Despre carte

Această culegere de probleme este un instrument indispensabil în învățarea programării, permițând atât fixarea sintaxei și semanticii instrucțiunilor limbajului, cât și însușirea unor principii corecte de programare. Ea constituie un suport consistent pentru orice curs de programare a calculatoarelor.

Lucrarea, structurată pe 8 capitole, conține 120 de probleme rezolvate și alte 165 de probleme propuse.

Exemplele sunt alese cu grijă, fiind sortate în ordinea dificultății. Pentru fiecare problemă rezolvată se prezintă succint, în limbaj natural, metoda de rezolvare, după care este dat programul comentat. Problemele rezolvate au fost testate de către autori, care garantează astfel corectitudinea soluțiilor propuse. Problemele propuse sunt asemănătoare sau sunt dezvoltări ale problemelor rezolvate și contribuie la creșterea încrederii programatorului în propriile forțe.

Despre autori

Valeriu IORGA, doctor inginer (din 1982), este prof. univ. la Facultatea de Automatică și Calculatoare din Universitatea *Politehnica* București. Predă cursuri de:

- *Programarea Calculatoarelor*
- *Metode numerice*
- *Structuri de Date și Algoritmi*
- *Sisteme de programare pentru timp real*

A publicat diverse cărți de specialitate în prestigioase edituri din România, precum și cursuri universitare litografiate, articole în reviste de specialitate. Este membru al ACM și al Societății Române de Informatică.

Paul - Alexandru CHIRIȚĂ, student în anul V la Facultatea de Automatică și Calculatoare din Universitatea *Politehnica* București, a obținut numeroase premii la concursurile de cercetare studențești. În perioada 2000-2001 a fost bursier Socrates la Ecole Polytechnique, Palaiseau, Franța. De asemenea, a efectuat un stagiu de specializare în cadrul firmei *Schlumberger* din Paris. În prezent este profesor asistent pentru cursurile de *Programarea Calculatoarelor* și *Structuri de Date și Algoritmi* și își pregătește proiectul de diplomă în cadrul Learning Lab Lower Saxony, Hanovra, Germania.

Corina STRATAN, studentă în anul V la Facultatea de Automatică și Calculatoare din Universitatea *Politehnica* București, a participat la realizarea a numeroase proiecte de cercetare în cadrul laboratorului de e-Business din UPB și al Centrului Național de Tehnologia Informației **CoLaborator**. În prezent este profesor asistent pentru cursurile de *Programarea Calculatoarelor* și *Structuri de Date și Algoritmi* și își pregătește proiectul de diplomă sub supervizarea Universității Caltech din Statele Unite.

Cristian OPINCARU este student în anul V la Facultatea de Automatică și Calculatoare din Universitatea *Politehnica* București. În cadrul acestei facultăți ține ore de laborator la cursurile de *Programarea în limbajul C* (Anul I), *Structuri de date și algoritimi* (Anul I) și *Structura Sistemelor de Calcul* (Anul IV).

Cristian OPINCARU este coautor al unui manual de informatică de clasa a X-a.

Valeriu Iorga

Paul Chiriță **Corina Stratan** **Cristian Opincaru**

Programare în C/C++

Culegere de probleme

NICULESCU

Adresa: 781821 – Bucureşti, Sector 1
Str. Octav Cocărăscu 79, Tel/Fax: 222.03.72
Tel.: 224.24.80, 223.25.05
E-mail: edit@niculescu.ro
Internet: www.niculescu.ro

Procesare computerizată: *TOP GAL S.R.L.*

Tipărit la *S.C. EURO PONTIC*

ISBN 973-568-800-X

Prefaţă

Cu toţii acceptăm faptul că nu este întotdeauna uşor să te faci înţeles... mai ales dacă nu te exprimi în limbaj natural, ci printr-un program.

