiile de tip explicite (**cast**).

Conversiile explicite de tip (numite și cast) pot fi forțate în orice expresie folosind un operator unar (*cast*) într-o construcție de forma:

(tip) expresie

în care expresia este convertită la tipul numit.

Operatorul *cast* are aceeași precedență cu unui operator unar.

Astfel funcția **sqrt()** din biblioteca **<math.h>** cere un argument **double**, deci va fi apelată cu un cast: **sqrt((double) n)**.

Apelurile cu argumente de alt tip vor fi convertite în mod automat la tipul **double**: **x=sqrt(2)** va converti constanta **2** în **2.0**.

4.2. Operatorii aritmetici.

Operatorii aritmetici binari sunt: **+**, **-**, *****, **/** și **%** (modul = restul impărțirii întregi).

Prioritatea operatorilor aritmetici este:

+, -	unari
*, /, %	binari
+, -	binari

Regula de asociativitate este de la stânga la dreapta (la priorități egale operatorii sunt evaluați de la stânga la dreapta).

Tabel 4.2. Operatori multiplicativi

operator	descriere	tip operanzi	tip rezultat	precedență
*	înmulțire	aritmetic	int, unsigned, long, double	3
/	împărțire	aritmetic	int, unsigned, long, double	3
%	rest împărțire întreagă	întreg	int, unsigned, long	3

Tabel 4.3. Operatori aditivi

operator	descriere	tip operanzi	tip rezultat	precedență
+	adunare	aritmetici, pointer și întreg	int, unsigned, long double pointer	4
-	scădere	aritmetici, pointer și întreg doi pointeri	int, unsigned, long double pointer int	4

4.3. Operatorii de atribuire.

Operația de atribuire modifică valoarea asociată unei variabile (partea stângă) la valoarea unei expresii (partea dreaptă). Valoarea transmisă din partea dreaptă este convertită implicit la tipul părții stângi.

Atribuirile de formă: **a = a op b** se scriu mai compact **a op= b** în care **op=** poartă numele de *operator de atribuire*, **op** putând fi un operator aritmetic (**+, -, *, /, %**) sau binar (**>>, <<, &, ^, |**).

O *atribuire multiplă* are forma **v1=v2=...=vn=expresie** și este asociativă la dreapta.

O operație de atribuire terminată prin punct-virgulă (terminatorul de instrucțiune) se transformă într-o *instrucțiune de atribuire*.

4.4. Operatorii relaționali.

Operatorii relaționali sunt: **>**, **>=**, **<**, **<=**, care au toți aceeași prioritate (precedență).

Cu prioritate mai mică sunt: **==**, **!=**.

Operatorii relaționali au prioritate mai mică decât operatorii aritmetici. Putem deci scrie **a < b -1** în loc de **a < (b -1)**

Exemple: `car >= 'a' && car <= 'z'`

Tabel 4.4. Operatori relaționali

operator	descriere	tip operanzi	tip rezultat	precedență
<code><</code>	mai mic	aritmetic sau pointer	<code>int</code>	6
<code>></code>	mai mare	aritmetic sau pointer	<code>int</code>	6
<code><=</code>	mai mic sau egal	aritmetic sau pointer	<code>int</code>	6
<code>>=</code>	mai mare sau egal	aritmetic sau pointer	<code>int</code>	6
<code>==</code>	egal	aritmetic sau pointer	<code>int</code>	7
<code>!=</code>	neegal	aritmetic sau pointer	<code>int</code>	7

4.5. Operatorii booleeni.

Există următorii operatori logici:

- `!` - NEGATIE (operator unar)
- `&&` - ȘI logic (operatori binari)
- `||` - SAU logic

Exemple:

```
i<n-1 && (c=getchar()) != '\n' && c != EOF
```

nu necesită paranteze suplimentare deoarece operatorii logici sunt mai puțin prioritari decât cei relaționali.

```
bisect= an % 4 == 0 && an % 100 != 0 || an % 400 == 0;
estecifra= c >= '0' && c <= '9'
```

Condiția `x == 0` este echivalentă cu `!x`

`x != 0 && y != 0 && z != 0` este echivalentă cu `x && y && z`

Tabel 4.5. Operatori booleeni (logici)

operator	descriere	Tip operanzi	Tip rezultat	precedență	asociativitate
<code>!</code>	negație	aritmetic sau pointer	<code>int</code>	2	DS
<code>&&</code>	ȘI logic	aritmetic sau pointer	<code>int</code>	11	SD
<code> </code>	SAU logic	aritmetic sau pointer	<code>int</code>	12	SD

4.6. Operatorii binari (la nivel de biți).

În C există 6 operații de manipulare a bițiilor aplicate asupra unor operanzi întregi (`char`, `short`, `int`, `long`) cu sau fără semn:

- `&` - ȘI
- `|` - SAU inclusiv
- `^` - SAU exclusiv
- `<<` - deplasare stânga
- `>>` - deplasare dreapta
- `~` - complement fată de 1 (inversare)

Operatorul ȘI se foloseste pentru *operația de mascare* a unor biți.

```
n=n & 0177; /* pune pe 0 bitii din pozitia 8 in sus */
```

Operatorul SAU pune pe 1 biți specificați printr-o mască:

```
x=x | MASCA; /* pune pe 1 bitii care sunt 1 in MASCA */
```

Deplasarea dreapta a unui întreg cu semn este aritmetică, iar a unui întreg fără semn este logică. De exemplu:

```
x |= 1 << 7; /* pune pe 1 bitul 0 din octetul x */
x &= ~(1 << 7); /* pune pe 0 bitul 0 din octetul x */
```

Tabel 4.6. Operatori binari (pe biți)

operator	descriere	Tip operand	tip rezultat	precedență
<<	deplasare stânga	întreg	ca operandul stâng	5
>>	deplasare dreapta	întreg	ca operandul stâng	5
&	ȘI pe biți	întreg	int, long, unsigned	8
^	SAU exclusiv pe biți	întreg	int, long, unsigned	9
 	SAU inclusiv pe biți	întreg	int, long, unsigned	10

4.7. Operatorul condițional.

```
Decizia if (a > b)
        max = a;
    else
        max = b;
```

poate fi reprezentată prin expresia condițională:

```
max = a > b ? a : b
```

În general **ex1 ? ex2 : ex3** determină evaluarea **ex1**; dacă aceasta nu este 0 atunci valoarea expresiei conditionale devine **ex2**, altfel **ex3**.

4.8. Operatorul sevență.

Este reprezentat prin , și se foloseste în situațiile în care sintaxa impune prezența unei singure expresii, dar prelucrarea presupune prezența și evaluarea mai multor expresii.

Exemplu: **a < b ? (t=a, a=b, b=t) : a**

4.9. Operatori unari

a) Operatorul **sizeof**.

Aplicat asupra unei variabile furnizează numărul de octeți necesari stocării variabilei respective. Poate fi aplicat și asupra unui tip sau asupra tipului unei expresii:

```
sizeof variabila
sizeof tip
sizeof expresie
```

sizeof este un operator cu efect la compilare.

Tabel 4.7. Operatorul sizeof

operator	descriere	tip operand	tip rezultat	precedență
sizeof	necesar de memorie	variabilă sau tip	unsigned	2

b) Operatorii de incrementare /decrementare.**Tabel 4.8. Operatori de incrementare / decrementare**

operator	descriere	tip operand	tip rezultat	precedență
++	preincrementare	aritmetic sau pointer	int, long, double, unsigned, pointer	2
++	postincrementare	aritmetic sau pointer	la fel	2
--	predecrementare	aritmetic sau pointer	la fel	2
--	postdecrementare	aritmetic sau pointer	la fel	2

```
int a, b=5;
a = b++; /* a=5 */
a = ++b; /* a=7 */
```

c) Operatori de adresare indirectă / determinare adresă

& entitate - obține adresa unei entități,

* **pointer** - pentru adresare indirectă - adică memorează adresa unei entități printr-o valoare a unui pointer

Tabel 4.9. Operatori unari

operator	descriere	tip operand	tip rezultat	precedență
*	indirectare	pointer la T	T	2
&	adresare	T	pointer la T	2
~	negație	aritmetic	int, long, double	2
!	negație logică	aritmetic sau pointer	int	2

d) Operatori de acces :

- la elementele unui tablou
- la câmpurile unei structuri sau unei uniuni
- indirect prin intermediul unui pointer la câmpurile unei structuri sau unei uniuni

Operatorii de acces sunt:

- **[] indexare** folosit în expresii de forma **tablou[indice]**
- **. selecție directă** - pentru adresarea unui câmp dintr-o structură sau uniune sub forma: **struct.selector**
- → **selecție indirectă** - pentru accesul la un câmp dintr-o structură sau uniune, a cărei adresă este memorată într-un pointer
pointer -> selector este echivalent cu **(* pointer) . selector**

Toți acești operatori de acces, împreună cu operatorul de apel de funcție () au cea mai ridicată prioritate, și anume 1.

Tabel 4.10. Operatori de acces

operator	descriere	exemple	precedență
()	apel de funcție	<code>sqrt(x), printf("salut\n")</code>	1
[]	indexare tablou	<code>x[i], a[i][j]</code>	1
.	selector structură	<code>student.nastere.an</code>	1
->	selector indirect structură	<code>pstud->nume</code>	1

Tabel 4.11. Ordinea evaluării operanzilor.

precedență	operatori	simbol	asociativitate
1	apel funcție / selecție	() [] . ->	SD
2	unari	* & - ! ~ ++ -- sizeof	DS
3	multiplicativi	* / %	SD
4	aditivi	+ -	SD
5	deplasări	<< >>	SD
6	relaționali	< > <= >=	SD
7	egalitate / neegalitate	== !=	SD
8	ȘI pe biți	&	SD
9	SAU exclusiv pe biți	^	SD
10	SAU inclusiv pe biți		SD
11	ȘI logic	&&	SD
12	SAU logic		SD
13	condițional	? :	DS
14	atribuire	= op=	DS
15	virgula	,	SD

5. Instrucțiuni.

5.1. Instrucțiunea expresie.

O instrucțiune expresie se obține punând terminatorul de instrucțiune (punct-virgula) după o expresie:

expresie;

Exemple:

```
a++;
scanf(...);
max=a>b ? a : b;
```

Exemplul 1: Un număr real, introdus de la tastatură reprezintă măsura unui unghi exprimată în radiani. Să se scrie un program pentru conversia unghiului în grade, minute și secunde sexagesimale.

```
#include <stdio.h>
#define PI 3.14159265
int main(){
    float rad, gfr, mfr;
    int g, m, s;
    printf("Introduceti numarul de radiani: ");
    scanf("%f", &rad);
    g=gfr=rad*180/PI;
    m=mfr=(gfr-g)*60;
    s=(mfr-m)*60;
    printf("%5.2f radiani=%4d grade %02d min %02d sec\n",
           rad, g, m, s);
    return 0;
}
```

5.2. Instrucțiunea compusă (blocul).

Forma generală:

```
{
    declaratii_si_definitii;
    instructiuni;
}
```

Se folosește în situațiile în care sintaxa impune o singură instrucțiune, dar codificarea impune prezența unei secvențe de instrucțiuni. Blocul de instrucțiuni conține ca o singură instrucțiune.

5.3. Instrucțiunea vidă.

Forma generală: ;

Sintaxa impune prezența unei instrucțiuni, dar logica problemei nu necesită nici o prelucrare. În acest mod se introduc unele relaxări în sintaxă.

5.4. Instrucțiunea if.

Forma generală:

```
if (expresie)
```

```

    instructiune1;
else
    instructiune2;

```

Se evaluează expresia; dacă este diferită de 0 se execută **instructiune1** altfel **instructiune2**

O formă simplificată are instrucțiune2 vidă:

```

if (expresie)
    instructiune;

```

În problemele de clasificare se întâlnesc decizii de forma:

```

if (expr1)
    instr1;
else if (expr2)
    instr2;
...
else
    instrn;

```

De exemplu dorim să contorizăm caracterele citite pe categorii: litere mari, litere mici, cifre, linii și altele:

```

if (c == '\n')
    lini++ ;
else if (c>='a' && c<='z')
    lmici++ ;
else if (c>='A' && c<='Z')
    lmari++ ;
else if (c>='0' && c<='9')
    cifre++ ;
else
    altele++ ;

```

Exemplul 2 Să se scrie un program pentru rezolvarea cu discuție a ecuației de grad 2: $ax^2+bx+c=0$ folosind operatorul condițional.

```

#include <stdio.h>
#include <math.h>
int main(){
    float a, b, c, d;
    printf("Introduceti coeficientii ecuatiei: a,b,c\n");
    scanf("%f %f %f", &a,&b,&c);
    a? d=b*b-4*a*c, d>=0? printf("x1=%f\tx2=%f\n", (-b- sqrt(d))/2/a,
                                    (-b+sqrt(d))/2/a):
                                printf("x1=%f+i*%f\tx2=%f-i*%f\n", -b/2/a,
                                       sqrt(-d)/2/a, -b/2/a, sqrt(-d)/2/a)):
    b? printf("x=%f\n", -b/2/a): c? printf("0 solutii\n"):
                                printf("identitate\n");
    return 0;
}

```

Exemplul 3: Data curentă se exprimă prin **an**, **luna** și **zi**. Să se scrie un program care determină data zilei de mâine.

```
#include <stdio.h>
int bisect(int a){
    return a%4==0 && a%100!=0 || a%400==0;
}

int ultima(int a, int l){
    if (l==2)
        return (28+bisect(a));
    else if (l==4||l==6||l==9||l==11)
        return 30;
    else
        return 31;
}

int main()
{int a, l, z;
    printf("Introduceti data curenta: an,luna,zi\n");
    scanf("%d%d%d", &a, &l, &z);
    printf("azi: zi:%02d luna:%02d an:%4d\n", z, l, a);
    if (z < ultima(a,l))
        z++;
    else
        {z=1;
        if (l < 12)
            l++;
        else
            {l=1;
            a++;
            }
        }
    printf("mâine: zi:%02d luna:%02d, an:%4d\n", z, l, a);
    return 0;
}
```

5.5. Instrucțiunea **switch**.

Criteriul de selecție într-o problemă de clasificare îl poate constitui un selector care ia valori întregi. Forma generală:

```
switch (expresie){
    case val1: secventa1;
    case val2: secventa2;
    . . .
    default:   secventa s;
}
```

Se evaluează expresia selectoare; dacă valoarea ei este egală cu una din constantele cazurilor, se alege secvența de prelucrare corespunzătoare, după care se continuă cu secvențele de prelucrare ale cazurilor următoare.

Dacă valoarea expresiei selectoare nu este egală cu nici una din constantele cazurilor, se alege secvența corespunzătoare etichetei default.

Pentru ca prelucrările corespunzătoare cazurilor să fie disjuncte se termină fiecare secvență de prelucrare prin break. De exemplu:

```
y=x;
switch (n)
{ case 5: y*=x;
  case 4: y*=x;
  case 3: y*=x;
  case 2: y*=x;
}
```

calculează x^n , unde n ia valori de la 1 la 5.

Exemplul 4 Scrieti o functie pentru determinarea ultimei zile din lună.

```
int ultima(int a, int l)
{ switch (l) {
    case 1: case 3: case 5: case 7:
    case 8: case 10: case 12: return 31;
    case 4: case 6: case 9: case 11: return 30;
    case 2: return (28 + bisect(a));
}
}
```

5.6. Instrucțiunea while.

Este ciclul cu test inițial; se repetă instrucțiunea componentă cât timp expresia are valoarea adevărat (diferit de 0).

```
while (expresie)
  instructiune;
```

Exemplu 5: Copiați fișierul standard de intrare **stdin** la ieșirea standard **stdout**

```
/*copierea intrarii la iesire*/
{ int c;
  c = getchar();
  while (c != EOF)
  { putchar(c);
    c = getchar();
  }

/* varianta simplificata */
{ int c;
  while ((c=getchar()) != EOF)
    putchar(c);
}
```

Exemplul 6: Să se calculeze cel mai mare divizor comun și cel mai mic multiplu comun a 2 numere folosind algoritmul lui Euclid cu scăderi. (cât timp numerele diferă se înlocuiește cel mai mare dintre ele prin diferența lor).

```
#include <stdio.h>
int main(){
unsigned long a, b, ca, cb;
printf("Introduceti cele doua numere\n");
scanf("%lu %lu", &a, &b);
ca=a; cb=b;
while (a!=b)
    if(a > b)
        a-=b;
    else
        b-=a;
printf("cmmdc(%lu,%lu)=%lu\nncmmmc(%lu,%lu)=%lu\n",
       ca,cb,a,ca,cb,ca*cb/a);
return 0;
}
```

5.7. Instrucțiunea do...while.

Reprezintă ciclul cu test final; repetarea instrucțiunii are loc cât timp expresia este diferită de 0.

```
do
    instructiune;
while (expresie);
```

Corpul buclei este format dintr-o singură instrucțiune. Repetarea se face cel puțin o dată.

```
/* citirea unui raspuns */
{ char opt;
printf("Continuam ? D / N");
do
scanf("%c", &opt);
while (opt == 'D' || opt == 'd');
```

Exemplul 7: Să se stabilească dacă un număr este sau nu palindrom (are aceeași reprezentare citit de la stânga sau de la dreapta).

```
#include <stdio.h>
int main(){
unsigned long n, c, r=0;
scanf("%lu", &n);
c=n;
do{ r=10*r+n%10;
   n/=10;
}while (n);
printf("%lu %s este palindrom\n",c, (c==r)? ":" "nu");
return 0;
}
```

5.8. Instrucțiunea **for**.

Reprezintă o altă formă a ciclului cu test inițial.

```
for (exp_init; exp_test; exp_modif)
    instructiune;
```

este echivalentă cu:

```
exp_init;
while (exp_test) {
    instructiune;
    exp_modif;
}
```

Exemplul 8: *Să se stabilească dacă un număr întreg n este sau nu prim.*

Vom încerca toți divizorii posibili (de la 2 la \sqrt{n}). Dacă nu găsim nici un divizor, numărul este prim. Continuarea ciclului pentru testarea posibililor divizori este determinată de două condiții:

- să mai existe divizori netestați
 - candidații deja testați să nu fi fost divizori.
- La ieșirea din ciclu se determină motivul pentru care s-a părăsit ciclul:
- s-au testat toți candidații și nu s-a găsit nici un divizor, deci numărul este prim
 - un candidat a fost găsit divizor, deci numărul este neprim.
- Ciclul de testare a candidaților are forma:

```
for (d=2; d*d <= n && n % d != 0; d++)
    ;
```

Programul poate fi îmbunătățit prin evitarea testării candidaților pari (cu excepția lui 2).

```
#include <stdio.h>
int main() {
    unsigned long n, d;
    scanf("%lu", &n);
    for(d=2; d*d <= n && n%d; (d==2) ? d=3: d+=2)
        ;
    printf("numarul %lu este %sprim\n", n, !(n%d) ? "ne": "");
    return 0;
}
```

5.9. Instrucțiunea **continue**.

Plasarea acestei instrucțiuni în corpul unui ciclu are ca efect terminarea iterației curente și trecerea la iterația următoare: **continue**:

Exemplul 9: *O secvență de numere întregi este terminată prin 0. Să se calculeze suma termenilor pozitivi din secvență.*

```
#include <stdio.h>
int main() {
    int n, suma;
    for(suma=0, scanf("%d", &n); n; scanf("%d", &n)) {
        if(n < 0) continue;
        suma += n;
    }
}
```

```

printf("suma pozitivi = %d\n", suma);
return 0;
}

```

5.10. Instrucțiunea **break**.

Are ca efect ieșirea dintr-o instrucțiune de ciclare sau dintr-o instrucțiune **switch**, pentru a face alternativele disjuncte (în caz contrar dintr-o alternativă se trece în următoarea). Permite implementarea unor cicluri cu mai multe ieșiri plasate oriunde în interiorul ciclului.

O structură repetitivă foarte generală cu mai multe ieșiri plasate oriunde este:

```

while (1) {
    . . .
    if(expresie1) break;
    . . .
    if(expresieN) break;
    . . .
}

```

5.11. Instrucțiunea **goto**.

Realizează saltul la o etichetă. Este o instrucțiune nestructurată și se evită.

```
goto eticheta;
```

5.12. Instrucțiunea **return**.

Orice funcție nedeclarată **void** va trebui să întoarcă un rezultat. Tipul acestui rezultat este specificat în antetul funcției. Transmiterea acestui rezultat este realizată de o instrucțiune **return** inclusă în corpul funcției.

```
return expresie;
```

Exemplul 10: Calculul factorialului.

```

long factorial( int p)
{ int i;
  long f;
  for ( f = 1,i=2; i <= n; i++)
    f *= i;
  return f;
}

```

Dacă expresia întoarsă este de alt tip decât cel al funcției, atunci se face conversia la tipul funcției.

5.13. Probleme rezolvate.

1. Să se obțină reprezentarea ca fracție zecimală a numărului m/n . Eventuala perioadă se afișează între paranteze.

Reprezentarea zecimală a fracției ordinare se obține prin simularea împărțirii cifră cu cifră. În prealabil se simplifică fracția și se separă partea întreagă.

Lungimea părții neperiodice a fracției zecimale reprezintă maximul dintre multiplicitățile cifrelor 2 și 5 din descompunerea numitorului.

Dacă fracția zecimală are și parte periodică, atunci descompunerea numitorului conține și alți factori primi în afară de 2 și 5. O condiție mai simplă care stabilește dacă există parte periodică este ca restul parțial rămas după obținerea cifrelor din partea neperiodică să fie diferit de 0.

Operația de împărțire se încheie în momentul în care apare un rest parțial egal cu primul rest parțial din partea periodică. La citirea datelor (m și n) se asigură verificarea $n \neq 0$.

```

citire m si n si validare n != 0
simplificarea fractiei cu cmmdc
separarea partii intregi si afisarea ei
determinarea lungimii partii neperiodice
simularea impartirii pe lungimea partii neperiodice
if (exista parte periodica)
    salveaza primul rest parțial din partea periodică
    afisare paranteza deschisa pentru partea periodică
do
    calcul cifra din partea periodică și afisare
    obtinerea urmatorului rest parțial
while (restul parțial != primul rest parțial)
    afisare paranteza inchisa pentru partea periodică

```

În efectuarea împărțirii, o cifră a câtului se obține prin împărțirea întreagă a restului parțial cu numitorul.

Următorul rest parțial se obține ca restul împărțirii cu numitorul a restului parțial curent, completat în ultima poziție cu un zero (înmulțit cu 10). Primul rest parțial din partea neperiodică este numărătorul înmulțit cu 10.

Simularea împărțirii pe lungimea părții neperiodice se exprimă prin:

```

rest_partial = 10 * m
for (i = 1 ; i<=lungime_parte_neperiodica; i++)
    printf("%d", rest_partial / n);
    rest_partial = rest_partial % n * 10

```

Programul complet este:

```

#include <stdio.h>
#include <stdlib.h>

int main(){
    int m, n,          /* numarator si numitor fractie */
        rp,           /* restul parțial */
        lfn,           /* lungimea fractiei neperiodice */
        m2, m5,         /* multiplicatii 2 si 5 in numitor */
        c, d, r, i;

    /*citire date */
    scanf("%d%d", &m, &n);
    while (n==0)      /* fortare n!=0 */
        scanf("%d", &n);
    /* simplificarea fractiei cu cmmdc calculat cu algoritmul lui
       Euclid */
    c = m;
    d = n;

```

```

do {                                /* algoritmul lui Euclid */
    r = c % d;
    c = d;
    d = r;
} while(r);
m /= c;                            /* simplificare fractie */
n /= c;
/* separare parte intreaga */
printf("%6d / %6d = %6d.", m, n, m / n);
m %= n;
/* lungimea fractiei neperiodice */
m2 = 0; c = n;          /* multiplicitate 2 */
while (c % 2 == 0) {
    m2++;
    c /= 2;
}
m5 = 0; c = n;          /* multiplicitate 5 */
while (c % 5 == 0) {
    m5++;
    c /= 5;
}
/* lfn = max ( m2, m5 ) */
lfn = m2;
if(m5 > lfn)
    lfn = m5;
/* efectuarea impartirii pentru partea neperiodica */
rp = 10 * m;                  /* primul rest parțial */
for(i=0; i<lfn; i++) {
    printf("%1d", rp / n);   /* cifra din partea neperiodica */
    rp %= n * 10;           /* urmatorul rest parțial */
}
/* efectuarea impartirii pentru partea periodica */
if(rp){                      /* există parte periodica */
    printf("(");
    c = rp;                /* salvează primul rest parțial */
    do{
        printf("%1d", c / n); /* cifra din partea periodica */
        c %= n * 10;         /* urmatorul rest parțial */
    } while(c!=rp);
    printf(")");
}
printf("\n");
return 0;
}

```

5.14. Probleme propuse.

1. De pe mediul de intrare se citește un număr real **rad** reprezentând un unghi exprimat în radiani. Să se convertească în **grade**, **minute** și **secunde** centesimale.
2. Un maratonist pornește în cursă la un moment de timp exprimat prin **ora**, **minutul** și

secunda startului. Se cunoaște de asemenei timpul necesar sportivului pentru parcurgerea traseului. Să se determine momentul terminării cursei de către sportiv.

3. Să se stabilească codomeniul D al valorilor funcției:

$$f : [x_1, x_2] \rightarrow D, f(x) = a \cdot x^2 + b \cdot x + c, \quad a, b, c \in \mathbb{R}, a \neq 0.$$

Se cunosc **a**, **b**, **c**, **x1**, **x2**.

4. Cunoscând data curentă exprimată prin trei numere întregi reprezentând anul, luna, ziua precum și data nașterii unei persoane exprimată în același mod, să se calculeze vârsta persoanei exprimată în ani, luni și zile. Se consideră în mod simplificator că toate lunile au 30 de zile.

5. Un punct în plan este dat prin coordonatele lui **(x, y)**. Să se stabilească poziția lui prin indicarea cadranului (1, 2, 3 sau 4) în care este plasat. Pentru un punct situat pe una din semiaxe se vor preciza cadranele separate de semiaxa respectivă (de exemplu 2-3).

6. Se citesc trei numere reale pozitive ordonate crescător. Să se verifice dacă acestea pot să reprezinte laturile unui triunghi și în caz afirmativ să se stabilească natura triunghiului: isoscel, echilateral, dreptunghic sau oarecare și să se calculeze aria sa.

7. Cunoscând data curentă și data nașterii unei persoane exprimată fiecare sub forma unui triplet **(an, luna, zi)** să se afle vârsta persoanei în ani impliniți.

8. De pe mediul de intrare se citește un unghi exprimat în **grade, minute, secunde**. Să se convertească în radiani.

9. Un număr întreg **S** reprezintă o durată de timp exprimată în secunde. Să se convertească în **zile, ore, minute și secunde** utilizând în program cât mai puține variabile.

10. Trei valori reale sunt citite în variabilele **a**, **b**, **c**. Să se facă schimbările necesare astfel încât valorile din **a**, **b**, **c** să apară în ordine crescătoare.

11. Să se scrie algoritmul pentru rezolvarea cu discuție a ecuației de gradul 1: **a*x+b=0**, cu valorile lui **a** și **b** citite de pe mediul de intrare.

12. Să se scrie algoritmul pentru rezolvarea cu discuție a ecuației de gradul 2: **a*x^2 + b*x + c = 0**. Se dau pe mediul de intrare coeficienții **a**, **b**, **c** care pot avea orice valori reale reprezentabile în memoria calculatorului.

13. Se consideră sistemul de ecuații:

$$\begin{aligned} a \cdot x + b \cdot y &= c \\ m \cdot x + n \cdot y &= p \end{aligned}$$

dat prin valorile coeficienților **a**, **b**, **c**, **m**, **n**, **p**. Să se rezolve sistemul cu discuție.

14. Să se scrie algoritmul pentru "casierul automat" care citește de pe mediul de intrare suma (intreagă) datorată de un client și calculează "restul" pe care acesta îl primește în număr minim de bancnote și monezi de **100000, 50000, 10000, 5000, 1000, 500, 100, 50, 25,**

10, 5, 3 și 1 leu considerând că suma plătită este cel mai mic multiplu de **100000** mai mare decât suma datorată.

15. Să se calculeze data revenirii pe pământ a unei rachete, exprimată prin **an, lună, zi, oră, minut, secundă**, cunoscând momentul lansării exprimat în același mod și durata de zbor exprimată în secunde.

16. Un număr perfect este un număr egal cu suma divizorilor săi, printre care este considerată valoarea 1 dar nu și numărul.

Să se găsească toate numerele perfecte mai mici sau egale cu un număr **k** dat pe mediul de intrare, și să se afișeze fiecare număr astfel determinat, urmat de suma divizorilor lui. De exemplu numărul **6** are divizorii **1, 2, 3, 6**. El este număr perfect deoarece: **6 = 1 + 2 + 3**.

17. Dându-se trei numere întregi reprezentând data unei zile (**an, lună, zi**), să se stabilească a câțiva din an este aceasta.

18. Se dau pe mediul de intrare un număr necunoscut de numere nenule terminate cu o valoare nulă. Să se stabilească dacă acestea:

- formează un sir strict crescător;
- formează un sir crescător;
- formează un sir strict descrescător;
- formează un sir descrescător;
- sunt identice;
- nu sunt ordonate.

19. Dându-se un număr întreg **n**, să se afișeze toți factorii primi ai acestuia precum și ordinea lor de multiplicitate.

20. Abaterea medie pătratică a rezultatelor obținute prin determinări experimentale se poate calcula cu formula:

$$\text{sigma} = \sqrt{\frac{N \sum_{i=1}^N x_i^2 - \left(\sum_{i=1}^N x_i \right)^2}{N(N - 1)}}$$

aplicabilă numai dacă s-au făcut cel puțin 2 măsurători.

Dându-se pe mediul de intrare **N** (**N≤25**) și rezultatele celor **N** determinări să se calculeze abaterea medie pătratică.

21. Să se calculeze **s = $\sum_{k=1}^n k!$** când se cunoaște **n**

22. Să se scrie algoritmul pentru rezolvarea a **n** ecuații de gradul 2. Se citesc de pe mediul de intrare valoarea lui **n** și **n** tripleți (**a, b, c**) reprezentând coeficienții ecuațiilor.

Se recomandă realizarea unui program care să utilizeze cât mai puține variabile.

23. Dându-se notele obținute de o grupă de n studenți la o disciplină, să se stabilească câți dintre ei au promovat (nu se vor utiliza decât variabile simple).

24. Se dau pe mediul de intrare notele obținute de către studenții unei grupe la un examen, precedate de numărul studenților. Să se determine dacă grupa este sau nu integralistă, precum și procentajul de note foarte bune (8..10).

25. Să se determine cel mai mare (**max**) precum și cel mai mic (**min**) element dintr-un sir a_0, a_1, \dots, a_{n-1} .

Se dau pe mediul de intrare n precum și cele n elemente ale sirului, care sunt citite pe rând într-o aceeași variabilă **a**.

27. De pe mediul de intrare se citește un număr real **b** și un sir de valori reale pozitive terminate printr-o valoare negativă (care nu face parte din sir).

Să se stabilească elementul din sir cel mai apropiat de **b**. Se va preciza și poziția acestuia. Elementele sirului vor fi păstrate pe rând în aceeași variabilă **a**.

28. Să se calculeze x^n pentru **x** (real) și **n** (intreg) dați, folosind un număr cât mai mic de înmulțiri de numere reale.

29. De pe mediul de intrare se citesc **n** valori întregi pozitive. Pentru fiecare element să se indice cel mai mare patrat perfect mai mic sau egal cu el.

30. De pe mediul de intrare se citește o listă de numere întregi pozitive terminate cu un număr negativ ce marchează sfârșitul listei. Să se scrie în dreptul fiecărei valori numărul prim cel mai apropiat mai mic sau egal cu numărul dat.

31. Să se rezolve ecuația $f(x) = 0$ cu precizia **epsilon** dată, știind că are o rădăcină în intervalul $\{a, b\}$ precizat. Se va utiliza metoda înjumătăririi intervalului ("bisecție").

Indicație: Se împarte intervalul $\{a, b\}$ în două jumătăți egale; fie **m** mijlocul intervalului. Dacă la capetele intervalului $\{a, m\}$ funcția are semne contrare soluția se va căuta în acest interval, altfel se va considera intervalul $\{m, b\}$. Se consideră determinată soluția dacă mărimea intervalului a devenit inferioară lui **epsilon** sau valoarea $|f(m)| < \text{epsilon}$.

32. Dându-se un număr întreg **n** să se afle cifrele reprezentării sale în baza **10** începând cu cifra cea mai semnificativă.

33. Să se afle cifrele reprezentării în baza **b** (începând cu cea mai semnificativă) a unui număr **a** dat în baza **10**.

Indicație:

$$a = c_n b^n + c_{n-1} b^{n-1} + \dots + c_0$$

$$c_n = a / b^n$$

$$a \% b^n = c_{n-1} b^{n-1} + \dots + c_0$$

34. Dându-se numărul real $a \quad (0 < a < 1)$ să se determine primele n cifre ale reprezentării lui într-o bază b dată.

35. De pe mediul de intrare se citesc n cifre constituind reprezentarea unui număr într-o bază $b_1 < 10$, începând cu cea mai puțin semnificativă. Să se obțină și să se afișeze cifrele reprezentării aceluiași număr într-o altă bază $b_2 < 10$.

36. Să se verifice dacă un numar întreg citit de pe mediul de intrare este palindrom, adică se citește la fel de la stânga la dreapta și de la dreapta la stânga (numărul este identic cu răsturnatul său). Un astfel de număr este **4517154**. Nu se vor folosi tablouri de variabile pentru păstrarea cifrelor numărului.

37. Să se determine toate numerele prime mai mici sau egale cu un număr k dat pe mediul de intrare. Pentru a verifica dacă un număr x este prim se va incerca divizibilitatea lui cu $2, 3, 4, \dots, \lfloor \sqrt{x} \rfloor$. Numărul este prim dacă nu se divide cu niciunul dintre aceste numere și este neprim dacă are cel puțin un divizor printre ele.

38. Printre numerele mai mici sau egale cu un număr n dat pe mediul de intrare să se găsească cel care are cei mai mulți divizori.

39. Se consideră funcția $f(x) = \ln(2 \cdot x^2 + 1)$. Să se scrie un program pentru tabelarea pe intervalul $[-10, 10]$ cu următorii pași:

$$\begin{aligned} 0.1 &\text{ pentru } |x| \leq 1.0 \\ 0.5 &\text{ pentru } 1.0 < |x| \leq 5.0 \\ 1.0 &\text{ pentru } 5.0 < |x| \leq 10.0 \end{aligned}$$

40. Se dă un număr întreg pozitiv reprezentat în baza 10. Să se determine și să se afișeze cifrele reprezentării sale în baza 16. Se precizează că în baza 16 cifrele utilizate sunt $0, \dots, 9, A, B, C, D, E, F$. Prin convenție la sfârșitul numerelor reprezentate în baza 16 (hexazecimal) se scrie litera H. Dacă cifra cea mai semnificativă a reprezentării sale este A..F atunci se va afișa ca primă cifră un 0. Exemplu:

$$\begin{array}{ccc} 175 & = & 0AFH \\ (10) & & (16) \end{array}$$

$$0AFH = 10 * 16^1 + 15 * 16^0$$

41. De pe mediul de intrare se citesc cifrele reprezentării unui număr întreg în baza 16 terminate cu caracterul H (cifrele hexazecimale sunt $0, \dots, 9, A, B, C, D, E, F$). Să se calculeze și să se afișeze reprezentarea numărului în baza 10.

42. Dându-se un număr întreg n să se afișeze reprezentarea sa cu cifre romane impunând regula ca o cifră să nu poată fi urmată de o alta cu valoare strict mai mare decât ea. Numărul 99 se va reprezenta în aceste condiții ca **LXXXXVIIII** și nu ca **XCIX**.

43. Se dau două numere întregi, primul reprezentând un an și al doilea, numărul de zile scurse din

acel an. Să se determine data (luna și ziua).

44. Să se calculeze coeficienții binomiali $C_n^1, C_n^2, \dots, C_n^p$ în care n și p sunt intregi pozitivi dați ($p \leq n$), știind că există următoarea relație de recurență:

$$C_n^k = (n-k+1) / k * C_n^{k-1} \quad \text{pornind cu } C_n^0 = 1$$

45. Se consideră polinomul: $p_n(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_n$

Să se calculeze valoarea polinomului într-un punct x dat, dacă valorile coeficienților lui x se citesc pe rând, în aceeași variabilă a :

- a) în ordinea descrescătoare a puterilor lui x (adică în ordinea a_0, a_1, \dots, a_n)
- b) în ordinea crescătoare a puterilor lui x , (adică în ordinea a_n, a_{n-1}, \dots, a_0)

46. Pentru a, b și n dați ($a, b \in \mathbb{R}, n \in \mathbb{Z}$) să se calculeze x și y astfel ca: $x+i*y=(a+i*b)^n$ fără a folosi formula lui Moivre

47. Să se calculeze și să se afișeze valorile integralei:

$$I_k(x) = \int_0^x u^k e^u du$$

pentru $k=1, 2, \dots, n$, în care n și x sunt dați, cunoscând că:

$$I_k(x) = [x^k - A_k^1 x^{k-1} + A_k^2 x^{k-2} - \dots - (-1)^k A_k^k] e^x \quad \text{unde :}$$

$$A_k^p = k(k-1) \dots (k-p+1)$$

48. Să se calculeze pentru n dat, f_n termenul de rangul n din sirul lui Fibonacci, cunoscând relația de recurență:

$$f_p = f_{p-1} + f_{p-2} \quad \text{pentru } p > 2 \text{ și } f_0 = 1, f_1 = 1$$

49. Sirul $\{x_n\}$ generat cu relația de recurență $x_n = (x_{n-1} + a/x_{n-1})/2$ pornind cu $x_0 = a/2$ este convergent pentru $a > 0$ și are ca limită \sqrt{a} . Pentru a oarecare, dat, să se construiască un algoritm care calculează \sqrt{a} ca limită a acestui sir, cu o precizie eps dată.

50. Sirurile $\{u_n\}$ și $\{v_n\}$ generate cu relațiile de recurență:

$u_n = (u_{n-1} + v_{n-1})/2$ și $v_n = \sqrt{u_{n-1}v_{n-1}}$ pornind cu $u_0 = 1/|a|$, $v_0 = 1/|b|$, unde $a \neq 0$, $b \neq 0$ au o aceeași limită comună, valoarea integralei eliptice:

$$I = \frac{2}{\pi} \int_0^{\pi/2} \frac{dx}{\sqrt{a^2 \cos^2 x + b^2 \sin^2 x}}$$

Să se calculeze această integrală pentru a și b dați, ca limită comună a celor două siruri, determinată aproximativ cu precizia eps , în momentul în care distanța între termenii celor două siruri devine inferioară lui eps , adică $|u_n - v_n| < eps$.

51. Pentru calculul lui $\lg_2 x$ se generează sirurile $\{a_n\}$, $\{b_n\}$ și $\{c_n\}$

$$a_n = \begin{cases} a_{n-1}^2 & \text{daca } a_{n-1}^2 < 2 \\ \frac{a_{n-1}^2}{2} & \text{daca } a_{n-1}^2 \geq 2 \end{cases} \quad \text{pornind cu } a_0 = x$$

Valeriu Iorga

Programare în C

cu relațiile de recurență:

$$a_n = \begin{cases} a_{n-1}^2 & \text{dacă } a_{n-1}^2 < 2 \\ \frac{a_{n-1}^2}{2} & \text{dacă } a_{n-1}^2 \geq 2 \end{cases} \quad \text{pornind cu } a_0 = x$$

$$b_n = b_{n-1} / 2 \quad \text{pornind cu } b_0=1$$

$$c_n = \begin{cases} c_{n-1} & \text{dacă } a_{n-1}^2 < 2 \\ c_{n-1} + b_n & \text{dacă } a_{n-1}^2 \geq 2 \end{cases} \quad \text{pornind cu } c_0 = 0$$

Se știe că pentru $1 < x < 2$, $\lim c_n = \lg_2 x$.

Dacă $x \notin (1, 2)$ se aduce argumentul în acest interval folosind relațiile:

$$\lg_2 x = -\lg_2 (1/x) \quad \text{pentru } x < 1$$

$$\lg_2 x = k + \lg_2 (x/2^k) \quad \text{pentru } x \geq 2^k$$

52. Dezvoltarea în serie:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

este rapid convergentă pentru x mic. Pentru x oarecare, acesta se descompune sub forma:

$x = i + f$ in care:

i = partea întreagă a lui x

f = partea fracționară a lui x .

Rezultă $e^x = e^i * e^f$ cu:

$$e^i = \underbrace{e \cdot e \cdots e}_i \quad \text{pentru } i > 0$$

$$e^i = \underbrace{\frac{1}{e} \cdot \frac{1}{e} \cdots \frac{1}{e}}_i \quad \text{pentru } i < 0$$

Pentru x dat, să se calculeze e^x cu o precizie **eps** dată.

53. Să se obțină reprezentarea ca fracție zecimală a numărului m/n . Eventuala perioadă se afișează între paranteze.

54. Să se determine valoarea n pentru care:

$$s = \sum_{k=1}^n \frac{2}{\sqrt{4n^2 - k}}$$

satisfacă condiția $|s - \pi/3| < \varepsilon$, în care **eps** este dat. Se știe că

$$\lim_{k \rightarrow \infty} S = \frac{\pi}{3}$$

55. Să se calculeze funcția Bessel de speță I-a $J_n(x)$ știind că există relația de recurență:

$$J_p(x) = (2p-2)/x * J_{p-1}(x) - J_{p-2}(x)$$

$$J_0(x) = \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{x}{2}\right)^{2k}}{(k!)^2}$$

$$J_1(x) = \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{x}{2}\right)^{2k+1}}{k! (k+1)!}$$

Calculele se fac cu precizia **eps** (**x**, **n** și **eps** se dau pe mediul de intrare).

56. Pentru **n** dat să se calculeze suma:

$$S = \frac{1}{2} + \frac{1 * 3}{2 * 4} + \dots + \frac{1 * 3 * \dots * (2n-1)}{2 * 4 * \dots * 2n}$$

57. Să se calculeze π cu o precizie cunoscută **epsilon** știind că:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

58. Să se calculeze **sin(x)** și **cos(x)** cu precizia dată **eps** utilizând dezvoltările în serie de puteri:

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

59. Fie sirurile $\{a_n\}$, $\{b_n\}$, $\{c_n\}$ generate cu relațiile de recurență:

$$a_n = (b_{n-1} + c_{n-1})/2 \quad b_n = (c_{n-1} + a_{n-1})/2 \quad c_n = (a_{n-1} + b_{n-1})/2 \text{ cu}$$

$$a_0 = \text{alfa} ; \quad b_0 = \text{beta} ; \quad c_0 = \text{gama}$$

alfa, **beta**, **gama** date. Știind că cele trei siruri sunt convergente și au o limită comună, să se calculeze cu o precizie **eps** dată această limită.

60. Să se calculeze prin dezvoltare în serie, cu precizia **eps** dată, **sin(x)** :

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Deoarece seria este rapid convergentă când argumentul se află în primul cadran, se va face reducerea sa la primul cadran utilizând următoarele relații:

$\sin(x) = -\sin(-x)$	dacă $x < 0$
$\sin(x) = \sin(x-2\pi n)$	dacă $x > 2\pi n$
$\sin(x) = -\sin(x-\pi)$	dacă $x > \pi$
$\sin(x) = \sin(\pi-x)$	dacă $x > \pi/2$

6. Funcții.(1)

6.1. Apelarea funcțiilor.

In C noțiunea de funcție este esențială, deoarece asigură *un mecanism de abstractizare a controlului*: rezolvarea unei părți a problemei poate fi încredințată unei funcții, moment în care suntem preocupați de *ce face funcția*, fără a intra în detalii privind *cum face funcția* anumite operații. Însăși programul principal este o funcție cu numele **main()**, iar programul C este reprezentat de o mulțime de definiri de variabile și de funcții.

Funcțiile pot fi clasificate în:

- funcții care întorc un rezultat
- funcții care nu întorc nici un rezultat (similar procedurilor din Pascal).

Apelul (referirea sau utilizarea) unei funcții se face prin:

```
nume_funcție (listă_parametri_efectivi)
```

Acesta poate apărea ca o instrucțiune:

```
nume_funcție (listă_parametri_efectivi);
```

De exemplu:

```
printf("x=%5.2lf\n", x);  
mesaj();
```

Pentru funcțiile care întorc un rezultat apelul de funcție poate apărea ca operand într-o expresie. De exemplu:

```
y=sin(x);  
nr_zile=bisect(an)+365;
```

Se remarcă faptul că lista de argumente (sau de parametri efectivi) poate fi vidă.

Funcțiile comunică între ele prin *lista de argumente* și prin *valorile întoarse de funcții*. Comunicarea poate fi realizată și prin *variabilele externe*, definite în afara tuturor funcțiilor.

Exemplul 11: O fracție este cunoscută prin numărătorul **x** și numitorul **y**, valori întregi fără semn. Să se simplifice această fracție.

Simplificarea se va face prin cel mai mare divizor comun al numerelor **x** și **y**. Vom utiliza o funcție având ca parametri cele două numere, care întoarce ca rezultat, cel mai mare divizor comun a lor. Funcția **main()** apelează funcția **cmmdc()** transmîndu-i ca argumente pe **x** și **y**. Funcția **cmmdc()** întoarce ca rezultat funcției **main()**, valoarea celui mai mare divizor comun. Programul, în care vom ignora deocamdată definirea funcției **cmmdc()**, este:

```
#include <stdio.h>  
int main()  
{ unsigned long x, y, z;  
  scanf("%lu%lu", &x, &y);  
  printf("%lu / %lu =", x, y);  
  z=cmmdc(x,y);  
  x/=z;  
  y/=z;  
  printf("%lu / %lu\n", x, y);
```

```
return 0;
}
```

6.2. Definiții de funcții.

În utilizarea curentă, o funcție trebuie să fie *definită* înainte de a fi *apelată*. Aceasta impune o definire a funcțiilor programului în ordinea sortării topologice a acestora: astfel mai întâi se vor defini funcțiile care nu apelează alte funcții, apoi funcțiile care apelează funcții deja definite. Este posibil să eliminăm această restricție, lucru pe care îl vom trata ulterior.

O funcție se definește prin *antetul* și *corpu* funcției.

Funcțiile nu pot fi incluse unele în altele; toate funcțiile se declară pe același nivel cu funcția `main()`.

In versiunile mai vechi ale limbajului, în antetul funcției parametrii sunt numai enumerați, urmând a fi declarați ulterior. Lista de parametri, în acest caz, este o enumerare de identificatori separați prin virgule.

```
tip_rezultat nume_functie( lista_de_parametri_formali )
{ declarare_parametri_formali;
  alte_declaratii;
  instructiuni;
}
```

In mod ușual parametrii funcției se declară în antetul acesteia. Declararea parametrilor se face printr-o listă de declarații de parametri, cu elementele separate prin virgule.

```
tip_rezultat nume_functie( tip nume, tip nume, ... )
{ declaratii;
  instructiuni;
}
```

O funcție este vizibilă din locul în care a fost declarată spre sfârșitul fișierului sursă (adică definiția funcției precede apelul).

Definirea funcției `cmmdc()` se face folosind algoritmul lui Euclid.

```
unsigned long cmmdc(unsigned long u, unsigned long v)
{ unsigned long r;
  do {r=u%v;
    u=v;
    v=r;
  } while (r);
  return u;
}
```

In cazul în care apelul funcției precede definiția, trebuie dat, la începutul textului sursă, un *prototip al funcției*, care să anunțe că definiția funcției va urma și să furnizeze tipul rezultatului returnat de funcție și tipul parametrilor, pentru a permite compilatorului să facă verificările necesare.

Prototipul unei funcții are un format asemănător antetului funcției și servește pentru a informa compilatorul asupra:

- tipului valorii furnizate de funcție;
- existența și tipurile parametrilor funcției

Spre deosebire de un antet de funcție, un prototip se termină prin ;

```
tip nume( lista_tipurilor_parametrilor_formali );
```

Din prototip interesează numai tipurile parametrilor, nu și numele acestora, motiv pentru care aceste nume pot fi omise.

```
void f(); /*functie fara parametri care nu intoarce nici un rezultat*/
int g(int x, long y[], double z);
int g(int, long[], double); /*aici s-au omis numele parametrilor*/
```

Dintre toate funcțiile prezente într-un program C prima funcție lansată în execuție este **main()**, independent de poziția pe care o ocupă în program.

Apelul unei funcții **g()**, lansat din altă funcție **f()** reprezintă un transfer al controlului din funcția **f()**, din punctul în care a fost lansat apelul, în funcția **g()**. După terminarea funcției **g()** sau la întâlnirea instrucțiunii **return** se revine în funcția **f()** în punctul care urmează apelului **g()**. Pentru continuarea calculelor în **f()**, la revenirea din **g()** este necesară salvarea stării variabilelor (contextului) din **f()** în momentul transferului controlului. La revenire în **f()**, contextul memorat a lui **f()** va fi refăcut.

O funcție apelată poate, la rândul ei, să apeleze altă funcție; nu există nici o limitare privind numărul de apeluri înălțuite.

6.3. Comunicarea între funcții prin variabile externe. Efecte laterale ale funcțiilor.

Comunicarea între funcții se poate face prin variabile externe tuturor funcțiilor; acestea își pot prelua date și pot depune rezultate în variabile externe. În exemplul cu simplificarea fracției vom folosi variabilele externe **a**, **b** și **c**. Funcția **cmmdc()** calculează divizorul comun maxim dintre **a** și **b** și depune rezultatul în **c**. Întrucât nu transmite date prin lista de parametri și nu întoarce vreun rezultat, funcția va avea prototipul **void cmmdc()**:

```
#include <stdio.h>
unsigned long a, b, c; // variabile externe
// definirea functiei cmmdc()
int main()
{ scanf("%lu%lu", &a, &b);
  printf("%lu / %lu = ", a, b);
  cmmdc();
  a/=c;
  b/=c;
  printf("%lu / %lu\n", a, b);
  return 0;
}
```

Definiția funcției **cmmdc()** din Exemplul 11, este:

```
void cmmdc()
{ unsigned long r;
  do { r = a%b;
        a = b;
        b = r;
    } while (r);
```

```
c = a;
}
```

Dacă se execută acest program, se constată un rezultat ciudat, și anume, orice fracție, prin simplificare ar fi adusă la forma **1/0** ! Explicația constă în faptul că funcția **cmmdc()** prezintă *efecte laterale*, și anume modifică valorile variabilelor externe **a**, **b** și **c**; la ieșirea din funcție **a==c** și **b==0**, ceea ce explică rezultatul.

Așadar, un *efect lateral* (sau secundar), reprezintă modificarea de către funcție a unor variabile externe.

În multe situații aceste efecte sunt nedorite, duc la apariția unor erori greu de localizat, făcând programele neclare, greu de urmărit, cu rezultate dependente de ordinea în care se aplică funcțiile care prezintă efecte secundare. Astfel într-o expresie în care termenii sunt apeluri de funcții, comutarea a doi termeni ar putea conduce la rezultate diferite!

Vom corecta rezultatul, limitând efectele laterale prin interzicerea modificării variabilelor externe **a** și **b**, ceea ce presupune modificarea unor copii ale lor în funcția **cmmdc()** sau din programul de apelare

```
void cmmdc()
{ unsigned long r, ca, cb;
  ca = a;
  cb = b;
  do{r = ca%cb;
     ca = cb;
     cb = r;
  } while (r);
  c = ca;
}
```

Singurul efect lateral permis în acest caz – modificarea lui **c** asigură transmiterea rezultatului către funcția apelantă.

Soluția mai naturală și mai puțin expusă erorilor se obține realizând *comunicația* între funcția **cmmdc()** și funcția **main()** nu prin variabile externe, ci prin parametri, folosind o funcție care întoarce ca rezultat cmmdc.

Transmiterea parametrilor prin valoare, mecanism specific limbajului C, asigură păstrarea intactă a parametrilor actuali **x** și **y**, deși parametru formali corespunzători: **u** și **v** se modifică! Parametrii actuali **x** și **y** sunt copiați în variabilele **u** și **v**, astfel încât se modifică copiile lor nu și **x** și **y**.

În fișierul sursă funcțiile pot fi definite în orice ordine. Mai mult, programul se poate întinde în mai multe fișiere sursă. Definirea unei funcții nu poate fi totuși partajată în mai multe fișiere.

O funcție poate fi apelată într-un punct al fișierului sursă, dacă în prealabil a fost definită în același fișier sursă, sau a fost anunțată.

Exemplul 12:

```
#include <stdio.h>
unsigned long fact(unsigned char); // prototipul anunta functia
int main()
{ printf("5!=%ld\n", fact(5));      // apel functie
  printf("10!=%ld\n", fact(10)); }
```

```

    getch();
}

long fact(unsigned char n)          // antet functie
{ long f=1;                      // corp functie
  short i;
  for (i=2; i<=n; i++)
    f*=i;
  return(f);
}

```

Tipurile funcțiilor pot fi:

- tipuri predefinite
- tipuri pointer
- tipul structură (înregistrare)

6.4. Funcții care apelează alte funcții.

Programul principal (funcția `main()`) apelează alte funcții, care la rândul lor pot apela alte funcții. Ordinea definițiilor funcțiilor poate fi arbitrară, dacă se declară la începutul programului prototipurile funcțiilor. În caz contrar definițiile se dau într-o ordine în care nu sunt precedate de apeluri ale funcțiilor.

Exemplul 13: Pentru o valoare întreagă și pozitivă **n** dată, să se genereze triunghiul lui Pascal,adică combinările:

$$\begin{array}{ccccccc}
 C_0^0 & & & & & & \\
 C_1^0 & C_1^1 & & & & & \\
 \cdot & \cdot & \cdot & & & & \\
 C_n^0 & C_n^1 & \dots & C_n^n & & &
 \end{array}$$

Combinările vor fi calculate utilizând formula cu factoriale: $C_n^p = n! / p! / (n-p)!$

```

#include <stdio.h>
unsigned long fact(int);      /*prototip functie factorial*/
unsigned long comb(int,int);  /*prototip functie combinari*/
int main()                    /*antet functie main*/
{ int k,j,n;
  scanf("%d", &n);
  for (k=0;k<=n;k++)
    { for (j=0;j<=k;j++)
        printf("%6lu ", comb(k,j));
        printf("\n");
    }
}
unsigned long comb(int n, int p) /*antet functie comb*/
{ return (fact(n)/fact(p)/fact(n-p));
}
/* functia fact a mai fost definita */

```

6.5. Programe cu mai multe fișiere sursă.

Pentru programele mari este mai comod ca acestea să fie constituite din mai multe fișiere sursă, întrucât programul este conceput de mai mulți programatori (echipă) și fiecare funcție poate constitui un fișier sursă, ușurându-se în acest mod testarea. Reamintim că o funcție nu poate fi împărțită între mai multe fișiere.

Un exemplu de program constituit din 2 fișiere sursă este:

```
/* primul fisier Fis1.c */
void F(); /* prototipul functiei definite in fisierul 2*/
#include <stdio.h>
void main(){
    F(); /* apelul functiei F */
    ...
}

/* al doilea fisier Fis2.c */
#include <stdio.h>
void F(){
    ...
}
```

6.6. Fișiere antet.

In C se pot utiliza o serie de funcții aflate în bibliotecile standard. Apelul unei funcții de bibliotecă impune prezența prototipului funcției în textul sursă. Pentru a simplifica inserarea în textul sursă a prototipurilor funcțiilor de bibliotecă, s-au construit *fișiere de prototipuri*. Acestea au extensia .h (*header* sau *antet*).