Programarea structurată a căutat, prin strategia de abordare top-down, prin impunerea unui număr redus de structuri de control cu o intrare şi o ieşire, prin evitarea instrucţiunii de salt necondiţionat şi prin comentarii inteligente, să facă programele mai clare, mai uşor de înţeles şi de controlat, pentru a opera modificări fără riscuri.

A fost definit astfel un *stil de programare* reprezentând mai mult un set de reguli, acceptate şi respectate de comunitatea programatorilor. Acestea stabilesc:

- utilizarea unei scrieri aliniate (indentate) în interiorul instrucţiunilor structurate (decizie, selecţie, ciclu, instrucţiune compusă), cu intervale de decalare bine stabilite (8 spaţii după unii, 4 după alţii mai economi)
- scrierea unei singure instrucţiuni pe o linie
- comentarea semnificativă a unor secvenţe mai sofisticate de instrucţiuni
- definirea de funcţii care realizează un singur obiectiv – bine precizat, formulat clar printr-un comentariu ce însoţeşte funcţia şi stabileşte semnificaţia fiecărui parametru.

Prezenta lucrare, structurată pe 8 capitole, conţine 120 de probleme rezolvate şi alte 165 de probleme propuse. Ea constituie un suport consistent pentru orice curs de programare a calculatoarelor. Multe dintre problemele propuse şi rezolvate au reprezentat aplicaţii de seminar şi laborator, subiecte date la lucrări de control şi la examene. Toate problemele rezolvate au fost testate de către autori, care garantează astfel corectitudinea soluţiilor propuse.

Pentru rezolvarea unei probleme concrete folosind calculatorul nu este suficient să cunoşti instrucţiunile unui limbaj de programare. Dromey, în lucrarea „How to solve it by computer" abordează această chestiune oferind un număr important de modele (probleme rezolvate semnificative).

O culegere de probleme este un instrument indispensabil în învăţarea programării folosind un limbaj de programare, permiţând atât fixarea sintaxei şi semanticii instrucţiunilor limbajului, cât şi însuşirea unor principii corecte de programare.

Exemplele sunt alese cu grijă, fiind sortate în ordinea dificultăţii. Pentru fiecare problemă rezolvată se prezintă succint, în limbaj natural, metoda de rezolvare după care este dat programul comentat.

În capitolul 1, ***Instrucțiuni***, sunt dezvoltați algoritmi simpli folosind numai tipurile de date primitive simple și instrucțiunile limbajului C. O serie de aplicații se bazează pe aplicarea unor relații de recurență pentru calculul unor limite de șiruri sau sume de serii.

În capitolul 2 se definesc și se utilizează funcții nerecursive și recursive.

Capitolul 3, ***Vectori***, introduce tipul tablou și pointerii, permițând dezvoltarea unor algoritmi de selecție, sortare, calcule cu polinoame, numere lungi și mulțimi.

În capitolul 4, ***Matrici***, se introduc tablourile cu mai multe dimensiuni, alocarea dinamică de memorie și se dezvoltă probleme de algebră liniară.

Capitolul 5, intitulat "Șiruri de caractere", prezintă aplicații cu caracter nenumeric.

În capitolul 6, ***Structuri***, este introdus tipul înregistrare (structură) printr-o serie de exemple semnificative.

În capitolul 7, denumit ***Fișiere***, se dezvoltă aplicații ce utilizează fișiere text și fișiere binare.

Capitolul 8, ***Clase***, utilizează facilitățile limbajului C++ relative la programarea orientată pe obiecte – clase, supraîncărcare operatori, derivare și polimorfism. În exemplele alese se definesc clase pentru lucrul cu numere complexe, fracții raționale, polinoame, numere lungi, șiruri de caractere, figuri geometrice, etc.

Problemele propuse sunt asemănătoare sau sunt dezvoltări ale problemelor rezolvate și au un rol important pentru student. Ele dezvoltă atitudinea activă și contribuie la creșterea încrederii programatorului în propriile forțe.