De exemplu fișierul **stdio.h** conține prototipuri pentru funcțiile de bibliotecă utilizate în operațiile de intrare / ieșire; fișierul **string.h** conține prototipuri pentru funcțiile utilizate în prelucrarea sirurilor de caractere.

Includerea în textul sursă a unui fișier antet se face folosind directiva `#include`

Tabel 6.1. Funcții declarate în fișierele antet

Fișier antet	Funcții conținute
stdio.h	printf, scanf, gets, puts, ...
conio.h	putch, getch, getche, ...
math.h	sqrt, sin, cos, ...

6.7. Funcții matematice uzuale.

Fișierul antet **<math.h>** conține semnăturile (prototipurile) unor funcții matematice des folosite. Dintre acestea amintim:

Tabel 6.2. Semnăturile funcțiilor din biblioteca matematică

Notăție	Semnătură (prototip)	Operație realizată
sin(x)	double sin(double);	funcții trigonometrice directe
cos(x)	double cos(double);	
tg(x)	double tan(double);	
arcsin(x)	double asin(double);	
arcos(x)	double acos(double);	funcții trigonometrice inverse

<code>arctg(x)</code>	<code>double atan(double);</code>	
<code>arctg(y/x)</code>	<code>double atan2(double, double);</code>	
<code>sinh(x)</code>	<code>double sinh(double);</code>	
<code>cosh(x)</code>	<code>double cosh(double);</code>	funcții hiperbolice
<code>th(x)</code>	<code>double tanh(double);</code>	
<code>exp(x)</code>	<code>double exp(double);</code>	exponențială naturală
<code>10^n</code>	<code>double pow10(int);</code>	exponențială zecimală
<code>a^b</code>	<code>double pow(double, double);</code>	exponențială generală
<code>ln(x)</code>	<code>double log(double);</code>	logaritm natural
<code>lg(x)</code>	<code>double log10(double);</code>	logaritm zecimal
	<code>double fabs(double);</code>	
<code> x </code>	<code>int abs(int);</code>	valoare absolută
	<code>long labs(long);</code>	
<code>√x</code>	<code>double sqrt(double);</code>	rădăcină pătrată
	<code>long double sqrtl(long double);</code>	
<code>[x]</code>	<code>double ceil(double);</code>	întregul minim $\geq x$
<code>[x]</code>	<code>double floor(double);</code>	întregul maxim $\leq x$
<code>conversii</code>	<code>double atof(const char *);</code>	conversie sir de caractere în float
	<code>long double atold (const char *);</code>	conversie sir de caractere în long double

6.8. Probleme rezolvate.

1. Numerele naturale pot fi clasificate în: *deficiente*, *perfecte* sau *abundente*, după cum suma divizorilor este mai mică, egală sau mai mare decât valoarea numărului. Astfel: **n=12** este abundant deoarece are suma divizorilor: **sd = 1+2+3+4+6 = 16 > 12**, **n=6** este perfect: **sd = 1+2+3 = 6**, iar **n=14** este deficient deoarece **sd = 1+2+7 < 14**.

- Definiți o funcție având ca parametru un număr întreg **n**, funcție care întoarce ca rezultat **-1**, **0** sau **1** după cum numărul este deficient, perfect sau abundant.
- Scrieți o funcție **main()** care citește două valori întregi **x** și **y** și clasifică toate numerele naturale cuprinse între **x** și **y** afișând după fiecare număr tipul acestuia, adică deficient, perfect sau abundant.

```
#include <stdio.h>
#include <stdlib.h>
int tip(int n){
    int sd, d;
    sd = 1;
    for(d=2; d<=n/2; d++)
        if(n%d==0)
            sd += d;
    if(sd < n) return -1;
    if(sd == n) return 0;
    return 1;
}

int main() {
```

```

int n, x, y;
scanf("%d%d", &x, &y);
for(n=x; n<=y; n++) {
    printf("%5d ", n);
    switch(tip(n)){
        case -1: printf("deficient\n"); break;
        case 0: printf("perfect\n"); break;
        case 1: printf("abundent\n");
    }
}
system("PAUSE");
return 0;
}

```

2. Verificați „conjectura lui Goldbach”, potrivit căreia orice număr par poate fi scris ca cel puțin o sumă a două numere prime. Programul va genera și afișa descompunerile tuturor numerelor pare până la o limită dată L.

Vom considera și pe 1 ca număr prim. Exceptând primele două descompuneri: $2=1+1$ și $4=1+3=2+2$, celelalte descompuneri vor avea termenii numere prime impare. Pentru un număr par p, o eventuală descompunere, având termenii a și p-a impune ca acestea să fie prime și impare.

```

#include <stdio.h>
#include <stdlib.h>
int prim(int n){
    int d;
    for(d=2; d*d <= n; ){
        if(n%d==0) return 0;
        if(d==2)
            d = 3;
        else
            d+=2;
    }
    return 1;
}

int main(){
    int L, p, a, nd;
    scanf("%d", &L);
    /* scrierea primelor două descompuneri */
    printf(" 2=1+1\n 4=1+3=2+2\n");
    for(p=6; p<=L; p+=2){
        printf("%4d ", p);
        nd = 0;
        for(a=1; a<=p/2; a+=2)
            if(prim(a) && prim(p-a)){
                nd++;
                printf("=%4d+%4d", a, p-a);
            }
    }
}

```

```

    if(nd==0)
        printf("nu verifica conjectura\n");
    else
        printf("\n");
}
system("PAUSE");
return 0;
}

```

6.9. Probleme propuse.

1. O pereche de numere naturale a și b se numesc *numere prietene*, dacă suma divizorilor unuia dintre numere este egală cu celălalt număr. De exemplu 220 și 284 sunt numere prietene deoarece:

$$\text{sd}(220) = 1+2+4+5+10+11+20+22+44+55+110 = 284$$

$$\text{sd}(284) = 1+2+4+71+142 = 220$$

- a) Scrieți o funcție având ca parametri un număr natural, care întoarce suma divizorilor numărului.
- b) Scrieți o funcție având ca parametri două numere naturale, care întoarce 1 sau 0, după cum cele două numere sunt sau nu prietene.
- c) Scrieți o funcție **main()**, care în intervalul **x**, **y** dat găsește toate perechile de numere prietene și le afișează.

3. Să se rezolve ecuația $f(x)=0$, prin metoda tangentei, pornind cu un $x^{(0)}$ dat, și calculând $x^{(k+1)}=x^{(k)} - f(x^{(k)}) / f'(x^{(k)})$ cu o precizie dată **eps**, care se atinge când $|x^{(k+1)} - x^{(k)}| < \text{eps}$. Funcțiile **f(x)** și **f'(x)** sunt date de programator.

4. Se consideră funcția $f(x) = \ln(1 + x^2)$. Să se scrie un program care tabelează funcția pe **n** intervale, fiecare interval fiind precizat prin capetele **a** și **b** și pasul de afișare **h**.

5. Să se scrie toate descompunerile unui număr par ca o sumă de două numere prime. Se va utiliza o funcție care stabilește dacă un număr este sau nu prim.

6. Să se scrie în C:

- a) Un subprogram funcție pentru calculul valorii unei funcții **f(x)** pentru o valoare dată **x**, definită astfel:

$$f(x) = \begin{cases} 1 - x\sqrt{-x} & \text{pentru } x < -1 \\ \sqrt{1 - x^2} & \text{pentru } -1 \leq x \leq 1 \\ 1 + x\sqrt{x} & \text{pentru } x > 1 \end{cases}$$

- b) O funcție pentru calculul integralei definite:

$$I = \int_a^b f(x) dx$$

prin metoda trapezelor, cu **n** pași egali, pe intervalul **[a, b]**, după formula

$$I \cong \frac{h}{2} \left[f(a) + f(b) + \sum_{i=1}^{n-1} f(a + ih) \right]$$

unde $h = (b-a)/n$

c) Un program care calculează integrala unei funcții definite la punctul a), folosind funcția b), pe un interval dat $[p, q]$, cu precizia **epsilon** (se repetă calculul integralei pentru $n=10, 20, \dots$ pași, până când diferența dintre două integrale succesive devine mai mică decât **epsilon**)

7. Pentru un număr dat **N** să se afișeze toți factorii primi ai acestuia și ordinul lor de multiplicitate. Se va utiliza o funcție care extrage dintr-un număr un factor prim.

8. Utilizând o funcție pentru calculul celui mai mare divizor comun a două numere, să se calculeze c.m.m.d.c. a n elemente întregi ale unei liste date.

9. Pentru fiecare element al unei liste de numere întregi date, să se afișeze numărul prim cel mai apropiat de el ca valoare. Dacă două numere prime sunt la distanță egală de un element din listă se vor afișa ambele numere prime.

8. Tablouri și pointeri.

8.1. Tablouri cu o dimensiune (vectori).

Un tablou cu o singură dimensiune este o succesiune de variabile având toate de același tip (*tipul de bază al tabloului*), care ocupă o zonă contiguă de memorie. Un tablou are:

- o dimensiune (egală cu numărul de elemente al tabloului)
- un nume (care identifică global tabloul)
- o clasă de alocare
- un tip comun tuturor elementelor tabloului

Dimensiunea tabloului precizează numărul de elemente printr-o constantă întreagă sau printr-o expresie constantă.

La declararea unui tablou se specifică: numele, tipul de bază, clasa de alocare și dimensiunea.

tip nume[dimensiune];

sau

clasă tip nume[dimensiune];

Exemple:

```
int x[10];           /* tablou de 10 intregi */
char litere[2*26];  /* tablou de 52 caractere */
```

Tipul elementelor tabloului poate fi un tip fundamental, enumerat, înregistrare, pointer sau un tip definit.

Numele tabloului este adresa primului element din tablou (de exemplu **x** este adresa primului element, adică **@x[0]**). Aceasta explică de ce nu este permisă o atribuire între două tablouri.

Accesul la un element din tablou se face printr-o *variabilă indexată*, formată din numele tabloului și un *index* - o expresie cuprinsă între 0 și **dimensiune-1**.

Primul element va fi desemnat aşadar prin **x[0]**, al doilea element prin **x[1]**, al N-lea prin **x[N-1]**

Un tablou declarat în interiorul unei funcții are implicit clasa **auto**, în timp ce tablourile declarate în exteriorul tuturor funcțiilor au în mod implicit clasa **extern**.

Un tablou declarat în exteriorul tuturor funcțiilor cu specificatorul **static** este alocat la adrese fixe, fiind vizibil numai *în fișierul* în care este declarat.

Un tablou declarat în interiorul unei funcții cu specificatorul **static** este alocat la adrese fixe, fiind vizibil numai *în interiorul funcției*.

Prelucrările pe tablouri se implementează cu cicluri **for**.

Exemplul 15: Să se afișeze elementele unui tablou citit de la intrarea standard, câte 10 pe un rând.

```
#include <stdio.h>
int main()
/* citirea și afisarea elementelor unui vector */
{ int x[10], n, j;
  scanf("%d", &n);
  for (j=0; j < n; j++)
    scanf("%d", &x[j]);
  for (j=0; j < n; j++)
```

```

    printf("%5d%c", x[j], (j%10==9||j==n-1)?'\n':' ');
    return 0;
}

```

La declararea unui tablou, acesta poate fi și inițializat, dacă declarația este urmată de semnul = și de o listă de valori inițiale, separate prin virgule și incluse între acolade.

Exemplu:

```

int prime[5]={2,3,5,7,11};
char vocale[5]={'a','e','i','o','u'};

```

La declararea unui tablou inițializat se poate omite dimensionarea, situație în care se ia ca dimensiune numărul de valori inițiale:

```

char operator[]={'+','-','*','/'};
long x[]={1,10,100,1000,10000,100000};

```

Tablourile vor avea 4, respectiv 6 elemente.

Exemplul 16: Scrieți un program care convertește un sir de caractere reprezentând un număr scris cu cifre romane în corespondență cu cifre arabe.

Notația cu cifre romane este un sistem nepozitional, care folosește cifrele: **M, D, C, L, X, V, I**, având respectiv valorile: **1000, 500, 100, 50, 10, 5, 1**.

Pentru a obține valoarea numărului, se pleacă cu acesta de la 0 și se adaugă pe rând contribuțiile cifrelor astfel: dacă valoarea cifrei romane curente este mai mare sau egală cu cifra care urmează, atunci valoarea cifrei curente se adaugă la valoarea numărului arab, altfel se scade din acesta. De exemplu numărul roman **MCMXCVIII** are ca valoare pe **1998**

Tabel 8.1. Transformarea unui număr din cifre romane în notație arabă

Cifra curentă	Cifra următoare	Relația dintre ele	Contribuția cifrei curente	Valoare număr
M	C	>	+1000	1000
C	M	<	-100	900
M	X	>	+1000	1900
X	C	<	-10	1890
C	V	>	+100	1990
V	I	>	+5	1995
I	I	=	+1	1996
I	I	=	+1	1997
I		>	+1	1998

Se observă că pentru a considera și contribuția ultimei cifre a numărului am fost nevoiți să “prelungim” numărul cu caracterul spațiu liber, căruia i-am asociat valoarea 0. Pentru stabilirea corespondenței cifră română – valoare asociată, vom defini o funcție **int conv(char)** care folosește două tablouri: **roman** și **arab**, inițializate respectiv cu caracterele reprezentând cifrele romane și cu valorile acestora, definite ca externe. Numărul roman **nrom**, citit de la intrarea standard va fi terminat prin spațiu liber.

```

#define LMAX 15
#include <stdio.h>
char roman[]="MDCLXVI ";
int arab[]={1000,500,100,50,10,5,1,0};

```

```

int conv(char);
int main(){
    char nrom[LMAX];
    int i,n=0; /*n = lungimea numarului scris cu cifre romane
    int narab=0;
    int crt,urm;
    while((nrom[n++]=getchar()) !=' ')
        ;
    n--;
    for (i=0; i<n-1; i++){
        crt=conv(nrom[i]);
        urm=conv(nrom[i+1]);
        if(crt>=urm)
            narab+=crt;
        else
            narab-=crt;
    }
    for (i=0;i<n;i++)
        printf("%c",nrom[i]);
    printf("=%d\n",narab);
    return 0;
}

int conv(char c){
    int j=0;
    while(roman[j++]!=c && j<8)
        ;
    if(j<8)
        return arab[--j];
    else
        return -1;
}

```

8.2. Probleme rezolvate.

1. Definiți o funcție care determină poziția elementului minim al unui tablou cu elemente reale.

```

int pmin(int n, double x[]){
    int i, pm=0;
    for(i=1; i<n; i++)
        if(x[i]<x[pm])
            pm = i;
    return pm;
}

```

2. Definiți o funcție care întoarce poziția în care se află o valoare întreagă dată **y** (cheie), într-un tablou **x** nesortat cu **n** elemente întregi. Se va efectua o căutare secvențială. Dacă valoarea căutată nu se află în tablou funcția întoarce -1.

```

int cautsecv(int n, int x[], int y){
    int i;
    for(i=0; i<n; i++)

```

```

    if(x[i]==y) return i;
    return -1;
}

```

3. Definiți o funcție care întoarce poziția în care se află o valoare întreagă dată **y** (cheie), într-un tablou **x** sortat cu **n** elemente întregi. Se va efectua o căutare binară. Dacă valoarea căutată nu se află în tablou funcția întoarce poziția pe care ar trebui să o ocupe cheia în tabloul sortat, precedată de semnul - .

- se compară valoarea căutată **y** cu elementul din mijlocul tabloului **x[m]**
- dacă sunt egale, elementul **y** a fost găsit în poziția **m**
- în caz contrar se continuă căutarea într-o din jumătățile tabloului (în prima jumătate, dacă **y < x[m]** sau în a doua jumătate dacă **y > x[m]**).

Funcția întoarce poziția **m** a valorii **y** în **x** sau **-i**, dacă **y** nu se află în **x**, **i** fiind poziția unde ar trebui să se afle **y**. Complexitatea este **O(log₂n)**

```

int CB(int i, int j, int y, int x[]){
    int m;
    while(i <= j){
        m = (i+j)/2;
        if(y == x[m]) return m;
        if(y < x[m])
            j = m-1;
        else
            i = m+1;
    }
    return -i;
}

int CautBin(int n, int x[], int y){
    if(n==0) return 0;
    return CB(0, n-1, y, x);
}

```

4. Definiți funcții care realizează operațiile de adunare, înmulțire și împărțire între polinoame. Un polinom va fi precizat prin gradul său și un tablou al coeficienților, dați în ordine descrescătoare a puterilor. Dacă gradul polinomului este **n**, atunci tabloul coeficienților va avea **n+1** elemente, chiar dacă unele din ele sunt nule.

Polinomul sumă va avea gradul **max (na, nb)**. Ultimii **min (na, nb)** coeficienți ai polinomului sumă sunt de forma **a[i]+b[i]**, iar primii coeficienți vor fi termenii corespunzători din **a**, dacă acesta are gradul mai mare sau din **b**, în caz contrar.

```

void adpol(int na, int a[], int nb, int b[], int c[]){
    int i, max, min;
    if(na>nb){
        max = na;
        min = na-nb;
        for(i=0; i<min; i++)
            c[i] = a[i];
    }
    else {
        max = nb;
        min = nb-na;
        for(i=0; i<min; i++)
            c[i] = b[i];
    }
}

```

```

    for(i=0; i<min; i++)
        c[i] = b[i];
    }
    for(i=min; i<=max; i++)
        c[i] = a[i] + b[i];
}

```

Gradul polinomului produs va fi **na+nb**. Coeficienții **c_k**, cu **k=0 : na+nb** se calculează ca:

$$c_k = \sum_{i+j=k} a_i b_j = \sum_{\substack{k-i \geq 0, \\ k-i \leq nb}} a_i b_{k-i} = \sum_{i=k}^{nb-k} a_i b_{k-i}$$

Ceea ce se traduce prin:

```

for(k=0; k<=na+nb; k++) {
    c[k] = 0;
    for(i=k; i<=nb-k; i++)
        c[k]+=a[i]*b[k-i];
}

```

Este posibil să folosim numai prima parte a relației de mai sus, situație în care la fiecare iterație se adaugă un produs **a_ib_j** de fiecare dată la alt termen **c_{i+j}**.

```

void mulpol(int na, int a[], int nb, int b[], int c[]) {
    int i, j;
    for(i=0; i<=na+nb; i++)
        c[i] = 0;
    for(i=0; i<=na; i++)
        for(j=0; j<=nb; j++)
            c[i+j]+=a[i]*b[j];
}

```

La împărțirea a două polinoame, gradul polinomului cât va fi **na-nb**, iar a polinomului rest **nb-1**. Se simulează împărțirea polinoamelor. În pasul **i**, un termen din cât este: **c_i = a_i/b₀**, iar restul parțial:

$$a_{i+j} = a_{i+j} - c_i b_j$$

Restul se regăsește în ultimele nb elemente din deîmpărțit, de unde este transferat în vectorul rest.

```

void divpol(int na, int a[], int nb, int b[], int c[], int r[]) {
    int i, j;
    for(i=0; i<=na-nb; i++) {
        c[i] = a[i]/b[0];
        for(j=0; j<=nb; j++)
            a[i+j]-=c[i]*b[j];
    }
    for(i=0; i<nb; i++)
        r[i] = a[na-nb+i+1];
}

```

5. Definiți o funcție care sortează un tablou **x** cu **n** elemente reale, folosind metoda selecției.

Tabloul de sortat este partaționat în două zone: zona din stînga – deja sortată și zona din dreapta. În zona nesortată se determină poziția elementului minim. Dacă aceasta nu coincide cu poziția primului element din zona nesortată, se interschimbă elementele din cele două poziții și se extinde zona sortată cu o poziție spre dreapta.

Fie **k** – începutul zonei nesortate; inițial zona nesortată cuprinde întreg tabloul, deci **k=0**. În final, zona nesortată nu mai are elemente, deci **k=n**.

```
void sortsel(int n, double x[]){
    int pm, k, j;
    for(k=0; k<n; k++){
        /*restrange zona nesortata*/
        /*determina pozitia pm a elementului minim*/
        pm = k;
        for(j=k+1; j<n; j++)
            if(x[j] < x[pm])
                pm = j;
        if(x[pm] != x[k]) {
            t = x[pm];
            x[pm] = x[k];
            x[k] = t;
        }
    }
}
```

6. Definiți o funcție care sortează un tablou **x** cu **n** elemente reale, folosind metoda inserției.

Tabloul este partionat într-o zonă sortată și o zonă nesortată încă. Inițial elementul **x[k]**, **k=1** din zona nesortată este inserat în zona sortată, care se extinde cu o poziție spre dreapta, păstrând relația de ordine în zona sortată.

```
void sortins(int n, double x[]){
    int k, t, p, i;
    for(k=1; k<n; k++){
        t = x[k];
        /*inserare t in zona sortata x[0]:x[k-1] in pozitia p */
        /*determinare pozitie p a primului element x[p]>=t */
        for(p=0; p<k && x[p]<t; p++)
            ;
        if(x[p]>=t) {
            for(j=p; j<k; j++)
                x[j+1] = x[j];
            x[p] = t;
        }
    }
}
```

7. Eventualele rădăcini întregi ale ecuației cu coeficienți întregi:

$x^n + a_0x^{n-1} + \dots + a_{n-1} = 0$ sunt $\pm 1, \pm a_{n-1}, \pm d$, în care **d** este un divizor nebalan a lui **a_{n-1}**

Se citesc **n** și tabloul **a**. Se cere să se găsească și să se afișeze rădăcinile întregi simple, duble, etc ale ecuației.

Indicație: Se vor defini și folosi funcțiile:

int ndiv(int n, int *div); -crează un tablou al tuturor divizorilor lui **n** și a celor cu semn schimbat și întoarce numărul acestora
int eval(int n, int *a, int x); - stabilește dacă polinomul are sau nu rădăcina **x** folosind schema lui Horner

void copy(int *ns, int *as, int n, int *a); - copiază **n** și tabloul **a** în **ns** și **as**
void deriv(int n, int *a, int *nd, int *ad); - derivează polinomul cu grad **n** și coeficienții **a** și pune în **nd** gradul polinomului derivat și în **ad** coeficienții acestuia

8. Se consideră fracția rațională:

$$F(x) = \frac{P_m(x)}{(x - a_0) \cdots (x - a_{n-1})} = \frac{b_0 x^m + b_1 x^{m-1} + \cdots + b_m}{(x - a_0) \cdots (x - a_{n-1})} \text{ cu } m < n < 20.$$

Scrieți o funcție cu semnătura:

```
void dezv(int m, int n, int *a, int *b, int *c);
```

care calculează coeficienții dezvoltării fracției raționale: $F(x) = \frac{c_0}{x - a_0} + \cdots + \frac{c_{n-1}}{x - a_{n-1}}$

Funcția care evaluează un polinom într-un punct dat **x** se consideră cunoscută.

$$\text{Indicație: } c_j = \frac{P_m(a_j)}{\prod_{\substack{k=0, \\ k \neq j}}^n (a_j - a_k)},$$

```
void dezv(int m, int n, double *a, double *b, double *c) {
    int j, k;
    double p=1.;
    for(j=0; j<n; j++) {
        for(k=0; k<=n; k++)
            if(k!=j)
                p*=(a[j]-a[k]);
        c[j] = eval(m, b, a[j])/p;
    }
}
```

8.9. Probleme propuse.(Tablouri cu o dimensiune)

1. Intr-un sir **x** cu **n** componente reale, să se determine media aritmetică a elementelor pozitive situate între primul element pozitiv și ultimul element negativ al sirului, exceptând aceste elemente. Cazurile speciale vor fi clarificate prin mesaje corespunzătoare.

2. Intr-un sir **S** cu **n** elemente întregi (**n ≤ 100**) să se determine elementele distințe.

3. Doi vectori **x** și **y** au **n**, respectiv **m** elemente reale distințe (**m, n≤10**). Să se creeze un nou vector **z** cu elementele comune ale celor doi vectori. (intersecția elementelor mulțimilor reprezentate de cei doi vectori).

4. Doi vectori **x** și **y** au **n**, respectiv **m** elemente reale distințe (**m, n≤10**). Să se creeze un nou vector **z** cu conținând elementele celor doi vectori. Elementele comune din cei doi vectori apar în **z** o singură dată. (reuniunea elementelor mulțimilor reprezentate de cei doi vectori).

5. Se citește o valoare întreagă **n** (**0 < n ≤ 100**) și **n** valori reale cu care se crează un vector **x**. Scrieți un program C care calculează și afișează:

- Valoarea medie

- Abaterea medie pătratică:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{n-1} (x_i - x_{\text{med}})^2}{n(n-1)}}$$

- Numărul de componente care depășesc valoarea medie
- Să se creeze un vector \mathbf{y} cu componente din \mathbf{x} mai mari decât valoarea medie și să se afișeze câte 5 elemente pe o linie.

6. Se citesc n ($n \leq 100$) coordonate reale \mathbf{x} , \mathbf{y} ale unor puncte în plan și se crează cu acestea două tablouri \mathbf{x} și \mathbf{y} .

- Să se afișeze toate tripletele de puncte coliniare.
- Să se afișeze punctele i , j , k pentru care aria triunghiului determinat de aceste puncte este maximă..

Indicație: Aria determinată de punctele i , j , k este:

$$S = \frac{1}{2} \begin{vmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ 1 & 1 & 1 \end{vmatrix}$$

Dacă punctele sunt coliniare, atunci $S=0$.

7. Să se calculeze coeficienții binomiali $C_n^1, C_n^2, \dots, C_n^p$ în care n și p sunt întregi pozitivi dați ($p < n$), cunoscând relația de recurență:

$$C_n^k = (n-k+1)/k * C_n^{k-1} \quad \text{pornind cu } C_n^0 = 1$$

Se vor utiliza tablouri

8. Să se calculeze coeficienții: c_0, c_1, \dots, c_{n-1} , pentru n dat, știind că:

$$c_0/(p+1) + c_1/p + \dots + c_p/1 = 1 \quad \text{pentru } p=0, 1, 2, \dots, n.$$

9. Să se calculeze coeficienții polinomului Cebâșev de ordinul n , pornind de la relația de recurență

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \quad k > 2$$

$$T_0(x) = 1, \quad T_1(x) = x$$

obținând în prealabil relații de recurență pentru coeficienți.

10. Un număr întreg este reprezentat prin cifrele sale $c[0], c[1], \dots, c[n-1]$, ($c[0]$ fiind cifra cea mai semnificativă).

Să se calculeze câtul $q[0], q[1], \dots, q[m-1]$ obținut prin impărțirea numărului dat prin numărul întreg p .

11. Dându-se o valoare întreagă n , să se genereze reprezentarea fracțiilor zecimale $1/2^k$ unde $k = 1, 2, \dots, n$.

12. Să se ordoneze crescător sirul x cu n elemente utilizând observația că în sirul ordonat crescător orice subșir $x[i], |x[i+1], \dots, x[n]$ cu $i=0, 1, 2, \dots, n-2$ are elementul $x[i]$ maxim.

13. Să se ordoneze crescător o listă având n componente ($n \leq 100$) de numere intregi, utilizând metoda contorizării inversărilor (denumită și metoda bulelor)

14. Un număr de bare ($n \leq 100$) sunt date prin lungimile lor. Se dau de asemenea P categorii de lungimi (sau standarde) între care trebuie să se încadreze lungimile pieselor ($P \leq 10$).

O categorie de lungimi este precizată prin 2 limite: una minimă și cealaltă maximă; presupunem că aceste categorii de lungimi formează intervale disjuncte.

O piesă i se incadrează în categoria de lungimi j dacă: $LMIN[j] \leq L[i] \leq LMAX[j]$.

Să se calculeze și să se afișeze:

- -numărul de piese din fiecare clasă
- -dimensiunea medie a pieselor din fiecare clasă de lungimi
- -numărul de rebuturi și lungimile barelor rebutate.

15. Să se calculeze coeficienții b_i , $i=0..n-1$ din dezvoltarea produsului:

$$(x+a_0)(x+a_1)\dots(x+a_{n-1}) = x^n + b_0 x^{n-1} + \dots + b_{n-1}$$

Se dau n și coeficienții a_0, a_1, \dots, a_{n-1}

16. N copii identificăți prin numerele $1, 2, \dots, N$, joacă următorul joc: se așează în cerc în ordinea $1, 2, \dots, N$ și începând de la copilul k numără de la 1 la p eliminând din cerc pe cel care a fost numărat cu p ; numărătoarea începe de la următorul eliminându-se al 2 -lea și.m.d.

Folosindu-se identificarea inițială să se stabilească ordinea de ieșire a copiilor din joc.

17. Într-un sir s cu n elemente să se determine elementele distincte.

Exemplu: în sirul $2 \ 2 \ 5 \ 4 \ 5 \ 1 \ 2$ elementele distincte sunt : $2 \ 5 \ 4 \ 1$

Indicație

- elementele unice sunt distincte și se tipăresc
- dintre elementele cu mai multe apariții se tipărește un singur reprezentant. La întâlnirea primei egalități $s[i]=s[j]$ se abandonează căutarea și se tipărește $s[i]$ numai dacă $i < j$ (este prima apariție a lui $s[i]$).

18. De pe mediul de intrare se citește un număr întreg $n \neq 0$ urmat de n valori reale. Să se introducă aceste valori într-un tablou x pe măsura citirii lor, astfel încât tabloul să fie ordonat crescător.

19. Dându-se o valoare x și un tablou a cu n elemente, să se separe acest tablou în două partiții astfel încât elementele din prima partiție să fie mai mici sau egale cu x , iar cele din a doua partiție strict mai mari decât x .

20. Să se determine elementul maxim dintr-un sir cu n elemente și poziția pe care el apare. În cazul unui maxim cu mai multe apariții se va nota prima sa apariție.

21. Se dau două siruri x și y ordonate strict crescător, având M și respectiv N elemente. Să se construiască un sir z ordonat strict crescător conținând elementele sirurilor x și y . ("interclasare de siruri").

22. Se consideră un vector x cu n componente, ordonat strict crescător și o valoare y . Să se insereze această valoare în vectorul x astfel încât el să rămână ordonat strict crescător. Se va face o căutare rapidă a lui y în sir (căutare binară). Sirul inițial și cel rezultat se vor tipări cu câte 5 elemente pe linie.

23. Dintr-un sir dat să se determine lungimea și poziția subșirului strict crescător cel mai lung, format din elemente alăturate din sirul dat. Sirul dat se tipărește în ecou cu câte 10 elemente pe linie. Subșirul de lungime maximă se afișează cu 5 elemente pe linie.

24. Se spune că sirul x cu k elemente "intră" în sirul s cu n elemente $k \leq n$, dacă există un subșir contiguu $s_i, s_{i+1}, \dots, s_{i+k}$ cu proprietatea că elementele lui sunt identice cu elementele sirului x . Să se stabilească numărul de "intrări" ale lui x în s și pozițiile în s ale elementelor de unde începe intrarea.

25. Doi vectori a și b au câte n componente fiecare, precizate pe mediul de intrare. Să se calculeze unghiul dintre ei exprimat în radiani.

26. Două numere sunt păstrate prin tablourile cifrelor lor. Să se obțină tabloul cifrelor produsului numerelor.

27. Dându-se un număr întreg n să se construiască două tablouri, primul conținând factorii primi ai lui n , iar cel de-al doilea multiplicitatele acestora.

28. Să se calculeze cel mai mare divizor comun cmmdc a două numere date m și n utilizând următoarea metodă:

- se creează tablouri cu factorii primi ai celor două numere și multiplicitatele lor
- se selectează în cmmdc factorii primi comuni la puterea cea mai mică

29. Să se calculeze cel mai mare divizor comun a două numere date m și n prin următoarea metodă:

- pentru fiecare număr se creează un tablou cu toți divizorii acestuia, inclusiv divizorii banali;
- se selectează apoi într-un tablou divizorii comuni și se ia cel mai mare dintre aceștia.

9. Pointeri.

9.1. Operatori specifici pointerilor.

Un *pointer* este o variabilă care are ca valori adrese ale altor variabile, sau mai general adrese de memorie.

Un pointer este asociat unui tip de variabile, deci avem pointeri către **int**, **char**, **float**, etc.

În general o variabilă pointer **p** către tipul **T** se declară: **T *p;**

Un tip pointer la tipul **T** are tipul **T***.

Exemple:

```
int j, *pj; /*pj este o variabila de tip pointer la intregi*/
char c, *pc;
```

Se introduc doi noi operatori:

- *operatorul de adresare &* - aplicat unei variabile furnizează adresa acelei variabile

```
pj=&j; /* initializare pointer */
pc=&c;
```

Acstea inițializări pot fi făcute la definirea variabilelor pointeri:

```
int j, *pj=&j;
char c, *pc=&c;
```

O greșală frequentă comisă o reprezintă utilizarea unor pointeri neinițializați.

```
int *px;
*px=5; /* greșit, pointerul px nu este initializat (legat
          la o adresă de variabilă întreagă) */
```

Pentru a evita această eroare vom inițializa în mod explicit un pointer la **NULL**, atunci când nu este folosit.

- *operatorul de indirectare (derefențiere) ** – permite accesul la o variabilă prin intermediul unui pointer. Dacă **p** este un pointer de tip **T***, atunci ***p** este obiectul de tip **T** aflat la adresa **p**.

În mod evident avem:

```
*(&x) = x;
&(*p) = p;
```

Exemplu:

```
int *px, x;
x=100;
px=&x; // px contine adresa lui x
printf("%d\n", *px); // se afiseaza 100
```

Derefențierea unui pointer neinițializat sau având valoarea **NULL** conduce la o eroare la execuție.

9.2. Pointeri generici (pointeri void).

Pentru a utiliza un pointer cu mai multe tipuri de date, la declararea lui nu îl legăm de un anumit tip.

```
void *px; // pointerul px nu este legat de nici un tip
```

Un pointer nelegat de un tip nu poate fi dereferențiat.

Utilizarea acestui tip presupune conversii explicite de tip (cast). Exemplu:

```
int i;
void *p;
...
p=&i;
*(int*)p=5; // ar fi fost gresit *p=5
```

Exemplul 17: Definiți o funcție care afișează o valoare ce poate apartine uneia din tipurile: char, int, double.

```
#include <stdio.h>
enum tip {caracter, intreg, real};
void afisare(void *px, enum tip t) {
    switch(t) {
        case caracter:
            printf("%c\n", *(char*)px); break;
        case intreg:
            printf("%5d\n", *(int*)px); break;
        case real:
            printf("%6.2lf\n", *(double*)px); break;
    }
}

int main(){
    char c='X';
    int i=10;
    double d=2.5;
    afisare(&c, caracter);
    afisare(&i, intreg);
    afisare(&d, real);
}
```

9.3. Pointeri constanți și pointeri la constante.

In definițiile:

```
const int x=10,
*px=&x;
```

`x` este o constantă, în timp ce `px` este un pointer la o constantă. Aceasta însemnă că `x`, accesibil și prin `px` nu este modificabil (operațiile `x++` și `(*px)++` fiind incorecte, dar modificarea pointerului `px` este permisă (`px++` este corectă).

Un pointer constant (nemodificabil), se definește prin:

```
int y, * const py=&y;
```

In acest caz, modificarea pointerului (`py++`) nu este permisă, dar conținutul referit de pointer poate fi modificat (`(*py)++`).

Un pointer constant (nemodificabil) la o constantă se definește prin:

```
const int c=5, *const pc=&c;
```

In cazul folosirii unor parametri pointeri, pentru a preveni modificarea conținutului referit de aceștia se preferă definirea lor ca pointeri la constante. De exemplu o funcție care compară două siruri de caractere are prototipul:

```
int strcmp(const char *s, const char *d);
```

9.4. Operații aritmetice cu pointeri.

Asupra pointerilor pot fi efectuate următoarele operații:

- adunarea / scăderea unei constante
- incrementarea / decrementarea
- scăderea a doi pointeri de același tip

Prin incrementarea unui pointer legat de un tip `T`, adresa nu este crescută cu `1`, ci cu valoarea `sizeof(T)` care asigură adresarea următorului obiect de același tip.

În mod asemănător, `p + n` reprezintă de fapt `p+n*sizeof(T)`.

Doi pointeri care indică elemente ale aceluiași tablou pot fi comparați prin relația de egalitate sau neegalitate, sau pot fi scăzuți.

Pointerii pot fi comparați prin relațiile `==` și `!=` cu constanta simbolică `NULL` (definită în `stdio.h`).

9.5. Legătura între pointeri și tablouri.

Între pointeri și tablouri există o legătură foarte strânsă. Orice operație realizată folosind variabile indexate se poate obține și folosind pointeri.

În C numele unui tablou este un pointer constant la primul element din tablou: `x=&x[0]`

Numele de tablouri reprezintă pointeri constanți, deci nu pot fi modificați ca pointerii adevărați.

Exemplu:

```
int x[10], *px;
px=x; /* sunt operații permise */
px++;
x=px; /* sunt operații interzise, deoarece x este */
x++; /* pointer constant */
```

Prin urmare *variabilele indexate* pot fi transformate în *expresii cu pointeri* și avem echivalențele:

Tabel 8.2. Corespondență între tablouri și pointeri

Adresă		Valoare	
Notăție indexată	Notăție cu pointeri	Notăție indexată	Notăție cu pointeri
$\&x[0]$	x	$x[0]$	$*x$
$\&x[1]$	$x+1$	$x[1]$	$*(x+1)$
$\&x[i]$	$x+i$	$x[i]$	$*(x+i)$
$\&x[n-1]$	$x+n-1$	$x[n-1]$	$*(x+n-1)$

În C avem următoarea echivalență ciudată! Dacă x este un tablou de întregi

$$x[i] \equiv i[x]$$

$$\text{Într-adevăr: } x[i] = * (x+i) = * (i+x) = i[x]$$

9.6. Parametri tablouri.

Dacă în lista de parametri a unei funcții apare numele unui tablou cu o singură dimensiune se va transmite adresa de început a tabloului. Aceasta ne permite să nu specificăm dimensiunea tabloului, atât la definirea, cât și la apelul funcției.

Exemplul 18: Scrieți o funcție care calculează produsul scalar a doi vectori x și y , având câte n componente fiecare.

Antetul funcției va fi:

```
double scalar(int n, double x[], double y[])
```

Funcția poate fi declarată cu parametri formali pointeri în locul tablourilor:

```
double scalar(int n, double *x, double *y)
{ double P=0;
  for (int i=0; i<=n; i++)
    P=P+x[i]*y[i];
  return P;
}
```

Echivalentă cu:

```
double scalar(int n, double *x, double *y)
{ double P=0;
  for (int i=0; i<=n; i++)
    P=P+* (x+i) ** (y+i);
  return P;
}
```

9 Siruri de caractere.

9.1. Generalități.

O constantă sir de caractere se reprezintă intern printr-un tablou de caractere terminat prin caracterul nul '\0', memoria alocată fiind lungimea sirului + 1 (1 octet se adaugă pentru caracterul terminator al sirului).

Un tablou de caractere poate fi inițializat fără a-i specifica dimensiunea:

```
char salut []={ 'B', 'o', 'n', 'j', 'o', 'u', 'r', '!', '\0' };
```

sau mai simplu, specificând sirul între ghilimele:

```
char salut []="Bonjour!"
```

(tabloul este inițializat cu conținutul sirului de caractere -se alocă 9 octeți)

Folosirea unui pointer la un sir de caractere inițializat nu copiază sirul, ci are următorul efect:

- se alocă memorie pentru sirul de caractere, inclusiv terminatorul nul la o adresă fixă de memorie
- se inițializează spațiul cu valorile constantelor caractere
- se inițializează pointerul **Psalut** cu adresa spațiului alocat

```
char *Psalut="Buna ziua!" ;
```

(pointerul este inițializat să indice o constantă sir de caractere)

Așadar, în C nu există operația de atribuire de siruri de caractere (sau în general de atribuire de tablouri), ci numai atribuire de pointeri - atribuirea **t=s** nu copiază un tablou, pentru aceasta se folosește funcția **strcpy(t, s)**.

Pentru a ușura lucrul cu siruri de caractere, în biblioteca standard sunt prevăzute o serie de funcții, ale căror prototipuri sunt date în fișierele **<ctype.h>** și **<string.h>**.

În fișierul **<ctype.h>** există funcții (codificate ca macroinstructiuni) care primesc un parametru întreg, care se convertește în **unsigned char**, și întorc rezultatul diferit de 0 sau egal cu 0, după cum caracterul argument satisfac sau nu condiția specificată:

```
islower(c) 1 dacă c ∈ { 'a' .. 'z' }
isupper(c) 1 dacă c ∈ { 'A' .. 'Z' }
isalpha(c) 1 dacă c ∈ { 'A' .. 'Z' } ∨ { 'a' .. 'z' }
isdigit(c) 1 dacă c ∈ { '0' .. '9' }
isxdigit(c) 1 dacă c ∈ { '0' .. '9' } ∨ { 'A' .. 'F' } ∨ { 'a' .. 'f' }
isalnum(c) 1 dacă isalpha(c) || isdigit(c)
isspace(c) 1 dacă c ∈ { ' ', '\n', '\t', '\r', '\f', '\v' }
isgraph(c) 1 dacă c este afișabil, fără spațiu
isprint(c) 1 dacă c este afișabil, cu spațiu
iscntrl(c) 1 dacă c este caracter de control
ispunct(c) 1 dacă isgraph(c) && !isalnum(c)
```

Conversia din literă mare în literă mică și invers se face folosind funcțiile: **tolower(c)** și **toupper(c)**.

Exemplul 19: Scrieți o funcție care convertește un sir de caractere reprezentând un număr întreg, într-o valoare întreagă. Numărul poate avea semn și poate fi precedat de spații albe.

```
#include <ctype.h>
int atoi(char *s)
{ int i, nr, semn;
  for(i=0; isspace(s[i]); i++) /*ignora spatii albe*/
  ;
  semn=(s[i]=='-')?-1:1;      /*stabilire semn*/
  if(s[i]=='+'||s[i]=='-')    /*se sare semnul*/
    i++;
  for(nr=0;isdigit(s[i]);i++) /*conversie in cifra*/
    nr=10*nr+(s[i]-'0');     /*si alipire la numar*/
  return semn*nr;
}
```

Exemplul 20: Scrieți o funcție care convertește un întreg într-un sir de caractere în baza 10.

Algoritmul cuprinde următorii pași:

```
{ se extrage semnul numărului;
  se extrag cifrele numărului, începând cu cmps;
  se transformă în caractere și se depun într-un tablou;
  se adaugă semnul și terminatorul de sir;
  se inversează sirul;
}

void inversare(char[]);
void itoa(int n, char s[]){
  int j, semn;
  if((semn=n)<0)
    n=-n;
  j=0;
  do
    s[j++]=n%10+'0';
  while ((n/=10)>0);
  if(semn<0)
    s[j++]='-' ;
  s[j]='\0';
  inversare(s);
}
void inversare(char s[])
{ int i,j;
  char c;
  for(i=0,j=strlen(s)-1;i<j;i++,j--)
    c=s[i], s[i]=s[j], s[j]=c;
}
```

Exemplul 21: Scrieți o funcție care convertește un întreg fără semn într-un sir de caractere în baza 16.

Pentru a trece cu ușurință de la valorile cifrelor hexazecimale 0,1,...15 la caracterele corespunzătoare; '0','1',...,'a',...,'f', vom utiliza un tablou inițializat de caractere.

```
static char hexa[]="0123456789abcdef";
void itoh(int n, char s[])
```

```

{ j=0;
  do {
    s[j++]=hexa[n%16];
    while ((n/=16)>0);
    s[j]='\0';
    inversare(s);
}

```

Fișierul **<string.h>** conține prototipurile următoarelor funcții:

Tabel 9.1. Semnăturile funcțiilor din fișierul antet string.h

char* strcpy(char* d, const char* s)	copiază sirul s în d , inclusiv '\0', întoarce d
char* strncpy(char* d, const char* s, int n)	copiază n caractere din sirul s în d , completând eventual cu '\0', întoarce d
char* strcat(char* d, const char* s)	adaugă sirul s la sfârșitul lui d , întoarce d
char* strncat(char* d, const char* s, int n)	concatenează cel mult n caractere din sirul s la sfârșitul lui d , completând cu '\0', întoarce d
int strcmp(const char* d, const char* s)	compară sirurile d și s , întoarce -1 dacă d<s , 0 dacă d==s și 1 dacă d>s
int stricmp(const char* d, const char* s)	compară sirurile d și s (ca și strcmp()) fără a face distincție între litere mari și mici
int strncmp(const char* d, const char* s, int n)	similar cu strcmp() , cu deosebirea că se compară cel mult n caractere
int strcasecmp(const char* d, const char* s, int n)	similar cu strncmp() , cu deosebirea că nu se face distincție între literele mari și mici
char* strchr(const char* d, char c)	caută caracterul c în sirul d ; întoarce un pointer la prima apariție a lui c în d , sau NULL
char* strrchr(const char* d, char c)	întoarce un pointer la ultima apariție a lui c în d , sau NULL
char* strstr(const char* d, const char* s)	întoarce un pointer la prima apariție a subșirului s în d , sau NULL
char* strpbrk(const char* d, const char* s)	întoarce un pointer la prima apariție a unui caracter din subșirul s în d , sau NULL
int strspn(const char* d, const char* s)	întoarce lungimea prefixului din d care conține numai caractere din s
int strcspn(const char* d, const char* s)	întoarce lungimea prefixului din d care conține numai caractere ce nu apar în s
int strlen(const char* s)	întoarce lungimea lui s ('\0' nu se numără)
char* strlwr(char* s)	convertește literele mari în litere mici în s
char*strupr(char* s)	convertește literele mici în litere mari în s

<code>char* strtok(const char* d, const char* s)</code>	caută în d subșirurile delimitate de caracterele din s ; primul apel întoarce un pointer la primul subșir din d care nu conține caractere din s ; următoarele apeluri se fac cu primul argument NULL , întorcându-se de fiecare dată un pointer la următorul subșir din d ce nu conține caractere din s ; în momentul în care nu mai există subșiruri, funcția întoarce NULL
---	--

Ca exercițiu, vom codifica unele din funcțiile a căror prototipuri se găsesc în `<string.h>`, scriindu-le în două variante: cu tablouri și cu pointeri.

Exemplul 22: Scrieți o funcție având ca parametru un sir de caractere, care întoarce lungimea sirului

```
/*varianta cu tablouri*/
int strlen(char *s)
{ int j;
  for(j=0; s[j]; j++)
  ;
  return j;
}

/*varianta cu pointeri*/
int strlen(char *s)
{ char *p=s;
  while(*p)
    p++;
  return p-s;
}
```

Exemplul 23: Scrieți o funcție care copiază un sir de caractere **s** în **d**.

```
/*varianta cu tablouri*/
void strcpy(char *d, char *s)
{ int j=0;
  while((d[j]=s[j])!='\0')
    j++;
}

/*varianta cu pointeri*/
void strcpy(char *d, char *s)
{ while((*d=*s)!='\0'){
    d++;
    s++;
}}
```

Postincrementarea pointerilor poate fi făcută în operația de atribuire deci:

```
void strcpy(char *d, char *s) {
  while((*d++=*s++) !='\0')
```

```

    ;
}
```

Testul față de '\0' din **while** este redundant, deci putem scrie:

```
void strcpy(char *d, char *s) {
    while(*d++=*s++)
        ;
}
```

Exemplul 24: Scrieți o funcție care compară lexicografic două siruri de caractere și întoarce rezultatul -1, 0 sau 1 după cum d<s, d==s sau d>s.

```
/*varianta cu tablouri*/
int strcmp(char *d, char *s)
{ int j;
    for(j=0; d[j]==s[j]; j++)
        if(d[j]=='\0')
            return 0;
    return d[j]-s[j];
}

/*varianta cu pointeri*/
int strcmp(char *d, char *s)
{ for(; *d==*s; d++, s++)
    if(*d=='\0')
        return 0;
    return *d-*s;
}
```

Exemplul 25: Scrieți o funcție care determină poziția (indexul) primei aparitii a unui subșir s într-un sir d. Dacă s nu apare în d, funcția întoarce -1.

```
int strind(char d[], char s[]) {
    int i,j,k;
    for (i=0; d[i]; i++) {
        for (j=i, k=0; s[k] && d[j]==s[k]; j++, k++)
            ;
        if (k>0 && s[k]=='\0')
            return i;
    }
    return -1;
}
```

9.2. Funcții de intrare / ieșire relative la siruri de caractere.

Pentru a citi un sir de caractere de la intrarea standard se folosește funcția **gets()** având prototipul:

```
char *gets(char *s);
```

Funcția **gets()** citește caractere din fluxul standard de intrare **stdin** în zona de memorie adresată de pointerul **s**. Citirea continuă până la întâlnirea sfârșitului de linie. Marcajul de sfârșit de linie nu este copiat, în locul lui fiind pus caracterul nul ('\0'). Funcția întoarce adresa zonei de memorie în

care se face citirea (adică **s**) sau **NULL**, dacă în locul șirului de caractere a fost introdus marcajul de sfârșit de fișier.

Pentru a scrie un șir de caractere terminat prin caracterul **NULL**, la ieșirea standard **stdout**, se folosește funcția:

```
int puts(char *s);
```

Caracterul terminator nu este transmis la ieșire, în locul lui punându-se marcajul de sfârșit de linie. Caracterele citite într-un tablou ca un șir de caractere (cu **gets()**) pot fi convertite sub controlul unui format folosind funcția:

```
int sscanf(char *sir, char *format, adrese_var_formatate);
```

Singura deosebire față de funcția **scanf()** constă în faptul că datele sunt preluate dintr-o zonă de memorie, adresată de primul parametru (și nu de la intrarea standard).

Exemplul 26: Scrieți o funcție care citește cel mult n numere reale, pe care le plasează într-un tablou x. Funcția întoarce numărul de valori citite.

Vom citi numerele într-un șir de caractere **s**. De aici vom extrage în mod repetat câte un număr, folosind funcția **sscanf()** și îl vom converti folosind un format corespunzător. Ciclul se va repeta de **n** ori, sau se va opri când se constată că s-au terminat numerele.

Vom scrie funcția în 2 variante: folosind tablouri sau folosind pointeri.

```
/* varianta cu tablouri */
int citreal(int n, double x[])
{ char s[255];
  int j;
  double y;
  for (j=0; j<n; j++) {
    if(gets(s)==NULL)
      return j;
    if(sscanf(s,"%lf",&y)!=1) /*conversie in real*/
      break;                  /*s-au terminat numerele*/
    x[j]=y;
  }
  return j;
}

/* varianta cu pointeri */
int citreal(int n, double *px)
{ int j=0;
  double y;
  double *p=px+n;
  while(px<p) {
    if(gets(s)==NULL)
      return j;
    if(sscanf(s,"%lf",&y)!=1) /*conversie in real*/
      break;                  /*s-au terminat numerele*/
    *px++=y;
    j++;
  }
}
```

```
return j;
}
```

9.3. Tablouri de pointeri.

Un tablou de pointeri este definit prin: **tip *nume [dimensiune];**

Exemplul 27: Să se sorteze o listă de nume.

Folosirea unui tablou de şiruri de caractere este lipsită de eficiență, deoarece şirurile sunt de lungimi diferite. Vom folosi un tablou de pointeri la şiruri de caractere. Prin sortare nu se vor schimba şirurile de caractere, ci pointerii către acestea.

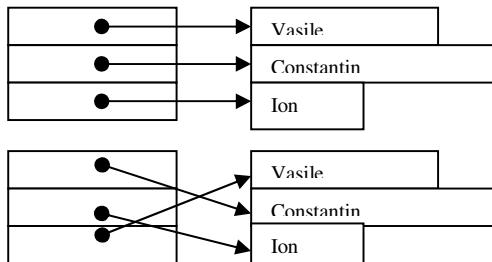


Fig.9.1. Sortarea şirurilor de caractere

Citirea şirurilor de caractere presupune:

- rezervarea de spațiu pentru şiruri
- inițializarea tabloului de pointeri cu adresele şirurilor

Pentru rezervarea de spațiu se folosește funcția **char *strdup(char *s);** care:

- salvează şirul indicat de **s** într-o zonă de memorie disponibilă, alocată dinamic
- întoarce un pointer către zona respectivă sau **NULL**.

Citirea numelor este terminată prin **EOF**. Funcția de citire întoarce numărul de linii citite:

```
int citire(char *tabp[]) {
    int j=0;
    char tab[80];
    while(1) {
        gets(tab);
        if(tab==NULL)
            break;
        tabp[j]=strdup(tab);
    }
    return j;
}
```

Sortarea o vom realiza cu algoritmul bulelor: dacă şirul de nume ar fi ordonat, atunci două nume consecutive s-ar afla în relația < sau ==. Vom căuta aşadar relațiile >, schimbând de fiecare dată între ei pointerii corespunzători (schimbare mai eficientă decât schimbarea şirurilor). Se fac mai multe parcurgeri ale listei de nume; la fiecare trecere, o variabilă martor – **sortat**, inițializată la 1 este pusă pe 0, atunci când se interschimbă doi pointeri. Lista de nume va fi sortată în momentul în care în urma unei parcurgeri a listei se constată că nu s-a mai făcut nici o schimbare de pointeri.

```
void sortare(char *tp[], int n) {
    int j, sortat;
```

```

char *temp;
for(sortat=0; !sortat;){
    sortat=1;
    for(j=0; j<n-1; j++)
        if(strcmp(tp[j], tp[j+1])>0){
            temp=tp[j],
            tp[j]=tp[j+1],
            tp[j+1]=temp,
            sortat=0;
        }
    }
void afisare(char *tp[], int n){
    int j;
    for (j=0; j<n; j++)
        if(tp[j])
            puts(tp[j]);
}
int main()
{ int n;
    char *nume[100];
    n=citire(nume);
    sortare(nume, n);
    afisare(nume, n);
}

```

Exemplul 28: Definiți o funcție, având ca parametru un întreg, reprezentând o lună, care întoarce (un pointer la) numele acelei luni

```

char *nume_luna(int n)
{ static char *nume[]={ "Luna inexistentă", "Ianuarie", "Februarie",
    "Martie", "Aprilie", "Mai", "Iunie", "Iulie", "August",
    "Septembrie", "Octombrie", "Noiembrie", "Decembrie" };
    return (n<1 || n>12)?nume[0]:nume[n];
}

```

Exemplul 29: Scrieți un program care extrage cuvintele distincte dintr-un text, scriind în dreptul fiecărui cuvânt numărul de apariții ale acestuia în text.

Pentru separarea cuvintelor din text vom folosi funcția **strtok()**. Un cuvânt separat din text este mai întâi căutat în tabloul de pointeri la cuvintele separate **cuv**, și este inserat acolo, în caz că nu este găsit; dacă este găsit este crescut contorul de apariții al cuvântului.

```

#include <stdio.h>
#include <string.h>
#define NC 100

/*cauta sirul p in tabloul de siruri cuv
intoarce pozitia in care se afla p in cuv sau -1 */
int exista(char* p, int nc, char* cuv[]){
    int j=0;

```

```

for(int j=0; j<nc; j++)
    if(strcmp(p,cuv[j])==0) return j;
return -1;
};

int main(){
    char sep[]=".,:;-()\t\n"; /*separatori intre cuvinte*/
    char linie[80];           /*tampon pentru citirea unei liniilor*/
    char *cuv[NC], ap[NC], *p; /*tablou de cuvinte*/
    int nc=0, poz;
    freopen("text.txt", "r", stdin);
    while(gets(linie))
        for(p=strtok(linie,sep); p; p=strtok(0,sep))
            if((poz=exista(p, nc, cuv))==-1){
                cuv[nc]=strdup(p);
                ap[nc++]=1;
            }
            else
                ap[poz]++;
    printf("numar cuvinte distincte %3d\n", nc);
    for(int i=0; i<nc; i++)
        printf("%10s %2d\n", cuv[i], ap[i]);
}

```

9.4. Probleme rezolvate.

1. Se citesc mai multe siruri de caractere reprezentand numere lungi in baza 16, ce pot avea pana la 100 de cifre. Sa se calculeze suma acestora, ignorand numerele incorecte.