Ne exprimăm speranța ca această lucrare să fie de un real folos celor interesați.

Autorii

Capitolul 1
Instrucţiuni

Breviar

Atribuirea simplă:
```
variabila   = expresie;
```
Atribuirea compusă:
```
variabila op = expresie;
```
Atribuirea multiplă:
```
variabila_1  = variabila_2 = … = expresie;
```
Instrucțiunea compusă:
```
{ declaratii_si_definitii;
  instructiuni;
}
```
Decizia (forma generală):
```
if (expresie)
  instructiune1;
else
  instructiune2;
```
Decizia (forma simplificată):
```
if (expresie)
      instructiune;
```
Decizia multiplă:
```
if (expr1)
  instr1;
else if (expr2)
       instr2;
…
else
  instrn;
```
Selecția:
```
switch (expresie){
  case val1:  secventa1;
  case val2:  secventa2;
                . . .
  default:    secventa s;
}
```
Ciclul `while` (cât timp):
```
while (expresie)
  instructiune;
```
Ciclul `do`:
```
do
  instructiune;
while (expresie);
```
Ciclul cu contor:
```
for (exp_init; exp_test; exp_modif)
  instructiune;
```
Saltul la pasul următor al unui ciclu :
```
continue;
```
Ieșirea dintr-un ciclu :
```
break;
```
Întoarcerea unui rezultat dintr-o funcție:
```
return expresie;
```

Funcții matematice uzuale

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

Notație	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);	Funcții trigonometrice inverse
arcos(x)	double acos(double);	
arctg(x)	double atan(double);	
arctg(y/x)	double atan2(double, double);	
sinh(x)	double sinh(double);	Funcții hiperbolice
cosh(x)	double cosh(double);	
th(x)	double tanh(double);	
exp(x)	double exp(double);	Exponențială naturală
10^n	double pow10(int);	Exponențială zecimală
a^b	double pow(double, double);	Exponențială generală
ln(x)	double log(double);	Logaritm natural
lg(x)	double log10(double);	Logaritm zecimal
|x|	double fabs(double);	Valoare absolută
	int abs(int);	
	long labs(long);	
√x	double sqrt(double);	Rădăcină pătrată
	long double sqrtl(long double);	
⌈x⌉	double ceil(double);	Întregul minim >= x
⌊x⌋	double floor(double);	Întregul maxim <= x
conversii	double atof(const char *);	Conversie șir de caractere în float
	long double atold (const char*);	Conversie șir de caractere în long double

Probleme rezolvate

R1_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 sexagesimale. (*Probleme similare*: P1_11.)

Rezolvare: Numărul de grade se obține reținând partea întreagă a produsului $\frac{rad \cdot 180}{\pi}$. Numărul de minute este partea întreagă a fracțiunilor de grad înmulțite cu 60 și analog, secundele se obțin reținând partea întreagă din fracțiunile de minut, înmulțite cu 60.

Au fost folosite atribuiri multiple pentru a calcula numărul de grade (minute) și fracțiuni și pentru reținerea părții întregi a acestuia.

1_1.cpp

```
#include <stdio.h>
#include <conio.h>
#define PI 3.1416

void main (void) {
  // Unghiul in radiani
  double unghi_rad, gfr, minfr;
  // Unghiul in grade, minute si secunde sexagesimale
    int grade, minute, secunde;
  // Stergere ecran
  clrscr();
  // Citirea datelor de intrare
  printf (" Introduceti unghiul in radiani: ");
  scanf ("%lf", &unghi_rad);
  grade = gfr = unghi_rad*180./PI;
  minute = minfr = (gfr - grade) * 60.;
  secunde =(minfr - minute) * 60.;
  // Afisarea rezultatelor
  printf ("\n Unghiul are %d grade, %d minute si %d
secunde.\n",\
        grade, minute, secunde);
  getch();
}
```