Indicatie: Pentru a aduna doua numere lungi păstrate in doua siruri de caractere, se inverseaza in prealabil cifrele lor, astfel incat prima cifra sa fie cea mai putin semnificativa si se transforma apoi caracterele cifre hexadecimale in numere intregi de la 0 la 15. Se extinde apoi cu zerouri, numarul lung cel mai scurt, aducandu-l la lungimea celuilalt si se face adunarea rang cu rang, cu propagarea transportului:

```

sum = s_i + a_i + t
s_i = sum % 16
t = sum / 16

```

Daca in final ramane transport, acesta devine cifra cea mai semnificativa a sumei..

Numerele din rangurile sumei se transforma in cifre hexadecimale si se inverseaza aceste cifre.

1. initializare numar lung sumă
2. bucla citire numere lungi
 - 2.1. verificare corectitudine număr lung citit
 - 2.2. conversie caractere cifre hexa in numere
 - 2.3. inversare cifre numare lung
 - 2.4 completare numar mai scurt cu zerouri
 - 2.5. adunare pe ranguri cu propagare transport
 - 2.6. completare suma cu ultimul transport
 - 2.7. inversare cifre
 - 2.8. conversie numere hexa in caractere
3. afisare numar lung sumă

2. O fracție rațională are numărătorul și numitorul numere lungi în baza 10, având până la 100 cifre. Se cere să se simplifice fracția cu divizori întregi primi, până la o limită dată **n**.

Numerele lungi se citesc ca două siruri de caractere, din primele două linii; a treia linie conține valoarea lui **n**.

Programul va afișa fracția nesimplificată și fracția simplificată.

Indicație: Se va defini o funcție care împarte un număr lung **a**, cu un întreg **d** și calculează câtul **c** – un număr lung și restul – un întreg mai mic ca **d**.

$$\frac{a_0 a_1 \dots a_{n-1}}{d} = c_0 c_1 \dots c_{n-1} + \frac{r}{d}$$

rest deîmpărțit împărțitor cât

↓ ↓ ↓ ↓

int divlung(char *a, int d, char *c);

În prealabil, caracterele cifre ale deîmpărțitului se transformă în întregi.

Funcția simulează împărțirea cifră cu cifră; într-un pas al împărțirii:

- se adaugă la restul parțial (initializat la 0) cifra curentă a deîmpărțitului: **rp=10*rp+a_i**
- se calculează cifra curentă a câtului: **c_i = rp / d**
- se actualizează restul parțial: **rp = rp % d**

Ultimul rest parțial este restul împărțirii.

Câtul **c₀c₁...c_{n-1}** are aceeași lungime cu deîmpărțitul, dar cifrele sale cele mai semnificative pot fi 0.

În final, cifrele întregi ale câtului se transformă în caractere cifre.

1. încercare divizori primi
2. simplificare cu divizorul comun
3. afișare fracție simplificată
4. funcția divlung()
 - 4.1. conversie caractere în cifre
 - 4.2. simulare împărțire cifră cu cifră
 - 4.3. conversie cifre în caractere

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int divlung(char*, int, char*);
int main(){
    char a[100], b[100], ca[100], cb[100];
    int i, n;
    gets(a); puts(a);
    gets(b); puts(b);
    scanf("%d", &n);
    for(i=0; i<=n; i++)
        if(divlung(a, i, ca)==0 && divlung(b, i, cb)==0){
            strcpy(a, ca);
            strcpy(b, cb);
        }
    printf("%s\n", a);
}
```

```

printf("%s\n", b);
system("PAUSE");
return 0;
}

int divlung(char *a, int d, char *c) {
    int i, l, rp = 0;
    l = strlen(a);
    for(i=0; i < l; i++)
        a[i] -= '0';
    for(i=0; i<l; i++){
        rp = 10*rp + a[i];
        c[i] = rp / d;
        rp %= d;
    }
    for(i=0; i < l; i++)
        c[i] += '0';
    return rp;
}

```

Probleme propuse (Şiruri de caractere).

1. Scrieți o funcție C care stabilește dacă un sir de caractere dat ca parametru reprezintă un palindrom. Pentru aceasta este necesar ca primul și ultimul caracter din sirul de caractere să fie egale, al doilea și penultimul, s.a.m.d.
Funcția întoarce 1 dacă sirul de caractere este un palindrom și 0 în caz contrar.
2. Un text citit de pe mediul de intrare reprezintă un program C. Să se copieze pe mediul de ieșire, păstrând structura liniilor, dar suprimând toate comentariile.
3. Dintr-un text, citit de pe mediul de intrare, să se separe toate cuvintele, plasându-le într-un vector cu elemente siruri de caractere de lungime 10 (cuvintele mai scurte se vor completa cu spații libere, iar cele mai lungi se vor trunchia la primele 10 caractere.
Se vor afișa elemenele acestui tablou, câte 5 elemente pe o linie, separate între ele prin 2 asteriscuri.
4. Dintr-un text citit de pe mediul de intrare să se afișeze toate cuvintele care conțin cel puțin 3 vocale distințe.
5. Scrieți un program care citește și afișează un text și determină numărul de propoziții și de cuvinte din text. Fiecare propoziție se termină prin punct, iar în interiorul propoziției cuvintele sunt separate prin spații, virgulă sau liniuță.
6. De pe mediul de intrare se citește un text format din cuvinte separate prin spații libere. Să se afișeze acest text la ieșire, păstrând structura liniilor și scriind în dreptul liniei cel mai lung cuvânt din linie. Dacă mai multe cuvinte au aceeași lungime maximă, va fi afișat numai primul dintre ele.
7. Scrieți un program care citește de la intrarea standard cuvinte, până la întâlnirea caracterului punct și afișează câte un cuvânt pe o linie, urmat de despărțirea acestuia în silabe. Se utilizează următoarele reguli de despărțire în silabe:

- o consoană aflată între două vocale trece în silaba a doua
- în cazul a două sau mai multe consoane aflate între două vocale, prima rămâne în silaba întâia, iar celelalte trec în silaba următoare.
- Nu se iau în considerare excepțiile de la aceste reguli.

8. Să se transcrie la ieșire un text citit de la intrarea standard, suprimând toate cuvintele de lungime mai mare ca 10. Cuvintele pot fi separate prin punct, virgulă sau spații libere și nu se pot continua de pe o linie pe alta.

9. Scrieți un program care citește de la intrarea standard un text terminat prin punct și îl transcrie la ieșirea standard, înlocuind fiecare caracter ‘*’ printr-un număr corespunzător de spații libere care ne poziționează la următoarea coloană multiplu de 5. Se va păstra structura de linii a textului.

10. Scrieți un program pentru punerea în pagină a unui text citit de la intrarea standard. Se fac următoarele precizări:

- cuvintele sunt separate între ele prin cel puțin un spațiu
- un cuvânt nu se poate continua de pe o linie pe alta
- lungimea liniei la ieșire este N
- lungimea maximă a unui cuvânt este mai mică decât **N / 2**

In textul rezultat se cere ca la început de linie să fie un început de cuvânt, iar sfârșitul de linie să coincidă cu sfârșitul de cuvânt. În acest scop, spațiile se distribuie uniform și simetric între cuvinte. Face excepție doar ultima linie. Caracterul punct apare doar la sfârșit de text.

11. Modificați funcția **strind()** astfel încât să întoarcă în locul indexului un pointer și **NULL** în caz că că subșirul **s** nu apare în sirul destinație **d** (adică scrieți funcția **strstr()**).

10. Alocarea dinamică a memoriei.

10.1. Funcții pentru gestiunea dinamică a memoriei.

Utilizatorul poate solicita în timpul execuției programului alocarea unei zone de memorie. Această zonă de memorie poate fi eliberată, în momentul în care nu mai este necesară. Alocarea și eliberarea de memorie la execuție permite gestionarea optimă a memoriei.

Biblioteca standard oferă 4 funcții, având prototipurile în `<alloc.h>` și `<stdlib.h>`. Acestea sunt:

```
void *malloc(unsigned n);
```

Funcția alocă un bloc de memorie de **n** octeți. Funcția întoarce un pointer la începutul zonei alocate. În caz că cererea de alocare nu poate fi satisfăcută, funcția returnează **NULL**.

```
void *calloc(unsigned nelem, unsigned dim);
```

Alocă **nelem*dim** octeți de memorie (**nelem** blocuri formate din **dim** octeți fiecare). Întoarce un pointer la începutul zonei alocate sau **NULL**. Memoria alocată este inițializată cu zerouri.

```
void free(void *p);
```

Funcția eliberează o zonă de memorie indicată de **p**, alocată în prealabil prin `malloc()` sau `calloc()`.

```
void *realloc(void *p, unsigned dim);
```

Modifică dimensiunea spațiului alocat prin pointerul **p**, la **dim**. Funcția întoarce adresa zonei de memorie realocate, iar pointerul **p** va adresa zona realocată.

- dacă dimensiunea blocului realocat este mai mică decât a blocului inițial, **p** nu se modifică, iar funcția va întoarce valoarea lui **p**.
- dacă **dim==0** zona adresată de **p** va fi eliberată și funcția întoarce **NULL**.
- dacă **p==NULL**, funcția alocă o zonă de **dim** octeți (echivalent cu `malloc()`).

Funcțiile de alocare întorc pointeri generici (**void***) la zone de memorie, în timp ce utilizatorul alocă memorie ce păstrează informații de un anumit tip. Pentru a putea accesa memoria alocată, indirect, prin intermediul pointerului, acesta va trebui să fie un pointer cu tip, ceea ce impune conversia explicită (prin cast) a pointerului întors de funcția de alocare într-un pointer cu tip.

De exemplu, pentru a aloca un vector de întregi, având **n** elemente vom folosi:

```
int *p;
if (p=(int*)malloc(n*sizeof(int))==NULL) {
    printf("Memorie insuficientă\n");
    exit(1);
}
```

Funcțiile care întorc pointeri sunt utile în alocarea de spațiu pentru *variabile dinamice*. Variabilele dinamice sunt alocate în momentul execuției, nu au nume și sunt accesate prin pointeri.

Un *constructor* este o funcție care alocă spațiu în mod dinamic pentru o variabilă și întoarce un pointer la spațiul rezervat.

Un *destructor* este o funcție care primește un pointer la o zonă alocată dinamic și eliberează această zonă.

O funcție utilă, **strdup()** – salvează un sir de caractere într-o zonă alocată dinamic și întoarce un pointer la acea zonă sau **NULL**.

```
char *strdup(char *s) {
    char *p;
    p=(char*) malloc(strlen(s)+1);
    if (p!=NULL)
        strcpy(p,s);
    return p;
}
```

Exemplul 29: *Citiți de la intrarea standard un sir de caractere de lungime necunoscută într-un vector alocat dinamic. Alocarea de memorie se va face progresiv, în incremente de lungime INC, după citirea a INC caractere se face o reallocare.*

```
char *citire()
{ char *p, *q;
  int n;
  unsigned dim=INC;
  p=q=(char*)malloc(dim);
  for(n=1; (*p=getchar())!=='\n' && *p!=EOF; n++) {
      if(n%INC==0) {
          dim+=INC;
          p=q=realloc(q,dim);
          p+=n;
          continue;
      }
      p++;
  }
  *p='\0';
  return realloc(q,n);
}
```

Fișierul `<string.h>` ne pune la dispoziție o serie de funcții care ne permit să lucrăm cu zone de memorie care conțin date de tip nespecificat. Acestea vor fi considerate octeți (caractere):

Tabel 10.1. Funcții pentru lucru cu zone de memorie cu conținut nespecificat

void* memcpy(void* d, const void* s, int n)	copiază n octeți din s în d ; întoarce d
void* memmove(void* d, const void* s, int n)	ca și memcpy , folosită dacă s și d se întrepătrund
void* memset(void* d, const int c, int n)	copiază caracterul c în primele n poziții din d
int memcmp(const void* d, const void* s, int n)	compară zonele adresate de s și d

Probleme rezolvate.

1. O școală are **nc** clase (**nc > 12**, datorită existenței claselor paralele). Clasele paralele sunt identificate suplimentar cu o majusculă (de exemplu 8A, 8B, etc). Clasele au număr diferit și cunoscut de elevi. Numele elevilor din fiecare clasă sunt de asemenea cunoscute. Dându-se un nume, să se stabilească în ce clasă este acesta (de exemplu 9H). Datele problemei sunt:

- un sir de 12 caractere, în care caracterul **i** specifică ultima dintre clasele paralele **i+1** (de exemplu dacă **s[4]** este **C** atunci avem 3 clase a 5-a: 5A, 5B și 5C).
- mai mulți întregi specificând numărul elevilor din fiecare clasă
- numele elevului căutat
- numele celorlalți elevi, introduse, câte unul pe linie, în ordinea claselor.

Mentionăm că nu se dă explicit **nc** – numărul total de clase. Alocarea de memorie pentru tablouri se face dinamic.

1. citire sir ultime clase
2. determinarea numărului de clase **nc**
3. citire nume elev căutat
4. citirea numărului de elevi din fiecare clasă
5. alocare memorie pentru păstrare nume elevi
6. citire nume elevi
7. căutare elev și determinare număr clasă
8. formare nume clasă pe baza numărului
9. afișare nume elev și nume clasă

2. Să se întocmească și să se afișeze un index de cuvinte, pe baza liniilor citite de la tastatură. Acesta este un tablou (alocat dinamic, întrucât nu îi cunoaștem dimensiune de la început), în care sunt puse toate cuvintele distincte din text de lungime > **1gmin**. Fiecare cuvânt va fi însoțit de numărul său de apariții, și de numerele liniilor în care a apărut. Dacă un cuvânt apare de mai multe ori într-o linie, numărul de apariții va fi actualizat corespunzător, dar numărul liniei va apărea o singură dată.

Valoarea **1gmin** se citește înaintea textului. Între cuvintele din text pot exista ca separatori spații albe, virgulă, punct două puncte și punct-virgulă .

1. citire lgmin, inițializare contor linii
2. buclă citire linii
 - bucă separare cuvinte din linie
 - verificare lungime cuvânt separat
 - căutare cuvânt în index
 - dacă e găsit
 - actualizare nr.apariții
 - eventuala actualizare nr.linii
 - dacă nu e găsit
 - creerea unei noi intrări în index
 - copiere cuvânt
 - actualizare nr. apariții
 - actualizare nr. linii
3. afișare tablou index

10.2. Probleme propuse.

11. Funcții (2).

11.1. Mecanisme de transfer ale parametrilor.

În limbajele de programare există două mecanisme principale de transfer ale parametrilor: **transferul prin valoare** și **transferul prin referință**.

În C parametrii se transferă numai prin valoare- aceasta înseamnă că parametrii actuali sunt copiați în zona de memorie rezervată pentru parametrii formali. Modificarea parametrilor formali se va reflecta asupra copiilor parametrilor actuali și nu afectează parametrii actuali, adică **parametrii actuali nu pot fi modificați** !

Astfel funcția:

```
void schimb(int a, int b)
{ int c=a;
  a=b;
  b=c;
}
```

nu produce interschimbul parametrilor actuali **x** și **y** din funcția **main()**:

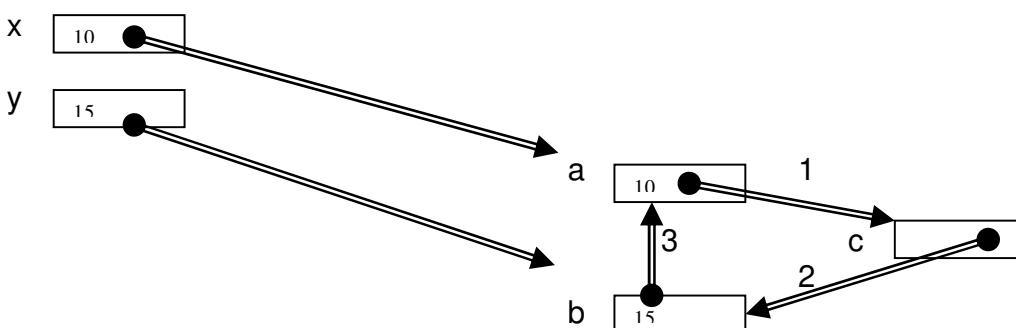
```
int main()
{ int x=10,y=15;
  printf("%d\t%d\n", x, y);
  schimb(x, y);
  printf("%d\t%d\n", x, y);
}
```

Se afișează:

10 15
10 15

Stiva funcției apelante

Stiva funcției apelate



În cazul transferului prin referință, pentru modificarea parametrilor actuali, funcției i se transmit nu valorile parametrilor actuali, ci adresele lor.

Forma corectă a funcției **schimb()** obținută prin **simularea transferului prin referință cu pointeri** este:

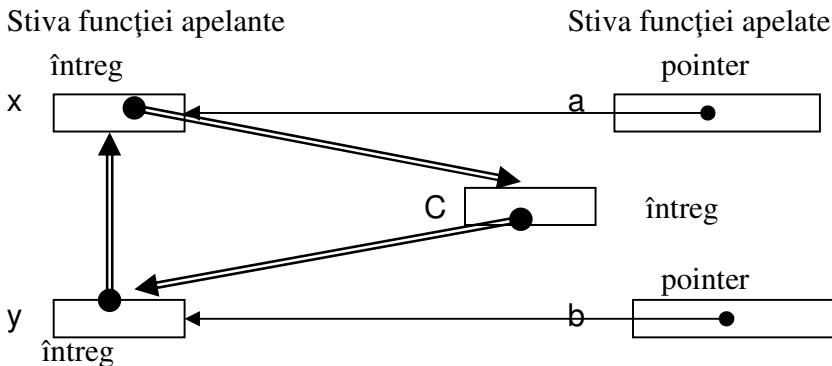
```
void schimb(int *a, int *b)
{ int c;
  c = *a;
  *a = *b;
```

```

    *b = c;
}

int main()
{ int x=10, y=15;
  printf("%d\t%d\n", x, y);
  schimb(&x, &y);
  printf("%d\t%d\n", x, y);
}

```



In consecință, pentru a transmite un rezultat prin lista de parametri (adică pentru a modifica un parametru) va trebui să declarăm parametrul formal ca pointer.

Tablourile sunt transmise întotdeauna prin referință, adică un parametru formal tablou reprezintă o adresă (un pointer) și anume adresa de început a tabloului.

11.2. Funcții care întorc pointeri.

Funcția `strcpy()` întoarce adresa sirului destinație:

```

char *strcpy(char *d, char *s)
{ char *p=d;
  while (*p++=*s++)
  ;
  return d;
}

```

Aceasta ne permite ca simultan cu copierea să calculăm lungimea sirului copiat:

```
n=strlen((strcpy)d, s));
```

Dacă funcția întoarce adresa unei variabile locale, atunci aceasta trebuie să fie în mod obligatoriu în clasa **static**.

De asemenei nu trebuie să întoarcă adrese ale unor parametri, deoarece aceștia sunt transmiși prin stivă.

Să considerăm ca exemplu, două funcții: una care evaluează un polinom într-un punct, cealaltă care calculează derivata unui polinom. A doua funcție întoarce un pointer la tabloul coeficienților polinomului derivat. Acest tablou, declarat local funcției de derivare nu poate reprezenta rezultatul întors de funcție, decât dacă este alocat static, nu în stivă, ci în zona de date. În funcția `main()` evaluăm polinomul derivat într-un punct, compunând apelurile celor două funcții, în lista parametrilor funcției de evaluare apare apelul funcției de derivare.

```

#include <stdio.h>
#include <stdlib.h>
double peval(int, double*,double); //evaluare polinom
double *pderiv(int, double*, int*); //derivare polinom
int main()
{
    int i, na, nb;
    double a[]={2.,5.,-3.,4.}; // polinomul  $2x^3+5x^2-3x+4$ 
    na = sizeof(a)/sizeof(a[0])-1; //gradul polinomului
    printf("%4.0lf\n", peval(nb, pderiv(na,a,&nb), 1.));
    getchar();
    return 0;
}
double peval(int n, double *a,double x){
    int i;
    double p=a[0];
    for(i=1; i<=n; i++)
        p = x*p + a[i];
    return p;
}
double *pderiv(int n, double *a, int *nd){
    static double b[10];
    int i;
    for(i=0; i<n; i++)
        b[i] = (n-i)*a[i];
    *nd = n-1;
    return b;
}

```

11.3. Pointeri la funcții.

Numele unei funcții reprezintă adresa de memorie la care începe funcția. Numele funcției este, de fapt, un pointer la funcție.

Se poate stabili o corespondență între variabile și funcții prin intermediul pointerilor la funcții. Ca și variabilele, acești pointeri:

- pot primi ca valori funcții;
- pot fi transmiși ca parametrii altor funcții
- pot fi întoși ca rezultate de către funcții

La declararea unui pointer către o funcție trebuie să precizăm toate informațiile despre funcție, adică:

- tipul funcției
- numărul de parametri
- tipul parametrilor

care ne va permite să apelăm indirect funcția prin intermediul pointerului.

Declararea unui pointer la o funcție se face prin:

```
tip (*pf)(listă_parametri_formali);
```

Dacă nu s-ar folosi paranteze, ar fi vorba de o funcție care întoarce un pointer.

Apelul unei funcții prin intermediul unui pointer are forma:

```
(*pf)(listă_parametri_actuali);
```

Este permis și apelul fără indirectare:

```
pf(listă_parametri_actuali);
```

Este posibil să creem un tablou de pointeri la funcții; apelarea funcțiilor, în acest caz, se face prin referirea la componentele tabloului.

De exemplu, inițializarea unui tablou cu pointeri cu funcțiile matematice uzuale se face prin:

```
double (*tabfun[]) (double) = {sin, cos, tan, exp, log};
```

Pentru a calcula rădăcina de ordinul 5 din e este suficientă atribuirea:

```
y = (*tabfun[3])(0.2);
```

Numele unei funcții fiind un pointer către funcție, poate fi folosit ca parametru în apeluri de funcții. În acest mod putem transmite în lista de parametri a unei funcții – numele altrei funcții.

De exemplu, o funcție care calculează integrala definită:

$$I = \int_a^b f(x) dx$$

va avea prototipul:

```
double integrala(double, double, double(*)(double));
```

Exemplul : Definiți o funcție pentru calculul unei integrale definite prin metoda trapezelor,cu un număr fixat n de puncte de diviziune:

$$\int_a^b f(x)dx = h \left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(a + ih) \right] \quad \text{cu} \quad h = \frac{b - a}{n}$$

Folosiți apoi această funcție pentru calculul unei integrale definite cu o precizie dată ε. Această precizie este atinsă în momentul în care diferența între două integrale, calculate cu n, respectiv 2n puncte de diviziune este inferioară lui ε

```
#include <math.h>
double sinxp(double x){ return sin(x*x); }
double trapez(double, double, int, double(*)());
double a=0.0, b=1.0, eps=1E-6;
int N=10;
int main()
{ int n=N;
  double In,I2n,vabs;
  In=trapez(a,b,n,sinxp);
  do { n*=2;
    I2n=trapez(a,b,n,sinxp);
    if((vabs=In-I2n)<0) vabs=-vabs;
    In=I2n;
  } while(vabs > eps);
  printf("%6.2lf\n", I2n);
}
double trapez(double a,double b,int n,double(*f)())
{ double h,s;
  int i;
```

```

h=(b-a)/n;
for(s=0.0,i=1;i<n;i++)
    s+=(*f)(a+i*h);
s+=((*f)(a)+(*f)(b))/2.0;
s*=h;
return s;
}

```

11.4. Declaratii complexe (șarade).

O declaratie complexă este o combinație de pointeri, tablouri si functii. In acest scop se folosesc **atributele**:

() – functie

[] – tablou

***** - pointer

care pot genera urmatoarele *combinatii*:

*** ()** – funcție ce returnează un pointer

(*) () – pointer la o funcție

*** []** - tablou de pointeri

(*) [] – pointer la tablou

[] [] – tablou bidimensional

Există și **combinatii incorecte**, provenite din faptul că în C nu este permisă declararea:

- unui tablou de funcții
- unei funcții ce returnează un tablou.

Acestea sunt:

() [] – funcție ce returnează un tablou

[] () – tablou de funcții

() () – funcție ce returnează o funcție

Pentru a interpreta o declarație complexă vom inlocui *atributele* prin următoarele *șabloane text*:

Atribut	Şablon text
()	funcția returnează
[n]	tablou de n
*	pointer la

Descifrarea unei declarații complexe se face aplicând **regula dreapta – stânga**, care presupune următorii pași:

- se incepe cu identificatorul
- se caută în dreapta identificatorului un atribut
- dacă nu există, se caută în partea stângă
- se substituie atributul cu şablonul text corespunzător
- se continuă substituția dreapta-stânga
- se oprește procesul la întâlnirea tipului datei.

De exemplu:

```

int (* a[10]) ( );
                    tablou de 10
                    pointeri la
                    funcții ce returnează int

```

```
double (**pf) () ) [3] [4];
pointer la
o funcție ce returnează un pointer
la un tablou cu 3 linii și 4 coloane de double
```

In loc de a construi declarații complexe, se preferă, pentru creșterea clarității, să definim progresiv noi tipuri folosind **typedef**. Reamintim că declarația **typedef** asociază un nume unei definiri de tip:

```
typedef <definire de tip> identificator;
```

Exemplu:

```
typedef char *SIR; /*tipul SIR=sir de caractere*/
typedef float VECTOR[10];/*tipul VECTOR=tablou de 10float*/
typedef float MATRICE[10][10];/*tipul MATRICE= tablou de 10x10
float*/
typedef double (*PFADRD) (double);/*tipul PFADRD=pointer la
functie de argument double si rezultat double */
```

Vom putea folosi aceste tipuri noi în definirea de variabile:

```
SIR s1, s2;
VECTOR b, x;
MATRICE a;
PFADRD pf1;
```

11.5. Probleme propuse.

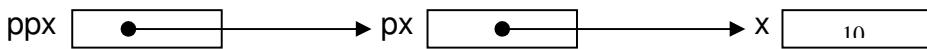
12. Tablouri și pointeri (2).

12.1. Pointeri la pointeri.

Adresa unei variabile pointer va fi de tip pointer către pointer (pointer dublu)
Să considerăm definițiile:

```
int x=10, *px=&x, **ppx=&px;
```

care corespund situației:



pentru a obține valoarea 10 putem folosi **x**, ***px** sau ****ppx**.

O funcție care interschimbă doi pointeri are forma:

```
void pschimb(int **pa, int **pb)
{ int *ptemp;
  ptemp=*pa;
  *pa=*pb;
  *pb=ptemp;
}
```

cu apelul:

```
int *px, *py;
pschimb(&px, &py);
```

Deoarece un tablou este accesat printr/un pointer, tablourile de pointeri pot fi accesate cu pointeri dubli:

```
char *a[10];
char **pa;
pa = a;
```

Funcția de afișare a șirurilor de caractere adresate de un tablou de pointeri poate fi rescrisă ca:

```
void afisare(char **tp, int n)
{ while(n--)
    printf("%s\n", *tp++);
}
```

12.2. Tablouri multidimensionale.

Un tablou cu mai multe dimensiuni se definește prin:

```
tip nume [d1] [d2] ... [dn];
```

Referirile la elemente unui tablou cu mai multe dimensiuni se fac folosind variabile indexate de forma: **nume[i1][i2]...[in]**, în care **0<=ik<=dk-1**

Un tablou cu **n** dimensiuni poate fi considerat ca un tablou cu o dimensiune având ca elemente tablouri cu **n-1** dimensiuni.

Elementele tabloului ocupă o zonă continuă de memorie de:

d1 x d2 x...x dn x sizeof(T) octeți.

Adresa în memorie a unui element **a[i1][i2]...[in]** este dată de funcția de alocare:

```
&a[i1][i2]...[in]=a+sizeof(T)*[i1*d2*...*dn+i2*d3*...*dn+...+in]
```

În cazul vectorilor: **`&a[i]=a+sizeof(T)*i`**, ceea ce ne permite ca la declararea unui parametru vector să nu specificăm dimensiunea tabloului.

În cazul matricilor, compilatorul le transformă în vectori:

`&a[i][j]=a+sizeof(T)*(i*c+j)`

Și în cazul matricelor, ca și la vectori putem înlocui indexarea prin operații cu indici și avem:

`a[i][j] = (*(&a+i))[j]=*((*(&a+i)+j)`

La transmiterea unui tablou multidimensional ca parametru al unei funcții vom omite numai prima dimensiune, celelalte trebuind să fie specificate.

Prin urmare, prototipul unei funcții de afișare a unei matrici având **1** linii și **c** coloane nu poate fi scris ca:

`void matprint(int a[][], int l, int c);`

ci:

`void matprint(int a[][DMAX], int l, int c);`

în care **DMAX** este numărul de coloane al matricii din apel, ceea ce ne limitează utilizarea funcției numai pentru matrici cu **DMAX** coloane!

Vom reda generalitatea funcției de afișare a matricilor, declarând matricea parametru ca un vector de pointeri:

```
void matprint(int (*a) [], int l, int c)
{ int i, j;
  for(i=0; i<l; i++)
  { for (j=0; j<c; j++)
    printf("%4d", ((int*)a)[i*c+j]);
    printf("\n");
  }
}
```

Problema transmiterii matricilor ca parametri poate fi evitată, dacă le linearizăm, transformându-le în vectori. În acest caz, în locul folosirii a doi indici **i** și **j** vom folosi un singur indice:

`k=i*c+j`

Exemplul 30: Scrieți o funcție pentru înmulțirea a două matrici **A** și **B** având **m**x**n**, respectiv **n**x**p** elemente.

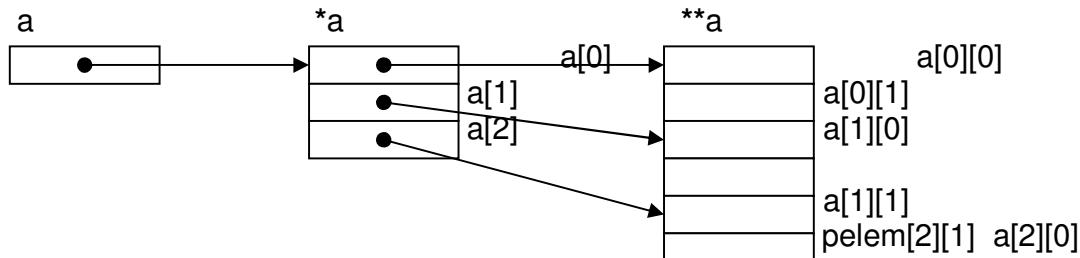
Matricea produs va avea **m**x**p** elemente, care vor fi calculate cu relația:

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik}B_{kj}$$

```
void matprod(int m, int n, int p, double A[], double B[], double C[])
{ int i, j, k, ij;
  for (i=0; i<m; i++)
  { for (j=0; j<p; j++) {
    ij=i*p+j;
    for (k=0; k<n; k++)
      C[ij]=C[ij]+A[i*n+k]*B[k*p+j];
    }
}
```

Soluția propusă nu ne permite să accesăm elementele matricelor folosind 2 indici. Am putea înlocui matricea printr-un vector de pointeri la liniile matricei.

Exemplul 31: Definiți o funcție care alocă dinamic memorie pentru o matrice având linii și coloane.



```
double **allocmat(int lin, int col)
{   double **a;
    int i;
    a=(double**)calloc(lin, sizeof(double*));
    if(a==(double**)NULL) return NULL;
    for(int i=0; i<lin; i++){
        a[i]=(double*)calloc(lin, sizeof(double));
        if(a[i]==(double*)NULL) return NULL;
    }
    return a;
}
```

Eliberarea memoriei se face în ordine inversă: mai întâi se eliberează memoria ocupată de elementele din liniile matricei și apoi pointerii la liniile.

```
void elibmat(double **a, int lin){
    for(int i=0; i<lin; i++)
        free(a[i]);
    free(a);
}
```

12.3. Probleme propuse.

1. Să se construiască un patrat magic de dimensiune **n** (cu **n** impar), adică o matrice cu **n** linii și **n** coloane având elemente numerele naturale **1, 2, ..., n²** astfel încât sumele elementelor pe linii, pe coloane și pe cele două diagonale să fie identice.

2 .Să se stabilească dacă există elemente comune tuturor liniilor unei matrici date. Se vor afișa câte asemenea elemente sunt, care sunt acestea și apoi se va indica ce poziție ocupă acestea în fiecare linie.

3. O matrice pătrată a are **n** linii și **n** coloane. Cele două diagonale determină patru zone notate 1,2,3,4 care nu includ elementele de pe diagonale.

Să se calculeze mediile geometrice ale elementelor pozitive din zonele 1 și 2. Dacă media nu se poate calcula, se va afișa un mesaj corespunzător.

Să se calculeze procentajul de elemente strict pozitive din zona 3 și numărul de elemente divizibile

cu 5 din zona 4. Dacă nu există elemente cu proprietățile cerute se va afișa un mesaj corespunzător.

4. Se dau două matrici **A** și **B** pătrate, de ordin **n**. Să se stabilească dacă cele două matrici sunt sau nu una inversa celeilalte. În acest scop se creează matricea produs **C=A*B** și se verifică dacă aceasta este matricea unitate.

5. Dintr-o matrice **A**, având **n** linii și **n** coloane să se afișeze liniile care reprezintă siruri ordonate crescător și coloanele care reprezintă siruri ordonate descrescător.

6. O matrice **a** are **p** linii și **q** coloane. Să se creeze o nouă matrice **b**, din matricea **a**, exceptând liniile și coloanele la intersecția cărora se află elemente nule. Se vor utiliza doi vectori în care se vor marca liniile, respectiv coloanele care nu vor apărea în **b**.

7. Se consideră o matrice **A** cu **p** linii și **q** coloane, de elemente reale. Să se creeze pe baza acesteia o nouă matrice **B** având **m** coloane cu elementele pozitive din **A** și un vector **C** cu elementele negative din matricea **A**.

8. Se dă o matrice **A** pătrată, cu **n** linii și **n** coloane. Să se facă schimbările de linii și de coloane astfel încât elementele diagonalei principale să fie ordonate crescător.

9. Să se calculeze coeficienți polinomului Cebâșev de ordinul **n**, pornind de la relația de recurență

$$\begin{aligned} T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x), & k > 2 \\ T_0(x) &= 1, \quad T_1(x) = x \end{aligned}$$

obținând în prealabil relații de recurență pentru coeficienți.

$$\overline{\overline{x^T y}} = \sum_{i=0}^{n-1} x_i y_i$$

10. a) Să se definească o funcție care calculează produsul scalar a doi vectori, adică:

b) Să se definească o procedură care calculează produsul diadic a doi vectori:

$$\overline{\overline{\overline{x^T y}}} = \begin{bmatrix} x_0 y_0 & x_0 y_1 & \cdots & x_0 y_{n-1} \\ x_1 y_0 & x_1 y_1 & \cdots & x_1 y_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n-1} y_0 & x_{n-1} y_1 & \cdots & x_{n-1} y_{n-1} \end{bmatrix}$$

c) Să se scrie un program care citește: un număr întreg **n** (**n** <= 10), o matrice pătrată **A** cu **n** linii și coloane și doi vectori **u** și **v** cu câte **n** componente și calculează matricea **B**, în care:

$$B = A - u.v' / u'.v$$

11. Să se scrie un program care citește un număr întreg **n** și o matrice pătrată **A** cu **n** linii și coloane și afișează numerele liniilor având în prima poziție elementul minim și în ultima poziție - elementul maxim din linie. Se vor afișa de asemenea numerele coloanelor având în prima poziție elementul maxim și în ultima elementul minim.

12. Să se realizeze un program care simulează jocul "viață", adică trasează populația unei comunități de organisme vii, prin generarea de nașteri și morți timp de G generații.

Comunitatea de organisme este descrisă printr-o matrice cu N linii și N coloane, fiecare element reprezentând o celulă care poate fi vidă sau poate conține un organism. Fiecare celulă din rețea, exceptându-le pe cele de la periferie are 8 vecini.

Nașterile și morțile de organisme se produc simultan la începutul unei noi generații. Legile genetice care guvernează creșterea și descreșterea populației sunt:

- fiecare celulă vidă care este adiacentă la 3 celule ocupate va da naștere în următoarea generație la un organism
- fiecare celulă care conține un organism ce are numai un vecin sau niciunul, va muri la începutul generației următoare (datorită izolării)
- orice organism dintr-o celulă cu patru sau mai mulți vecini în generația prezentă va muri la începutul generației următoare (datorită suprapopulației).

13. Să se scrie în C:

a) o funcție care verifică dacă 2 linii date i și j dintr-o matrice pătrată ($n \times n$) sunt identice sau nu.

b) o funcție care afișează numerele liniilor și coloanelor dintr-o matrice pătrată, unde se află elemente nule (zero).

c) un program care citește o matrice pătrată cu maxim 30 de linii și coloane de numere întregi, verifică dacă există sau nu 2 linii identice în această matrice, folosind funcția de la punctul a).

Dacă toate liniile sunt distințe, atunci se afișează poziția tuturor elementelor nule din matrice, folosind funcția de la punctul b)

14. Să se înmulțească două matrici utilizând o funcție pentru calculul produsului scalar a doi vectori.

15. Se dă o matrice A având L linii și C coloane ($L, C \leq 10, C > 3$) de elemente întregi.

Să se afișeze liniile în care există cel puțin trei elemente având minim cinci divizori nebanali. Se va defini și utiliza o funcție care stabilește câți divizori nebanali are un număr dat.

16. Să se definească o funcție care calculează diferența între elementul maxim și elementul minim ale unei liniilor date dintr-o matrice.

Să se scrie un program care citeste: numerele naturale l și c ($1, c \leq 10$), valoarea reală eps și matricea A având $l \times c$ elemente și afișează liniile din matrice care au diferență între extreime inferioară valorii eps .

17. Într-o matrice dată A cu L linii și C coloane să se permute circular dreapta fiecare linie i cu i pozitii. Se va utiliza o funcție care permute circular dreapta cu o poziție componentele unui vector.

18. Pentru o matrice dată A cu L linii și C coloane să se afișeze toate cuplurile (i, j) reprezentând numere de linii având elementele respectiv egale. Se va defini și utiliza o funcție care stabilește dacă doi vectori sunt sau nu egali.

19. Se dă două matrici **A** și **B** având n linii și coloane fiecare. Să se stabilească dacă una dintre ele este sau nu inversa celeilalte. Se va defini și se va utiliza o funcție pentru a înmulți două matrici.

20. Un punct să într-o matrice este un element maxim pe coloană și minim pe linia pe care se află sau minim pe coloană și maxim pe linia sa. Utilizând funcții care verifică dacă un element este minim/maxim pe linia/coloana sa să se determine punctele în care dintr-o matrice cu elemente distințe.

21. Doi vectori **x** și **y** cu câte n componente fiecare, ($n \leq 20$) se află în relația $\mathbf{x} \leq \mathbf{y}$ dacă $x_i \leq y_i$, pentru $i := 0..n-1$

a) Să se definească o funcție care primind ca parametri două linii **i** și **k** ale unei matrici, stabilește dacă acestea se află în relația \leq .

b) Să se definească o funcție care, primind ca parametri numerele a două linii dintr-o matrice calculează diferența lor, depunând rezultatul într-un vector.

c) Să se scrie un program care citește un număr întreg n ($n \leq 20$) și o matrice cu n linii și coloane și afișează pentru toate perechile de linii care nu se găsesc în relația \leq diferența lor.

22.a) Să se definească o funcție care stabilește dacă o linie specificată a unei matrici este sau nu o secvență ordonată crescător.

b) Să se definească o funcție care, pentru o linie specificată a unei matrici determină elementul minim și elementul maxim din acea linie.

c) Se citește o matrice patrată **A** cu n linii și coloane ($n \leq 10$). Să se afișeze pentru fiecare linie, care nu reprezintă un sir de valori crescătoare: numărul liniei, elementul maxim și elementul minim.

Indicație: O linie **i** dintr-o matrice reprezintă o secvență ordonată crescător dacă:

$\mathbf{A}_{i,j} \leq \mathbf{A}_{i,j+1}$, pentru $j = 1:n-1$

21. Să se definească o funcție care stabilește dacă doi vectori date ca parametri sunt sau nu ortogonali.

Să se scrie un program care citește o matrice patrată cu n linii și coloane și stabilește dacă matricea este sau nu ortogonală pe linii, și în caz afirmativ calculează matricea inversă. Se știe că pentru o matrice ortogonală, matricea inversă se obține transpunând matricea dată și împărțind fiecare coloană cu norma euclidiană a ei (radicalul produsului scalar al coloanei cu ea însăși). O matrice este ortogonală, dacă oricare două linii diferite sunt ortogonale.

13. Structuri.

13.1. Definirea tipurilor structurilor și declararea variabilelor structuri.

Structura este o colecție de variabile (care vor fi numite în continuare *membri*, *câmpuri* sau *selectori*) grupate sub un singur nume. Structura este eterogenă, adică poate cuprinde membri de tipuri diferite. Câmpurile structurii se declară prin nume și tip (ca variabilele). Declararea unei structuri se face prin:

```
struct nume_structură { declaratii_câmpuri; };
```

După acolada închisă din declarația câmpurilor se pune întotdeauna ; .

Prin declararea unei structuri nu se rezervă memorie ci se definește un tip. Exemple:

```
struct complex {double re; double im;};
struct data {int an; int luna; int zi;};
struct persoana { char* nume;
    char* strada;
    int numar;
    int varsta;};
struct student { struct persoana stud;
    char* grupa;
    int an;};
```

În cazul în care câmpurile unei structuri sunt pointeri, este necesar să alocăm dinamic memorie pentru datele referite de acei pointeri.

Alocarea de memorie se face în momentul definirii unor variabile structuri, folosind numele dat structurii:

```
struct nume_structură lista_variabile;
```

Exemple:

```
struct complex z1, z2;
struct data d;
struct persoana p;
struct student s;
```

Declararea structurii poate fi combinată cu definirea variabilelor de tip structură:

```
struct nume_structură {declaratii campuri;} lista_variabile;
```

Exemple:

```
struct complex { double re; double im;} z1, z2;
struct data {int an; int luna; int zi;} d;
```

Numele structurii poate lipsi, practică pe care nu o recomandăm:

```
struct { declaratii_campuri; } lista_variabile;
```

Exemplu:

```
struct {double re; double im;} z1, z2;
```

În C, folosirea declarației **typedef** ne permite o utilizare mai comodă a structurilor:

```
typedef tip nume;
```

```
struct complex {double re; double im;};
typedef struct complex COMPLEX;
COMPLEX z1, z2;
```

O formă combinată a declarării structurii cu **typedef** este următoarea:

```
typedef struct {double re; double im;} COMPLEX;
COMPLEX z1, z2;
```

13.2. Inițializarea structurilor.

Inițializarea variabilelor structuri se poate face la definirea lor, valorile inițiale fiind cuprinse între acolade.

Exemple:

```
COMPLEX z={1.5,-2.0};
struct persoana p={"Popescu Ion", "Popa Nan", 72, 53};
```

Dacă structurile au câmpuri tablouri sau alte structuri, la inițializare se pot folosi mai multe niveluri de acolade.

Exemple:

```
typedef struct stud {char *nume;
                    int note[5];
                    data nast;} STUDENT;
STUDENT s={"Popa Vasile", {9,7,10,8,6}, {1980,9,15}};
```

13.3. Accesul la câmpurile structurilor.

Pentru a obține acces la membrii unei structuri vom folosi operatorul punct:

variabilă_structură.nume_membru

Exemple:

```
COMPLEX z;
struct persoana p;
struct student s;
z.re=1.5;
z.im=-2.0;
p.nume = strdup("Popa Vasile");
s.stud.nume = strdup("Vasilescu Paul");
```

13.4. Pointeri la structuri.

Considerăm definițiile de pointeri la structuri:

```
COMPLEX z, *pz=&z;
struct persoana p, *pp=&p;
```

Adresarea la membrii structurilor prin intermediul pointerilor se face dereferențind pointerii:

```
(*pz).re=1.5;
(*pp).nume=strdup("Popescu Ion");
```

sau folosind un nou operator de selecție indirectă ->:

```
pz->re=1.5;
pp->nume=strdup("Popescu Ion");
```

13.5. Atribuirile de structuri.

Două variabile având același tip structură pot apărea ca termeni în atribuirile:

```
COMPLEX z1, z2, *pz;
z1=z2;
pz=&z2;
z1=*pz;
```

Am văzut că nu putem copia tablouri, și în particular, siruri de caractere prin atribuire. Totuși, este posibil să realizăm atribuirea pe siruri de caractere, declarându-le ca structuri:

```
typedef struct {char x[100];} STRING;
STRING s1, s2;
s1=s2; // !!
```

Atribuirea unei structuri reprezintă o copiere bit cu bit.

13.6. Structuri și funcții.

O funcție poate întoarce ca rezultat:

- o structură

```
// functie ce întoarce o structura
COMPLEX cmplx(double x, double y)
{ COMPLEX z;
  z.re=x;
  z.im=y;
  return z;
}
```

- un pointer la o structură

```
// functie ce întoarce un pointer la o structura
COMPLEX *cmplx(double x, double y)
{ COMPLEX z, *pz = &z; //functioneaza corect
  pz->re=x;
  pz->im=y;
  return pz;
}
```

- un membru al unei structuri

```
// functie ce întoarce un membru al structurii
double real(COMPLEX z)
{ return z.re;
}
```

Parametrii unei funcții pot fi:

- structuri

```
// functie ce întoarce o structura
```

```
COMPLEX conjg(COMPLEX z)
{ COMPLEX w;
  w.re=z.re;
  w.im=-z.im;
  return w;
}

• pointeri la structuri

void conjg(COMPLEX *pz) {
  py->im=-pz->im;
}
```

Funcția următoare initializează, prin citire, o structură de tip **persoană** (tip definit mai sus). Funcția își poate realiza efectul:

- prin modificarea unei structuri adresate printr-un pointer

```
void date_pers(struct persoana *p)
{ char buf[80];
  p->nume=strdup(gets(buf));
  p->strada=strdup(gets(buf));
  scanf("%d", &(p->numar));
  scanf("%d", &(p->varsta));
}
```

- prin rezultatul structură întors de funcție:

```
struct persoana date_pers(void)
{ struct persoana p;
  char buf[80];
  p.nume=strdup(gets(buf));
  p.strada=strdup(gets(buf));
  scanf("%d", &(p.numar));
  scanf("%d", &(p.varsta));
}
```

Exemplu 32: Să se calculeze numărul de zile dintre două date calendaristice.

Vom incrementa în mod repetat cea mai mică dintre date, până când devine egală cu cea de-a doua.

```
#include <stdio.h>
typedef struct {int an,luna,zi;} DATA;
int citdata(DATA *);
int nrzile(DATA, DATA);
int dateegale(DATA, DATA);
int precede(DATA, DATA);
void incrdata(DATA *);
int bisect(DATA);
int main(){
  int n;
  DATA d1, d2;
  citdata(&d1);
```

```

    citdata(&d2);
    if ((n=nrzile(d1, d2))>0)
        printf("intre cele doua date sunt %d zile\n",n);
    else
        printf("prima dintre date este dupa cealalta\n");
    return 0;
}

int citdata(DATA *d) {
    if (scanf("%d %d %d", &d->an, &d->luna, &d->zi)==3 &&
        d->an > 0 &&
        d->luna > 0 && d->luna < 13 &&
        d->zi > 0 && d-> zi < 32)
        return 1;
    else
        { printf("Data incorecta\n");
        return 0;
        }
}
}

int dateegale(DATA d1, DATA d2){
    return(d1.an==d2.an && d1.luna==d2.luna && d1.zi==d2.zi);
}

int precede(DATA d1, DATA d2){
    return(d1.an < d2.an ||
           d1.an==d2.an && d1.luna < d2.luna ||
           d1.an==d2.an && d1.luna==d2.luna && d1.zi < d2.zi);
}

int bisect(DATA d){ return (d.an%4==0 && d.an%100!=0 || d.an
%400==0);}

int nrzile(DATA d1, DATA d2){
    int i=0;
    if (precede(d1,d2)) {
        while(!dateegale(d1,d2)) {
            incrdata(&d1);
            i++;
        }
        return i;
    }
    else
        return 0;
}

void incrdata(DATA *d){
    static
    ultima[2][13]={{0,31,28,31,30,31,30,31,31,30,31,30,31},

```

```

{0,31,29,31,30,31,30,31,31,30,31,30,31,30,31};

if(d->zi < ultima[bisect(*d)][d->luna])
    d->zi++;
else
{ d->zi=1;
  if(d->luna < 12)
    d->luna++;
  else
{ d->luna=1;
  d->an++;
}
}
}

```

13.7. Uniuni.

Folosirea uniunilor permite suprapunerea în același spațiu de memorie a mai multor variabile. Conținutul zonei de memorie poate fi interpretat în mod diferit.

Definirea unei uniuni se face ca și a unei structuri. Uniunea poate fi considerată o structură, în care toți membrii au deplasamentul 0 față de adresa de început. Uniunea este suficient de mare pentru a conține cel mai voluminos membru, respectând restricțiile de aliniere.

De exemplu pentru a interpreta conținutul unei zone de memorie ca un întreg, un real sau un sir de caractere, le vom suprapune într-o uniune:

```
union supra {int i; double d; char* c;};
union supra u;
```

sau

```
typedef union {int i; double d; char* c;} SUPRA;
SUPRA u;
```

Accesul se face ca în cazul structurilor:

```
u.i; // pentru intreg (primii 2 octeti)
u.d; // pentru real (primii 8 octeti)
u.c[0]; // pentru primul octet caracter
```

Exemplul 33: Definiți o funcție care afișează reprezentarea internă a unui număr real.

```
void hexdouble(double x)
{ union {double d; char c;} u;
  unsigned n=sizeof(double);
  char *p = &(u.c);
  u.d=x;
  while(n--)
    printf("%02x ", *p++);
}
```

13.8. Probleme propuse.

1. Scrieți o funcție având ca parametri două date calendaristice (precizate prin an, lună și zi), care stabilește una din situațiile:

- Prima dată o precede pe cea de-a doua
- Cele două date sunt egale
- A doua dată o precede pe prima

Funcția va întoarce una din valorile -1, 0, 1.

2. La o disciplină cu notare pe parcurs, fiecărui student i s-au acordat 3 note la trei lucrări de control și un calificativ pentru activitatea la seminar (insuficient, suficient, bine, foarte bine). Pe baza acestor informații se acordă o notă finală în felul următor: se face media celor 3 note la care se adaugă 0 pentru insuficient, 0.25 pentru suficient, 0.5 pentru bine și 0.75 pentru foarte bine, și apoi rezultatul se trunchiază.

- Să se definească tipul situație ca o structură având ca membri câmpurile nume – un sir de 20 de caractere
note – un tablou de 3 întregi
calificativ – un caracter
- Să se definească o funcție având ca parametru o situație, care calculează nota finală.
- Să se scrie un program care citește situațiile a n studenți și afișează lista studenților promovați (având nota finală ≥ 5).

3. Definiți tipul mulțime de reali ca o structură cu următoarele câmpuri:

- numărul de elemente (o valoare întreagă)
- valorile elementelor (un tablou de reali, având cel mult 100 de elemente)

Se vor defini funcții pentru:

- a stabili dacă o valoare dată x aparține sau nu unei mulțimi date M
- crearea mulțimii diferență a două mulțimi.
- Calculul valorii unui polinom (definit printr-o structură identică cu a mulțimii de reali) într-un punct dat x
- Calculul polinomului derivat. Această funcție are doi parametri structuri: polinomul dat și polinomul derivat și nu întoarce rezultat.

Funcția **main()** :

- citește un polinom dat prin gradul **n** și cei **n+1** coeficienți
- citește cele r posibile rădăcini ale polinomului
- stabilește pentru fiecare rădăcină multiplicitatea ei.

Indicație: Componentele **x[k]** pentru care $P(x[k]) = 0$ sunt rădăcini cu multiplicitate cel puțin 1, cele pentru care $P'(x[k]) = 0$ sunt rădăcini cu multiplicitate cel puțin 2, și.a.m.d. Rădăcinile simple se obțin făcând diferența dintre mulțimea rădăcinilor cu multiplicitate cel puțin 1 și cele cu multiplicitate cel puțin 2, etc.

4. a) Definiți structurile:

- **complex** – având doi membri reali (partea reală și partea imaginară;

- **polinom** – cu membrii: **grad** – un întreg, **coeficienti** – un tablou de valori complexe.

b) Definiți funcții pentru: adunarea și înmulțirea a două numere complexe și pentru calculul valorii unui polinom într-un punct dat.

- c) Scrieți o funcție **main()** care:

- citește un polinom, de grad cel mult 20
- citește un număr complex **z**
- calculează și afișează valoarea polinomului în punctul **z**.

5. Un polinom este precizat printr-o structură având câmpurile: **grad** – un întreg și **coeficienti** – un tablou de reali. Vom considera gradul cel mult 20.

- a) Definiți o funcție care calculează valoarea unui polinom într-un punct dat. Funcția are ca parametri un polinom **P** și un real **x** și întoarce ca rezultat un real – valoarea **P(x)**.
- b) Definiți o funcție care derivează un polinom. Funcția are ca parametri două structuri: polinomul dat și polinomul derivat și nu întoarce nimic.

c) Scrieți o funcție **main()** care:

- Citește un polinom **P**
- Citește o valoare reală **x**
- Stabilește și afișează multiplicitatea rădăcinii **x** a polinomului **P**.

6. a) Să se definească tipurile **punct**, **dreptunghi**, **cerc** și **nor** de puncte ca tipuri înregistrare. Tipul **punct** va avea drept câmpuri abscisa **x** și ordonata **y** a punctului, tipul **cerc** va avea drept câmpuri **centrul** - un punct și **raza** - o valoare reală, tipul **dreptunghi** - colturile stânga-jos și dreapta-sus a două vârfuri opuse ale dreptunghiului, care sunt puncte, iar tipul **nor de puncte** - numărul de puncte din nor și coordonatele lor.

b) Să se definească funcții având ca parametri un **punct** și un **dreptunghi** (respectiv un **cerc**), care stabilesc dacă punctul este sau nu interior dreptunghiului (cercului).

Să se scrie un program care citește: un nor de puncte, un dreptunghi și un cerc și care stabileste câte puncte din nor sunt interioare dreptunghiului și câte cercului și afișează aceste informații.

7. Pentru aprovizionarea unui magazin se lansează **n** comenzi (**n <= 100**). O comandă este precizată prin două elemente: **tipul** produsului comandat (un tablou de 8 caractere) și **cantitatea** comandată (o valoare întreagă). Un produs poate fi comandat de mai multe ori. Să se centralizeze comenzile pe produse, astfel încât comenzile centralizate să se refere la produse diferite.

8. Să se rezolve ecuația $a \cdot x^3 + b \cdot x^2 + c \cdot x + d = 0$, cu $a, b, c, d \in \mathbb{R}$, $a \neq 0$, folosind formulele lui Cardan.

9. O matrice rară, adică o matrice având majoritatea elementelor nule se memorează economic într-o înregistrare conținând: numărul de linii, numărul de coloane, numărul de elemente nenule, precum și doi vectori, unul cu elementele nenule din matrice, iar celălalt cu pozițiile lor, facând vectorizarea matricii pe linii.

Să se definească funcții pentru adunarea și înmulțirea a două matrici rare, precum și pentru crearea structurii matricei rare și afișarea acesteia ca o matrice.

Se va scrie un program care citește și afișează două matrici rare și care apoi le adună și le înmulțește, afișând de fiecare dată rezultatele.

10. a) Să se definească tipul punct ca tip înregistrare.

b) Să se definească o funcție având ca parametri trei puncte care stabilește dacă acestea sunt sau nu coliniare.

c) Să se scrie un program care citește un întreg **n** (**n** <= 50) și **n** puncte și afișează numerele tripletelor de puncte coliniare.

11. Un bilet pronosport conține numele jucatorului și 13 caractere **1**, **2**, **x** constituind reprezentarea codificată a rezultatelor unor meciuri de fotbal.

a) Să se descrie structura unui bilet pronosport folosind tipul înregistrare.

b) Să se definească o funcție având ca parametri doi vectori de același tip, funcție care stabilește numărul de componente egale din cei doi vectori.

• c) Să se scrie un program care primește ca date **n** bilete pronosport, precum și rezultatele a 13 meciuri jucate (codificate 1, 2, x) și afiseaza:

lista jucatorilor, pentru fiecare indicându-se numărul de pronosticuri exacte

• lista jucatorilor, ordonată după numărul de pronosticuri exacte indicate.

Numele jucatorilor se citesc începând din coloana 1, iar pronosticurile din 21.

12. Un experiment fizic este precizat prin: numarul de determinări și valorile măsurate.

a) Să se descrie structura experiment folosind tipul înregistrare.

b) Să se definească o funcție având ca parametru un experiment, care calculează media aritmetică a măsurătorilor.

c) Să se scrie un program care citește numărul de determinări și valorile lor și creează cu acestea o înregistrare și calculează folosind funcția de mai sus abaterea standard:

$$s = \sqrt{\frac{\sum_{i=0}^{n-1} x_i^2 - n \sum_{i=0}^{n-1} \left(\frac{x_i}{\bar{x}}\right)^2}{n - 1}}$$

13. a) O dată calendaristică este exprimată prin trei valori întregi: anul, luna și ziua; o persoană este precizată prin: nume și prenume (maxim 30 de caractere) și data nașterii - o dată calendaristică. Să se descrie tipurile dată și persoană ca tipuri înregistrare.

b) Să se definească o funcție având ca parametru o persoană, funcție care calculează vârstă persoanei în ani împliniți.

c) Să se scrie un program care citește o listă de **n** persoane (**n** este citit înaintea listei) și datele lor de naștere și folosind funcția definită mai sus afișează lista persoanelor majore.

14. a) Să se descrie tipurile punct și dreaptă ca tipuri înregistrare.

b) Să se definească o funcție având ca parametri două drepte, un punct și o variabilă booleană, funcție care stabilește dacă cele două drepte se intersecțează, caz în care calculează coordonatele

punctului de intersecție, sau sunt paralele; parametrul boolean separă situația drepte paralele/concurente.

c) Să se scrie un program care citește **n** drepte (**n <= 100**) și afișează perechile de drepte paralele, iar pentru fiecare pereche de drepte neparallele - coordonatele punctului de intersecție. De exemplu:

Drepte paralele:

1 – 3
2 – 4

Drepte concurente:

1 – 2	(8.0, 7.0)
1 – 4	(17.0, 5.0)
2 – 3	(4.0, 4.0)
3 – 4	(13.0, 2.0)

15. a) Să se definească tipurile punct și triunghi ca tipuri înregistrare.
- b) Să se definească o funcție având ca parametru un triunghi funcție care calculează aria acestuia.
- c) Să se definească o funcție având ca parametri un punct și un triunghi, funcție care stabilește dacă punctul este interior sau exterior triunghiului. (dacă punctul **M** este interior triunghiului **ABC** atunci **aria(ABC)=aria(MAB)+aria(MBC)+aria(MCA)**).
- d) Să se scrie un program care citește 4 puncte și determină folosind funcția de la punctul c) dacă acestea pot forma un patrulater convex.

14. Fișiere.

14.1. Operațiile de intrare / ieșire.

In limbajul C nu există instrucțiuni de intrare / ieșire. Operațiile de intrare / ieșire sunt realizate prin apelul unor funcții ale sistemului de operare. Acestea sunt implementate prin funcții, sub o formă compatibilă pentru diversele sisteme de operare (sunt portabile).

Un *fișier* este o colecție ordonată de **articole** (înregistrări) păstrate pe un suport extern de memorie și identificate printr-un nume.

Pentru ***fișierul standard de intrare***, datele sunt introduse de la tastatură.

Pentru ***fișierul standard de ieșire***, rezultatele sunt afișate pe terminalul standard de ieșire.

Mesajele de eroare se afișează în ***fișierul standard de eroare***.

Fișierul are un articol care marchează ***sfârșitul fișierului***. Pentru fișierul standard de intrare de la tastatură, ***sfârșitul de fișier***, pentru sistemele de operare DOS și Windows se generează prin **Ctrl-Z** (pentru Unix – prin **Ctrl-D**).

Operațiile specifice prelucrării fișierelor sunt.

- deschiderea unui fișier
- închiderea unui fișier
- crearea unui fișier
- citirea de articole din fișier (consultarea fișierului)
- actualizarea (sau modificarea) fișierului
- adăugare de articole la sfârșitul fișierului
- poziționarea în fișier.
- ștergerea unui fișier
- schimbarea numelui unui fișier

Prelucrarea fișierelor se face pe două niveluri:

- nivelul inferior, care apelează direct la sistemul de operare.
- nivelul superior, care utilizează structuri speciale **FILE**

Funcțiile de pe nivelul superior nu asigură o independență totală față de sistemul de operare.

Funcțiile standard de intrare / ieșire au prototipurile în fișierul antet **<stdio.h>**.

14.2. Fișiere text și fișiere binare

Într-un fișier text, toate datele sunt memorate ca siruri de caractere, organizate pe linii, separate între ele prin marcajul sfârșit de linie '**\n**' .

Într-un fișier text spațiul de memorare pe disc nu este folosit în mod eficient pentru datele numerice (astfel întregul **12345** ocupă 5 octeți).

Într-un fișier binar, datele sunt păstrate în formatul lor intern (2 octeți pentru **int**, 4 octeți pentru **float**, etc).

La fișierele text **marcajul de sfârșit de fișier** (caracterul **0x1A**) există fizic în fișier. La întâlnirea acestui caracter funcția **fgetc()** întoarce **EOF (-1)**. Marcajul de sfârșit de fișier se generează de la tastatură prin **Ctrl-Z**.

În cazul *fișierelor binare*, marcajul de sfârșit de fișier nu există fizic în fișier, ci este generat de funcția **fgetc()**.

În MSDOS (și în Unix), la nivelul liniei de comandă intrările și ieșirile standard pot fi redirectate în fișiere disc, fără a opera nici o modificare la nivelul programului. Astfel:

- < redirecțează intrarea standard către fișierul specificat
- > redirecțează ieșirea standard către fișierul specificat

Fișierul standard de eroare nu poate fi redirectat.

Fișierul specificat poate fi:

con – pentru consola sistem (tastatura, respectiv ecranul)

prn – pentru imprimanta paralelă

com1 – pentru interfața serială de date

nume_fișier – pentru un fișier disc

NUL – pentru perifericul nul.

Exemple:

> **test.exe > prn** redirecțează ieșirea programului la imprimantă

> **test.exe < f1.dat > f2.dat** redirecțează atât intrarea cât și ieșirea programului

14.3. Accesul la fișiere.

Fișierele disc și fișierele standard sunt gestionate prin pointeri la structuri specializate **FILE**, care se asociază fiecărui fișier pe durata prelucrării.

Fișierele standard au pointerii predefiniți: **stdin**, **stdout**, **stderr**, **stdprn**, **stdaux**.

Declararea unui pointer la fișier se face prin:

FILE *pf;

1. deschiderea unui fișier:

Înainte de a fi prelucrat, un fișier trebuie să fie deschis. Prin deschidere:

- se asociază unui *nume de fișier* un *pointer la fișier*
- se stabilește un *mod de acces* la fișier

Pentru deschiderea unui fișier se folosește funcția cu prototipul:

FILE *fopen(char *nume_fisier, char *mod_acces);

Prin deschiderea unui fișier se stabilește o conexiune logică între fișier și variabila pointer și se alocă o zonă de memorie (buffer) pentru realizarea mai eficientă a operațiilor de intrare / ieșire. Funcția întoarce:

- un pointer la fișier, în caz că deschiderea fișierului se face în mod corect
- **NULL** dacă fișierul nu poate fi deschis.

Vom considera mai întâi două *moduri de acces* :

- *citire* (sau consultare) "r" – citirea dintr-un fișier inexistent va genera eroare
- *scriere* (sau creare) "w" - dacă fișierul există deja, el va fi șters

Fișierele standard nu trebuie deschise.

Redirectarea unui fișier deschis poate fi realizată cu:

FILE* freopen(char* nume, char* mod, FILE* flux));

Fișierul deschis este închis și este deschis un nou fișier având ca sursă fluxul, numele și modul de acces specificați ca parametri. O utilizare importantă o constituie redirectarea fluxului standard de intrare: datele vor fi citite din fișierul specificat, fără a face nici o modificare în program.

2. Închiderea unui fișier

După terminarea prelucrărilor asupra unui fișier, acesta trebuie închis. Un fișier este închis automat la apelarea funcției `exit()`.

```
int fclose(FILE *pf);
```

- funcția întoarce 0 la închidere normală și **EOF** la producerea unui incident
- fișierele standard nu se închid de către programator
- în cazul unui fișier de ieșire, se scriu datele rămase nescrise din buffer în fișier, așa că operația de închidere este obligatorie
- în cazul unui fișier de intrare, datele necitite din bufferul de intrare sunt abandonate
- se eliberează bufferele alocate
- se întrerupe conexiunea pointer – fișier

Secvența următoare deschide un fișier cu un nume dat și apoi îl închide.

```
FILE *pf = fopen("test1.dat", "w");
fclose(pf);
```

14.4. Operații de intrare – ieșire.

Tabel 14.1. Funcții pentru operații de intrare / ieșire

Tip fișier	Conversie	Unitate transferată	Funcții folosite
text	fără	caracter	<code>fgetc()</code> , <code>fputc()</code>
		linie	<code>fgets()</code> , <code>fputs()</code>
	cu	linii	<code>fscanf()</code> , <code>fprintf()</code>
binar	fără	articol (structură)	<code>fread()</code> , <code>fwrite()</code>

3.1. operații de intrare / ieșire la nivel de caracter

Scrierea unui caracter într-un fișier se face folosind funcția:

```
int fputc(int c, FILE *pf);
```

Funcția întoarce primul parametru sau EOF, în caz de eroare.

Citirea unui caracter dintr-un fișier se face cu funcția:

```
int fgetc(FILE *pf);
```

Funcția întoarce ca rezultat următorul caracter citit din fișier, convertit în întreg fără semn sau **EOF** dacă s-a citit sfârșit de fișier sau s-a produs o eroare la citire.

Exemplu : Scrieți un program care realizează copierea unui fișier. Numele celor două fișiere (sursă și destinație) sunt citite de la terminal.

```
#include <stdio.h>
```

```

/* copierea unui fisier */
void copiere1(FILE *, FILE *);
int main(){
    char numes[12], numed[12];
    gets(numes);
    gets(numed);
    FILE* s = fopen(numes, "r");
    FILE* d = fopen(numed, "w");
    copiere1(d, s);
    fclose(s);
    fclose(d);
}
void copiere1(FILE *d, FILE *s){
    int c;
    while ((c=fgetc(s)) != EOF)
        fputc(c, d);
}

```

3.2. operații de intrare / ieșire pentru șiruri de caractere

`char *fgets(char *s, int n, FILE *pf);`

- citește caractere din fișierul **pf**, până la întâlnirea primului caracter '\n' (cel mult **n-1** caractere) în tabloul **s**; pune la sfârșit '\n' și '\0'
- întoarce **s** sau **NULL** la întâlnirea sfârșit de fișier sau la eroare

Exemplu : Scrieți o funcție care simulează funcția fgets().

```

char *fgets(char *s, int n, FILE *pf) {
    char c;
    char *psir = s;
    while (--n > 0 && (c=fgetc(pf)) !=EOF)
        if((*psir++=c)=='\n')
            break;
    *psir='\0';
    return (c==EOF && psir==s) ? NULL: s;
}

```

`int fputs(char *s, FILE *pf);`

- copiază șirul în fișierul de ieșire
- nu copiază terminatorul de șir '\0'
- întoarce un rezultat nenegativ (numărul de caractere scrise în fișier), sau **EOF** la producerea unei erori

Exemplu : Scrieți o funcție care simulează funcția fputs().

```

int fputs(char *s, FILE *pf)
{ int c, n=0;
    while (c = *s++) {
        fputc(c, pf);

```

```

    n++;
}
return (ferror(pf)) ? EOF: n;
}

```

Copierea unui fișier folosind funcții orientate pe șiruri de caractere are forma:

```

#define MAX 100
void copiere2(FILE *d, FILE *s) {
    char linie[MAX];
    while(fgets(linie, MAX, s))
        fputs(linie, d);
}

```

Revenim acum asupra modurilor de acces la disc. Sunt posibile următoarele situații:

1. fișierul nu există; dorim să-l creem și să punem informații în el
 - “**w**” - deschidere pentru scriere, noile scrieri se fac peste cele vechi
2. fișierul există deja; dorim să extragem informații din el
 - “**r**” - deschidere pentru citire, fișierul trebuie să existe deja
 - “**r+**” - citire și scriere ; fișierul trebuie să existe
3. fișierul există deja; dorim să adăugăm informații la el, păstrând informațiile deja existente
 - “**a**” - deschidere pentru adăugare, toate scrierile se adaugă la sfârșitul fișierului existent sau nou creat
 - “**a+**” - citire și adăugare; dacă fișierul nu există, el va fi creat
4. fișierul există deja; dorim să punem alte informații în el ștergând pe cele existente
 - “**w+**” - citire și scriere; dacă fișierul există deja el este șters

Modul de acces binar se specifică cu sufixul “**b**”. Astfel avem: “**rb**”, “**w+b**”

Modul text este considerat implicit, dar poate fi specificat explicit prin “**t**”.

3.3. operații de intrare / ieșire cu format

- scrierea cu format

```
int fprintf(FILE *pf, char *format, lista_expresii);
```

- transferă în fișierul specificat, valorile expresiilor, convertite, potrivit formatului în caractere
- întoarce numărul de caractere scrise, sau o valoare negativă, dacă s-a produs o eroare.

Un descriptor de conversie din format începe prin % și poate avea un mai mulți specificatori opționali, care preced descriptorul:

```
%[indicator][lățime][.precizie][spec_lung]descriptor;
```

Indicatorul poate avea una din valorile:

- aliniere stânga
- + afișare numere cu semn
- 0 completare stânga cu zerouri
adăugare spațiu înaintea primei cifre, dacă numărul este pozitiv
- # %#o - scrie 0 inițial
- %#x - scrie 0x
- %#e, f, g, E, G – scrie punctul zecimal și nu elimină zerourile la sfârșit

Lățimea – număr ce indică lățimea minimă a câmpului în care se face scrierea.

* lățimea este dată de argumentul următor

Precizia este un număr interpretat diferit în funcție de descriptorul folosit. Astfel:

%e, %E, %f – numărul de cifre după punctul zecimal

%s – numărul maxim de caractere afișate

%g, %G – numărul de cifre semnificative

%d, %i – numărul minim de cifre (cu zerouri în față)

Specificarea lungimii se face prin:

H – short l – long L – long double

- *citirea cu format*

```
int fscanf(FILE *pf, char *format, lista_adrese_variabile);
```

- se citesc date din fișierul pf, sub controlul formatului, inițializându-se variabilele din listă
- funcția întoarce numărul de câmpuri citite sau **EOF** în caz de producere a unui incident la citire sau întâlnire a marcajului de sfârșit de fișier.

3.4. intrări / ieșiri în modul de acces binar

- sunt operații de transfer (citiri / scrieri) fără conversii
- se fac la nivel de articol
- poziția în fișier este actualizată după fiecare citire / scriere

```
unsigned fread(void *zona, unsigned la, unsigned na, FILE *pf);
```

- citește cel mult na articole, de lungime la fiecare, din fișierul pf în zona
- întoarce numărul de înregistrări citite sau 0 în caz de eroare sau sfârșit de fișier

```
unsigned fwrite(void *zona, unsigned la, unsigned na, FILE *pf);
```

- scrie na articole de lungime la, din zona în fișierul **pf**
- întoarce numărul de articole scrise.

Pentru a copia un fișier binar (sau text) folosind funcțiile **fread()** și **fwrite()** vom considera lungimea articolului 1 octet.

```
void copiere3(FILE * d, FILE * s) {
    int noc; /* numarul de octeti cititi */
    char zona[MAX];
    while((noc=fread(zona, 1, MAX, s)) > 0)
        fwrite(zona, 1, noc, d);
}
```

4. Operații pentru fișiere cu acces direct.

4.1. Poziționarea în fișier

```
int fseek(FILE *pf, long depl, int orig);
```

- modifică poziția curentă în fișierul **pf** cu **depl** octeți relativ la cel de-al treilea parametru **orig**, după cum urmează:
 - față de începutul fișierului, dacă **orig=0** (sau **SEEK_SET**)

- față de poziția curentă, dacă **orig=1** (sau **SEEK_CUR**)
- față de sfârșitul fișierului, dacă **orig=2** (sau **SEEK_END**)
- întoarce rezultatul 0 pentru o poziționare corectă, și diferit de 0 în caz de eroare.

```
void rewind(FILE *pf);
```

realizează o poziționare la începutul fișierului, fiind echivalent cu:

```
fseek(pf, 0L, SEEK_SET);
```

```
long ftell(FILE *pf);
```

- întoarce poziția curentă în fișier, exprimată prin numărul de octeți față de începutul fișierului

Exemplu : Scrieți o funcție care determină numărul de octeți ai unui fișier.

```
#include <stdio.h>
long FileSize(FILE *pf)
{ long pozv, noct;
  pozv = ftell(pf); /* salveaza pozitia curenta */
  fseek(pf, 0L, SEEK_END); /* pozitionare la sfarsit */
  noct = ftell(pf); /* numar de octeti din fisier */
  fseek(pf, pozv, SEEK_SET); /*revenirea la pozitia veche*/
  return noct;
}
```

4. tratarea erorilor

```
int feof(FILE *pf);
```

- întoarce o valoare diferită de 0, dacă s-a detectat marcajul de sfârșit de fișier

```
int ferror(FILE *pf);
```

- întoarce o valoare diferită de 0, dacă s-a detectat o eroare în cursul operației de intrare / ieșire

Exemplul : Fișierul "comenzi.dat" conține articole structuri cu câmpurile:

-denumire produs- un sir de 20 de caractere

-cantitate comandată – o valoare reală.

Fișierul "depozit.dat" este format din articole având câmpurile:

- denumire produs – un sir de 20 de caractere

- stoc și stoc_minim– valori reale

- preț unitar – valoare reală

Să se actualizeze fișierul de stocuri, prin onorarea comenziilor. O comandă este onorată, dacă prin satisfacerea ei, stocul rămas în magazie nu scade sub stocul minim.

Se va crea un fișier "facturi.dat", conținând pentru fiecare comandă onorată, denumirea produsului comandat și valoarea comenzi.

Se vor crea de asemenei două fișiere, unul cu comenzi care nu au fost satisfăcute, deoarece produsele erau în cantități insuficiente, celălalt cu comenzi de produse care nu există în depozit.

```
#include <stdio.h>
#include <stdlib.h>
```

```

typedef struct { char den[20];
                 double cant; } comanda;
typedef struct { char den[20];
                 double stoc, stoc_min, pret; } depozit;
typedef struct { char den[20];
                 double val; } factura;

int main(){
    comanda com;
    depozit dep;
    factura fac;
    double t;
    int gasit, eof;
    FILE *fc, *fd, *ff, *fc1, *fc2;
    /* deschidere fisiere */
    if((fc=fopen("comenzi.dat","rb"))==NULL){
        fprintf(stderr,"eroare deschidere comenzi\n");
        exit(1);
    };
    if((fd=fopen("depozit.dat","r+b"))==NULL){
        fprintf(stderr,"eroare deschidere depozit\n");
        exit(1);
    };
    ff=fopen("facturi.dat","wb");
    fc1=fopen("com1.dat","wb");
    fc2=fopen("com2.dat","wb");
    while(1) { /* ciclu procesare comenzi */
        if(!fread(&com, sizeof(com), 1, fc) break;
        gasit=0;
        rewind(fd);
        do {
            eof=fread(&dep, sizeof(dep), 1, fd)==0;
            if(!eof) {
                if(strcmp(com.den, dep.den)==0) {
                    gasit = 1;
                    if((t=dep.stoc - com.cant)>=dep.stoc_min) {
                        dep.stoc=t;
                        fseek(fd, -sizeof(dep), SEEK_CUR);
                        fwrite(&dep, sizeof(dep), 1, fd);
                        fac.val=com.cant * dep.pret;
                        strcpy(fac.den, com.den);
                        fwrite(&fac, sizeof(fac), 1, ff);
                    }
                else
                    fwrite(&com, sizeof(com), 1, fc1);
                break;
            }
        }
    }
} while(!gasit && !eof);
if(!gasit)

```

```

        fwrite(&com, sizeof(com), 1, fc2);
    }
    fclose(fc);
    fclose(fd);
    fclose(ff);
    fclose(fc1);
    fclose(fc2);
}

```

Tabel 14.2. Funcții utilizate în lucrul cu fișiere

Semnătură	Efect
FILE* fopen(char* nume, char* mod);	deschide fișierul; întoarce pointerul la fișierul deschis sau NULL dacă operația eșuează
int fclose(FILE* pf);	închide fișierul
int fgetc(FILE* pf);	citește 1 caracter din fișier și întoarce caracterul citit sau EOF
int fputc(char c, FILE* pf);	scrie caracterul în fișier
char* fgets(char* s, int n, FILE* pf);	citește din fișier în s cel mult n-1 caractere, sau până la întâlnirea ' \n ', în locul căruia pune ' \0 '. Întoarce s sau NULL , dacă s-a citit EOF .
int fputs(char* s, FILE* pf);	copiază sirul în fișierul de ieșire. Înlocuiește terminatorul ' \0 ' cu ' \n '.
Int fscanf(FILE* pf, char* fmt, lista_adrese);	citire din fișier sub controlul formatului. Întoarce numărul de câmpuri citite sau EOF , în caz de eroare sau sfârșit de fișier.
int fprintf(FILE* pf, char* fmat, lista_expresii);	scriere în fișier sub controlul formatului. Întoarce numărul de caractere scrise sau valoare negativă în caz de eroare.
int fread(char* zona, int la, int na, FILE* pf);	citește din fișier în zona , na articole de lungime la fiecare. Întoarce numărul de articole efectiv citite.
int fwrite(char* zona, int la, int na, FILE* pf);	scrie în fișier din zona , na articole de lungime la fiecare. Întoarce numărul de articole efectiv scrise.
int fseek(FILE* pf, long depl, int orig);	poziționare cu depl octeți față de început, poziția curentă sau sfârșitul fișierului
void rewind(FILE* pf);	poziționare la începutul fișierului
long ftell(FILE* pf);	determină poziția curentă în fișier.
int feof(FILE* pf);	întoarce nenul dacă s-a detectat sfârșit de fișier

int ferror(FILE* pf);	întoarce nenul dacă s-a detectat o eroare în cursul operației de intrare / ieșire
------------------------------	---

14.4. Probleme propuse.

1. Să se scrie un program pentru crearea unui fișier binar, având articole structuri cu următoarele câmpuri:

- **Nume** depunător - sir de maxim 30 de caractere
- **Data** depunerii – o structură având câmpurile întregi: **zi**, **lună**, **an**.
- **Sumă** depusă – o valoare reală.

Articolele sunt grupate pe zile în ordine cronologică. Datele se introduc de la consolă, fiecare pe trei linii.

2. Să se scrie un program care folosind fișierul creat în problema 1 calculează și afișează:

- Suma maximă depusă, împreună cu data și numele depunătorului.
- Numărul depunerilor din fiecare zi, și suma totală depusă în fiecare zi, ținând cont că tranzacțiile dintr-o zi sunt contigüe în fișier.

3. Să se scrie un program pentru actualizarea fișierului creat în problema 1, pe baza unor foi de restituire, introduse de la tastatură, conținând numele și suma solicitată. Programul semnalează la consolă următoarele situații:

- Solicitant inexistent
- Suma solicitată depășește soldul.

Fișierul actualizat este listat la consolă.

4. Să se scrie un program, care folosind fișierul creat în problema 1 actualizează acest fișier prin adăugarea dobânzii la data curentă.

Se precizează următoarele date:

- Data curentă la care se calculează dobânda (an, lună, zi)
- Dobânda anuală

Se va folosi o funcție care determină numărul de zile între data depunerii și data curent, pentru a calcula dobânda cuvenită.

1. Pentru a sorta elementele unui tablou **T**, fără a le deplasa, se creează un nou tablou **P**, în care un element **P_i** reprezintă poziția pe care ar avea-o elementul corespunzător din **T** în tabloul sortat, adică numărul de elemente care ar trebui să se găsească înaintea fiecărui element din tabloul sortat.:

$$P_i : n \leftarrow \#T_j \leq T_i$$

$0 \leq j \leq n-1$
 $j \neq i$

De exemplu:

I	0	1	2	3	4
T	8	2	5	9	6
P	3	0	1	4	2
X	1	2	4	0	3

Pe baza tabloului **P** se obține relativ simplu poziția (indexul) elementelor sortate din tabloul **T**. Cel mai mic element se află în **T** în poziția **k**, astfel încât **P_k=0**, următorul – în poziția corespunzătoare lui **P_{k+1}=1**, ultimul element corespunde poziției **k** pentru care **P_{k=n-1}**.

Dacă tabloul **T** nu are toate elementele distincte, pentru crearea tabloului **P** se face modificarea:

$$P_i = \text{numarul } T_j \leq T_i + \text{numarul } T_j < T_i$$

De exemplu:

I	0	1	2	3	4	5	6	7	8
T	8	2	5	8	5	9	2	6	5
P	6	0	2	7	3	8	1	5	4
X	1	6	2	4	8	7	0	3	5

Problema prezintă interes în cazul în care în locul tabloului **T** avem un fișier cu tip. Sortarea fișierului în raport cu o cheie (unul din câmpurile articolelor fișierului) revine la creearea unui fișier index care reprezintă un fișier de întregi (echivalent tabloului x), în care fiecare element x_i dă poziția celui de-al i -lea element din fișierul sortat în fișierul inițial.

- Să se definească o funcție, care primind ca parametru un fișier binar, creează un fișier index în raport cu o cheie.
- Să se definească o funcție care primind ca parametri un fișier și un fișier index asociat, afișează articolele fișierului sortate în raport cu indexul dat.

6. Se dă un fișier text.

- Să se determine numărul de linii din fișier.
- Să se creeze un nou fișier cu liniile din primul, apărând în ordine inversă.
- Pe baza fișierului inițial, să se creeze un fișier de caractere, în care nu mai apar caracterele de sfârșit de linie, iar fiecare linie este precedată de lungimea ei (un octet)
- Se dă un fișier de întregi reprezentând numere de linii din fișierul inițial. Să se afișeze la imprimantă liniile din fișier în ordinea precizată de fișierul de întregi.

7. Fișierul text **prog.c** reprezintă un program sursă C. Să se copieze acest fișier la ieșirea standard suprimând toate comentariile.

8. Se consideră fișierul **abonăți.dat** cu articole structuri având câmpurile:

- **Nume** – un sir de 20 de caractere

- **Adresă** – un sir de 30 de caractere
- **Data_expirării** – o structură cu câmpurile **an**, **lună**, **zi**.

Considerăm că data curentă se introduce de la tastatură.

Să se actualizeze fișierul de abonați, ștergând pe aceia al căror abonament a expirat la data curentă. Actualizarea se face creind un nou fișier în care se trec numai abonații al căror abonament nu a expirat și care la sfârșit va primi numele fișierului inițial.

Se va defini și folosi o funcție care compară două date (**d1** și **d2**) și întoarce **1**, dacă **d1** este înaintea lui **d2**, și **0** în caz contrar.

Un medicament este specificat printr-o structură care conține: denumirea comercială internațională (**dci**) – un sir de 20 caractere și **cantitate** – o valoare reală.

O rețetă compensată conține: **nume** pacient – un sir de 20 de caractere, **n** - numărul de medicamente prescrise și **n** structuri de tip medicament.

Farmacistul înlocuiește **dci** cu echivalentul medicamentului produs în țară, având cel mai mic preț pe care îl are în stoc și eliberează un bon care conține: numele pacientului, numărul de medicamente eliberate, și pentru fiecare medicament: numele echivalent autohton, valoarea medicamentului și la final, valoarea totală de plată.

Scrieți un program care automatizează aceste operații. Se precizează următoarele:

- rețetele sunt citite dintr-un fișier binar, cu numele dat ca parametru al comenzi
- echivalența dci – medicament autohton este dată într-un fișier binar, ce conține:

- **dci** – sir de 20 caractere
- **ne** – numărul de medicamente echivalente

și pentru fiecare medicament echivalent: numele medicamentului – 20 caractere și preț – valoare reală.

Dacă **dci** nu se găsește în fișierul de echivalențe, rețeta nu poate fi onorată, fapt consemnat printr-un mesaj la dispozitivul standard de eroare.

Dacă se găsesc echivalente pentru toate medicamentele din rețetă se crează un bon scris în fișierul de bonuri.

Numele celor 3 fișiere binare: fișierul de rețete, fișierul de echivalențe și fișierul de bonuri se dau ca parametri ai comenzi.

O comandă pentru un produs conține nume **produs** (20 caractere) și **cantitate** (real).

Un client este identificat prin: **nume** client (30 caractere) și număr produse comandate **npc** (întreg) și **npc** comenzi. Comenzile clienților sunt date într-un fișier binar.

Pentru satisfacerea acestor comenzi se consultă un fișier catalog ce conține articole de forma: nume **furnizor** (30 caractere), nume **produs** (20 caractere), **preț** unitar (real).

Scrieți un program care caută să satisfacă comenzile clienților folosind produsele din catalog cu prețurile cele mai mici.

Programul va crea un fișier de facturi, în care pentru fiecare comandă apare un articol de forma: nume client, număr produse livrate și suma totală de plată

Numele fișierelor se preiau ca parametri ai comenzi.

Un **produs** este precizat printr-o structură care conține: **nume** produs (20 caractere) și **cantitate** (real)

O firmă efectuează două tipuri de tranzacții: aprovisionări și vânzări. O tranzacție este specificată prin:

- **tip** tranzacție (un caracter ‘A’ sau ‘V’)
- **nume** furnizor sau client (30 caractere)
- **npc** - număr produse comandate (întreg)

și **npc** structuri **produs**.

In cazul unei aprovisionări mai există un câmp - **preț** unitar, care este dictat de furnizor

Pentru fiecare tranzacție, fiecare produs este căutat după nume într-un fișier de stocuri. Fișierul de stocuri conține articole cu lungime fixată, cu structura: nume produs (20 caractere), stoc (real), stoc minim (real) și preț unitar (real)

Pentru o aprovisionare, dacă produsul este găsit în fișierul de stocuri, cantitatea este adăugată la cea existentă în stoc, iar prețul întregului stoc se modifică la prețul unitar al produsului din această tranzacție.

Dacă produsul nu este găsit, el este adăugat la sfârșitul fișierului de stocuri, cu stoc minim zero.

Pentru o vânzare, dacă produsul este găsit în fișierul de stocuri, și este în cantitate suficientă pentru a satisface comanda fără a scădea sub stocul minim, comanda pentru acel produs este satisfăcută, și stocul este actualizat.

O comandă de vânzare poate fi satisfăcută total, pentru toate produsele solicitate, sau parțial, numai pentru produsele existente în stoc în cantități suficiente.

Folosind un fișier binar de comenzi se cere:

1)să se calculeze câștigul sau datoria rezultată din desfășurarea acestor tranzacții.

O aprovisionare va apărea cu o valoare negativă (o datorie), iar o vânzare cu o valoare pozitivă (un câștig).

2)să se actualizeze fișierul de stocuri conform tranzacțiilor

3)să se creeze un fișier de facturi de încasat pentru tranzacțiile vânzări. Într-o factură apar: nume client (30 caractere), număr produse livrate (întreg), numele produselor livrate(câte 20 caractere) și valoare de încasat (real)

Un client are o singură comandă (pentru mai multe produse). Numele fișierelor sunt preluate din linia de comandă

Implementați comanda **medie**, care calculează media aritmetică a unor valori reale, introduse din diferite surse. Comanda poate avea una din formele:

medie

medie nf

medie n1 n2 ... np

În primul caz, datele sunt citite de la tastatură, dintr-o singură linie, fiind separate prin spații albe, iar rezultatul este afișat pe ecran.

În a doua situație, datele sunt preluate dintr-un fișier text, cu numele dat ca argument al comenzi.

Media numerelor din fiecare linie este scrisă într-un fișier binar, cu același nume cu fișierul de intrare, dar cu extensia **.bin**.

În ultimul caz, datele sunt preluate ca parametri ai comenzi, iar rezultatul este afișat pe ecran.

Menționăm că, în acest ultim caz există cel puțin două argumente.

Se recomandă definirea și folosirea unei funcții **double med(char *s)**; care separă numerele reale din sirul de caractere s, le convertește și calculează media lor aritmetică. Pentru conversie se poate folosi funcția: **double atof(char *s)**; din stdlib.h care transformă sirul într-un real.

Utilizând un card de credit putem realiza atât operații de restituire cât și de depunere. Definiți tipul **card** ca o structură conținând următoarele câmpuri::

pin –un întreg reprezentând parola de acces a posesorului cardului

operatie –un caracter (R = restituire / D = depunere)

suma –un întreg lung reprezentând suma solicitată sau depusă

Scrieți un program care simulează funcționarea unui bancomat, preluând tranzacțiile (restituirile sau depunerile) dintr-un fișier binar de articole de tip card și actualizează o bază de date conturi ale clientilor. Acesta este tot un fișier binar, cu articole conținând: codul pin, contul clientului, datoria maximă pe care o poate face clientul.

Pentru păstrarea confidențialității numele clientilor sunt păstrate într-un fișier binar, care stabilește corespondența cod pin – nume client..

Programul crează și un bon care notifică tranzacția. Acesta va conține numele clientului, tipul operației, suma primită (sau depusă) și contul actualizat, și reprezintă un fișier binar cu articole având câmpurile menționate.

În cazul unei operații de restituire, clientul nu poate retrage o sumă care să facă datoria sa făță de bancă, mai mare decât cea specificată. Astfel dacă clientul are în cont 10.000 RON și poate avea o datorie de 15.000 RON, atunci el nu poate retrage mai mult de 25.000 RON.

Numele celor 4 fișiere (tranzacții, conturi, corespondențe și bonuri) sunt preluate ca parametri ai comenzi.

Dacă operația nu poate fi efectuată se afișează un mesaj de eroare, în care apare numele clientului.

Dintr-un fișier text se citesc mai multe matrice pătrate. O matrice este reprezentată pe o linie din fișier prin dimensiunea **n** și cele **n²** elemente reale, considerate în ordinea parcurgerii pe linii a matricei, separate între ele prin spații libere.

Creați un fișier binar, cu același nume cu fișierul de intrare, dar cu extensia **.ort**, conținând numai matricele ortogonale.(O matrice ortogonală satisface proprietatea **A · A^T=I_n**, unde **I_n** este matricea unitate de dimensiune **n**).

Numele fișierului text de intrare este dat ca parametru al comenzi.

La o bancă, pentru efectuarea unei operații (depunere sau restituire) clientul completează un formular în care trece: numele, tipul operației și suma. Pentru păstrarea secretului operațiilor banca folosește în locul numerelor clientilor, coduri. Corespondențele: nume client – cod și cod – cont sunt păstrate în fișiere binare. Tranzacțiile solicitate de clienți se introduc prin intermediul unui fișier text, în care o linie conține: numele clientului (scris cu nume și prenume separate prin **_**), tipul operației se specifică printr-un caracter **R** pentru restituire sau **D** pentru depunere și suma – o valoare reală. Aceste 3 elemente sunt separate în linie prin spații.

In fișierele binare câmpul nume are 20 de caractere, câmpul cod este un întreg lung, iar câmpul cont este un real.

Scrieți un program care prelucrează tranzacțiile și actualizează conturile clientilor. Pentru o depunere a unui client care nu are cont, i se asociază un cont, se adaugă perechea nume – cod în primul fișier de corespondențe și perechea cod – sumă în cel de-al doilea fișier binar. O restituire nu este făcută și se dă un mesaj de eroare în următoarele situații: 1.solicitantul nu există în primul fișier de corespondențe sau 2.suma solicitată depășește contul.

Numele celor două fișiere de corespondențe se dau ca parametri ai comenzi. Codul disponibil asociat unui client nou este păstrat într-o variabilă întreagă statică, și este incrementat pentru fiecare client nou adăugat.

Implementați comanda **indent numef**, care citește un fișier XML cu numele **numef**, corect sintactic și-l afișează fără marcaje, cu indentare (textul cuprins între două marcaje va fi afișat pe o linie nouă, decalat la dreapta cu 2 spații libere).

Un fișier XML este un fișier text care conține diverse siruri de caractere încadrate de marcaje. Un marcaj ("tag") este un sir între parantezele ascuțite '<' și '>'. Marcajele se folosesc în perechi: un marcaj de început (ex: <a>) și un marcaj de sfârșit (ex:). Perechile de marcaje pot fi incluse unele în altele.

Exemplu: <a> Nu sunt permise construcții de forma:

<a>

De exemplu fișierul: <a>T1T2<c>T3<d>T4</d>T5</c>T6 va apărea afișat ca:

T1

T2

T3

T4

T5

T6

Indicație: Se copiază fișierul în memorie. Intr-un ciclu se repetă operațiile:

- caută un marcaj (care începe cu '<')
- dacă este început de marcaj (următorul caracter nu este '/') se decalază poziția de afișare cu 2 poziții la dreapta
- dacă este sfârșit de marcaj se decalază poziția de afișare cu 2 poziții la stânga
- se sare peste marcaj, inclusiv peste >
- se afișează textul până la următorul marcaj (textul cuprins între '<' și '>')

Se recomandă definirea unei funcții care afișează cu decalaj dat caracterul între două adrese și o funcție care determină lungimea unui fișier text.

Un fișier binar conține mai multe matrice, date prin numărul de linii **m**, coloane **n** și cele **m x n** elemente reale, considerate în ordinea liniilor.

Se cere să se creeze un nou fișier binar, în care toate matricele de aceeași dimensiune să fie înlocuite cu suma lor.

În acest scop, se creează în memorie un tablou de matrice sume: la citirea unei matrice din fișierul binar se caută în acest tablou pentru a vedea dacă mai există o matrice cu aceleași dimensiuni – în caz afirmativ se adună la acea matrice, matricea citită din fișier, altfel se creează o nouă intrare în tablou cu matricea citită.

Numele celor două fișiere sunt date ca parametri ai comenzi. Nu se admite citirea integrală a fișierului de intrare în memorie.

Dintron-un fisier text, să se extragă cuvintele cu lungimea cuprinsă între m și n, având cel puțin p aparitii, și să se plaseze într-un fișier binar.

Un articol din fișierul binar va contine următoarele informații asupra unui cuvânt extras:

- l = lungimea cuvântului (întreg) $m \leq l \leq n \leq 15$
- cuv= tabloul continând cuvântul extras (tablou de 15 caractere)
- ap = numărul de aparitii a cuvântului (întreg) $ap \geq p$

m , n , p și numele fișierului text sunt preluate ca parametri ai comenzi. (numele fișierului text poate avea sau nu extensie).

Fișierul binar creat va avea același nume ca și fișierul text, dar extensia **.idx**.

Indicație: Întrucât numărul de aparitii al unui cuvânt este cunoscut numai după citirea întregului fișier text, care, în mod categoric, **nu va putea fi pastrat în memorie**, vom crea un fișier ajutător care va contine articole de forma:

```
struct art{  
    int l;          // lungime cuvânt  (m<=l<=15)  
    char cuv[15];  //cuvântul extras  
};
```

Fiecare cuvânt din fișierul temporar este căutat în fișierul de ieșire. Dacă se află acolo, înseamnă că a fost prelucrat și se trece mai departe; în caz contrar, se numără aparițiile lui în fișierul temporar și se trece în fișierul de ieșire, împreună cu numărul de apariții. În final, se scot din fișierul de ieșire articolele cu mai puțin de p apariții.

Functii recursive.

O definiție recursivă folosește termenul de definit în corpul definiției.
O funcție poate fi recursivă, adică se poate apela pe ea însăși, în mod direct sau indirect, printr-un lanț de apeluri.

```

/* apel recursiv direct */
void f(int x) {
    . . .
    f(10);
    . . .
}

/* apel recursiv indirect */
void f1(int x) {
    . . .
    f2(10);
    . . .
}

void f2(int y) {
    . . .
    f1(20);
    . . .
}

```

În cazul recursivității indirekte, compilatorul trebuie să verifice corectitudinea apelurilor recursive din fiecare funcție. O declarare anticipată a prototipurilor funcțiilor oferă informații suficiente compilatorului pentru a face verificările cerute.

Exemple de definiții recursive.

Multe obiecte matematice au definiri recursive, care pot fi exploataate algoritmice. De exemplu:

1⁰. numărul natural

- $1 \in \mathbb{N}$
 - dacă $x \in \mathbb{N}$ atunci $\text{succ}(x) \in \mathbb{N}$

2⁰. factorialul **n!** =

- 1 dacă $n=0$
 - $n*(n-1)!$ dacă $n > 0$

```
long fact(int n)
```

```
{ return (n? n*fact(n-1) : 1);}
```

3°. cel mai mare divizor comun **cmmdc** (a, b) =

- a daca $b=0$
 - $\text{cmmdc}(b, a\%b)$ dacă $b > 0$

```
int cmmdc(int a, int b)
{ return (b? cmmdc(b, a % b), a); }
```

4⁰. arborele binar

- arborele vid este arbore binar
-  este arbore binar, cu **SS** și **SD** arbori binari

○ Initial conditions

- baza sau partea nerecursivă
 - partea recursivă

Codul recursiv:

- rezolvă explicit cazul de bază
- apelurile recursive conțin valori mai mici ale argumentelor

Funcțiile definite recursiv sunt simple și elegante. Corectitudinea lor poate fi ușor verificată.

Definirea unor funcții recursive conduce la ineficiență, motiv pentru care se evită, putând fi înlocuită întotdeauna prin iterare.

Definirea unor funcții recursive este justificată dacă se lucrează cu structuri de date definite tot recursiv.

Recursivitate liniară.

Este forma cea mai simplă de recursivitate și constă dintr-un singur apel recursiv.

De exemplu pentru calculul factorialului avem:

$f(0) = 1$ cazul de bază

$f(n) = n * f(n-1)$ cazul general

Se vede că dacă timpul necesar execuției cazului de bază este **a** și a cazului general este **b**, atunci timpul total de calcul este: $a + b * (n-1) = O(n)$, adică avem complexitate liniară.

```
typedef unsigned long UL;

UL facrec(int n){
    if(n==0 || n==1) return 1UL;
    return n * facrec(n-1);
}

UL faciter(int n){
    UL fac = 1UL;
    int k;
    for(k=2; k<=n; k++)
        fac *= k;
    return fac;
}

UL cmmdc_rec(UL a, UL b){
    if(b==0) return a;
    return cmmdc_rec(b, a%b);
}

UL cmmdc_it(UL a, UL b){
    UL rest;
    if(b==0) return a; // e necesar pentru ca testul e la sfarsit
    do{rest = a % b;
       a = b;
       b = rest;
    } while(rest);
    return a;
}
```

Recursivitate binară.

Recursivitatea binară presupune existența a două apeluri recursive. Exemplul tipic îl constituie sirul lui Fibonacci:

```
long fibo(int n){  
    if(n<=2) return 1;  
    return fibo(n-1) +fibo(n-2);  
}
```

Funcția este foarte neficientă, având complexitate exponențială, cu multe apeluri recursive repetitive.

```
// termenul n din sirul Fibonacci (recursiv si iterativ)  
typedef unsigned long UL;  
  
UL fib_rec(int n){  
    // program simplu dar foarte neficient (complexitate exponentiala)  
    if(n==0 || n==1) return 1;  
    return fib_rec(n-1)+fib_rec(n-2);  
}  
  
UL fib_iter(int n){  
    // program simplu si eficient (complexitate liniara)  
    UL a=1, b=1, c;  
    int k;  
    for(k=2; k<=n; k++){  
        c = a + b;  
        a = b;  
        b = c;  
    }  
    return b; //funcționeaza corect si pt. n=0 sau n=1  
}  
  
// observatii: la n=40 varianta recursiva dureaza foarte mult  
// la n=45 varianta iterativa da inca rezultat corect  
// pentru n>45 executia iterativa este rapida dar rezultatul depaseste  
UL
```

Se poate înlocui dublul apel recursiv printr-un singur apel recursiv. Pentru aceasta trebuie să se țină seama că dacă a , b și c sunt primii 3 termeni din sirul Fibonacci, atunci calculul termenului situat la distanța n în raport cu a , este situat la distanța $n-1$ față de termenul următor b . Necesitatea cunoașterii termenului următor impune prezența a doi termeni în lista de parametri.

Algoritmul rezultat are complexitate liniară.

```
long f(long a, long b, int n){  
    if(n<=1) return b;  
    return f(b, a+b, n-1);  
}  
  
long fiblin(int n){  
    f(0, 1, n);  
}
```

Inversarea unui sir de caractere presupune în mod natural prezența unui tablou care să păstreze sirul de caractere. O soluție recursivă, în care după apelul recursiv se scrie

caracterul citit, permite evitarea folosirii tablourilor. Elementele sirului de caractere vor fi păstrate ca variabile locale (caractere) în înregistrările de activare generate lanțul de apeluri recursive.

```
#include <stdlib.h>
#include <stdio.h>

// inversarea recursiva a unui sir de caractere, introdus de la
// tastatura si terminat prin Enter, fara a folosi un tablou

void invsir(){
    char c;
    scanf("%c", &c);
    if(c=='\n') return;
    // printf("%c", c); vezi comentariul din final
    invsir();
    printf("%c", c);
}

int main(){
    invsir();
    printf("\n");
    return 0;
}

// Ce rezultat se obtine daca se transforma comentariul
// din main() in instructiune
```

Un algoritm recursiv obține soluția problemei prin rezolvarea unor instanțe mai mici ale aceleleași probleme.

- se împarte problema în subprobleme de aceeași natură, dar de dimensiune mai scăzută
- se rezolvă subproblemele obținându-se soluțiile acestora
- se combină soluțiile subproblemelor obținându-se soluția problemei
- ultima etapă poate lipsi (în anumite cazuri), soluția problemei rezultând direct din soluțiile subproblemelor.
- procesul de divizare continuă și pentru subprobleme, până când se ajunge la o dimensiune suficient de redusă a problemei care să permită rezolvarea printr-o metodă directă.
- metoda divizării se exprimă natural în mod recursiv: rezolvarea problemei constă în rezolvarea unor instanțe ale problemei de dimensiuni mai reduse

Exemple de probleme rezolvate recursiv.

1. Ridicarea unui număr la o putere întreagă.

$$x^n = \begin{cases} x^{n/2} \cdot x^{n/2} & \text{dacă } x \text{ este par} \\ x \cdot x^{n/2} \cdot x^{n/2} & \text{dacă } x \text{ este impar} \end{cases}$$

```

double putere(double x, int n) {
    if (n==0) return 1;
    double y = putere(x, n/2);
    if (n%2==0)
        return y*y;
    else
        return x*y*y;
}

```

Se constată că în urma separării se rezolvă o singură subproblemă de dimensiune $n/2$. Așadar: $T(n) = T(n/2) + 1$, care are soluția $T(n) = O(\log_2 n)$.

2. Căutarea binară – presupune căutarea unei valori într-un tablou sortat. În acest scop

- se compară valoarea căutată y cu elementul din mijlocul tabloului $x[m]$
- dacă sunt egale, elementul y a fost găsit în poziția m
- în caz contrar se continuă căutarea într-o din jumătățile tabloului (în prima jumătate, dacă $y < x[m]$ sau în a doua jumătate dacă $y > x[m]$).
- Funcția întoarce poziția m a valorii y în x sau -1 , dacă y nu se află în x . Complexitatea este $O(\log_2 n)$

```

int CB(int i, int j, double y, double x[]) {
    if(i > j) return -1;
    double m = (i+j)/2;
    if(y == x[m]) return m;
    if(y < x[m])
        return CB(i, m-1, y, x);
    else
        return CB(m+1, j, y, x);
}

int CautBin(int n, double x[], double y){
    if(n==0) return -1;
    return CB(0, n-1, y, x);
}

```

3. Localizarea unei rădăcini prin bisecție

Localizarea unei rădăcini a ecuației $f(x)=0$, separată într-un interval $[a, b]$ prin bisecție este un caz particular de căutare binară. Deoarece se lucrează cu valori reale, comparația de egalitate se evită.

```

double Bis(double a, double b, double (*f)(double)) {
    double c = (a+b)/2;
    if(fabs(f(c)) < EPS || b-a < EPS) return c;
    if(f(a)*f(c) < 0)
        return Bis(a, c, f);
    else
        return Bis(c, b, f);
}

```

4. Elementul maxim dintr-un vector

Aplicarea metodei divizării se bazează pe observația că maximul este cea mai mare valoare dintre maximele din cele două jumătăți ale tabloului.

```
double max(int i, int j, double *x){  
    double m1, m2;  
    if(i==j) return x[i];  
    if(i==j-1) return (x[i]>x[j] ? x[i]:x[j]);  
    int k = (i+j)/2;  
    m1 = max(i, k, x);  
    m2 = max(k+1, j, x);  
    return (m1>m2) ? m1: m2;  
}
```

5. Turnurile din Hanoi

Se dau 3 tije:stânga, centru, dreapta; pe cea din stânga sunt plasate **n** discuri cu diametre descrescătoare, formând un turn. Generați mutările care deplasează turnul de pe tija din stânga pe cea din dreapta. Se impun următoarele restricții:

- la un moment dat se poate muta un singur disc
- nu se permite plasarea unui disc de diametru mai mare peste unul cu diametrul mai mic
- una din tije servește ca zonă de manevră.

Problema mutării celor **n** discuri din zona sursă în zona destinație poate fi redusă la două subprobleme:

- mutarea a **n-1** discuri din zona sursă în zona de manevră, folosind zona destinație ca zonă de manevră
- mutarea a **n-1** discuri din zona de manevră în zona destinație, folosind zona sursă ca zonă de manevră
- Între cele două subprobleme se mută discul rămas în zona sursă în zona destinație

Avem satisfăcută ecuația recurrentă: $T(n) = 2 \cdot T(n-1) + 1$, care se rescrie:

$$\begin{aligned} T(n) &= 2^2 T(n-2) + 1 + 1 = 2^2 T(n-2) + 2 + 1 = 2^3 T(n-3) + 2^2 + 2 + 1 = \\ &= 2^{n-1} T(1) + 2^{n-2} + \dots + 2 + 1 \\ T(n) &= 2^{n-1} + \dots + 2 + 1 = 2^n - 1 = O(2^n) \end{aligned}$$

```
#include <stdio.h>  
#include <stdlib.h>  
// Turnurile din Hanoi - mutările pentru deplasarea a n discuri  
// de pe tija din stanga pe cea din dreapta, folosind tija din  
// mijloc ca zona de manevră  
void hanoi(int n, int sursa, int man, int dest);  
int main(){  
    int n;  
    printf("n="); scanf("%d", &n);  
    hanoi(n, 1, 2, 3);  
    return 0;  
}
```

```
void hanoi(int n, int sursa, int man, int dest){  
    if(n) {  
        hanoi(n-1, sursa, dest, man);  
        printf("%2d - %2d\n", sursa, dest);  
        hanoi(n-1, man, sursa, dest);  
    }  
}
```

Funcții polimorfice.

O entitate polimorfică poate avea mai multe tipuri (polimorfism = mai multe forme). O funcție polimorfică poate fi apelată cu parametri de tip diferit. La definirea unei funcții polimorfice se vor folosi parametri de tip **void ***.(pointeri generici). Aceștia vor fi înlocuiți, la apelul funcției, prin pointeri la tipuri definite.

În cazul pointerilor generici apar dificultăți legate de:

- accesul la informația referită printr-un pointer generic – pentru a accesa informația vom utiliza un pointer cu tip (de exemplu **char***)
- compararea informației referite prin pointeri generici – funcția polimorfică va conține un pointer la o funcție de comparație de informații referite prin **void***.

Aceasta va fi înlocuită printr-o funcție de comparație definită de utilizator.

Exemplificăm prin definirea unei funcții polimorfice de căutare binară.

Informația indicată prin pointerii **void *** o accesăm prin pointeri la **char***.

```
//definire tip pointer la functie de comparatie generala
typedef int (*PFCP) (void* ch, void* el);

//functie polimorfica de cautare binara
void* CautBin(void* ch, void* tb,
               int n, int dim, PFCP fcp) {
    // ch=pointer la cheia cautata
    // tb=pointer la tabelul in care se face cautarea
    // n=numarul de elemente din tabel
    //dim=numarul de octeti al unui element din tabel
    //fcp=pointer la functia generala de comparatie
    int rezcomp;
    //pointer la primul element din tabel
    char *min = (char*) tb;
    char *med;
    //pointer la ultimul element din tabel
    char *max = (char*) tb + (n-1)*dim;
    while(min <= max) {
        med = min + (max-min)/dim/2*dim;
        rezcomp = (*fcp) (ch, med);
        if(rezcomp < 0)
            max = med - dim;
        else
            if(rezcomp > 0)
                min = med + dim;
            else
                return med;
    };
    return NULL;
}
```

Pointerii la elementele de același tip pot fi comparați (`min <= max`). Pointerul `med` se obține pe o cale mai ocolită: se adună la pointerul `min` o valoare constantă – distanța între `min` și `max`, exprimată în octeți (`max-min`) este convertită mai întâi în indice, prin împărțirea cu dimensiunea elementului `dim`, este apoi împărțită la 2, pentru a obține mijlocul și transformată din nou în octeți.

Funcția de comparație are ca parametrii pointeri generici la valorile comparate și întoarce o valoare negativă, 0 sau o valoare pozitivă, în funcție de rezultatul comparației. Ea va fi adresată printr-un pointer (pointer la funcție).

Explicitarea funcției concrete de comparație se va face în apel. Vom considera două cazuri: compararea a două valori întregi, respectiv compararea a două siruri de caractere. Pointerii generici sunt convertiți în pointeri la tipul respectiv (întreg sau sir de caractere), se face dereferențierea și se returnează ca rezultat diferența valorilor, respectiv rezultatul comparației sirurilor de caractere.

```
int cpint(void* ch, void* el){
    return *(int*)ch - *(int*)el;
}

int cpsir(void* ch, void* el){
    return strcmp((char*) ch, (char*)el));
}
```