R1_2. Un maratonist porneşte în cursă la un moment de timp exprimat prin ora, minutul şi secunda startului. Se cunoaşte de asemeni timpul necesar sportivului pentru parcurgerea traseului. Să se determine momentul terminării cursei de către sportiv. (*Probleme asemănătoare*: P1_8, P1_10, P1_12, P1_14, P1_22)

***Rezolvare*:** Se efectuează adunarea pe cele trei ranguri: secunde, minute şi ore. Se propagă transport între ranguri (secunde → minute, minute → ore) dacă suma în rangul respectiv depăşeşte 59.

Pentru a evita testul depăşirii se calculează transportul în rangul următor ca `suma/60`. Refacerea sumei se face folosind operatorul `%`.

Trebuie avut grijă ca după efectuarea sumelor pe ranguri, mai întâi să se propage transportul în rangul următor şi apoi să se refacă suma din acel rang.

1_2.cpp

```
#include <stdio.h>
#include <conio.h>

void main (void) {
  // Momentul plecarii
  int ora_start, min_start, sec_start;
  // Timpul parcurs
  int ore, minute, secunde;
  // Momentul sosirii
  int ora_sosire, min_sosire, sec_sosire;
  // Citire date de intrare
  clrscr();
  printf (" Introduceti momentul plecarii (HH MM SS): ");
  scanf("%d%d%d", &ora_start, &min_start, &sec_start);
  printf (" Introduceti durata cursei HH MM SS ");
  scanf("%d%d%d", &ore, &minute, &secunde);
  // Calculam momentul sosirii.
  sec_sosire = sec_start + secunde;
  min_sosire = min_start + minute;
  ora_sosire = ora_start + ore;
  // Propagam transporturile
  min_sosire += sec_sosire / 60;
  ora_sosire += min_sosire / 60;
  // Refacem sumele in ranguri
  sec_sosire %= 60;
  min_sosire %= 60;
  ora_sosire %= 24;
  // Afisam rezultatele
  printf (" Momentul sosirii este: %02d:%02d:%02d .\n", \
  ora_sosire, min_sosire, sec_sosire);
  getch();
}
```


R1_3. Se consideră sistemul de ecuaţii:

$$a\,x + b\,y = c$$
$$m\,x + n\,y = p$$

dat prin valorile coeficienţilor a, b, c, m, n, p. Să se rezolve sistemul cu discuţie. (*Probleme similare*: P1_13)

***Rezolvare*:** Avem două cazuri:

- dacă $m\,b - n\,a = 0$, atunci:
  - dacă $m\,c - a\,p = 0$ şi $b\,p - n\,c = 0$, atunci sistemul are o infinitate de soluţii (compatibil nedeterminat)
  - dacă una din aceste valori este 0 şi cealaltă diferită de 0 sistemul este incompatibil
- altfel, soluţiile sunt: $y = (m\,c - a\,p) / (m\,b - n\,a)$ şi $x = (b\,p - n\,c) / (m\,b - n\,a)$.

1_3.cpp

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

void main (void) {
  double a, b, c;
  double m, n, p;
  double x, y;                  // Solutiile sistemului
  const double eps = 0.001; // Pentru teste pe variabile double
  // Citire date de intrare
  clrscr();
  printf (" Introduceti a: ");  scanf ("%lf", &a);
  printf (" Introduceti b: ");  scanf ("%lf", &b);
  printf (" Introduceti c: ");  scanf ("%lf", &c);
  printf (" Introduceti m: ");  scanf ("%lf", &m);
  printf (" Introduceti n: ");  scanf ("%lf", &n);
  printf (" Introduceti p: ");  scanf ("%lf", &p);
  if (abs(m*b - n*a) < eps) {
    // Sistem incompatibil sau compatibil nedeterminat
    if (abs(m*c - a*p) > eps)
      printf (" Y nu poate fi calculat! \n");
    else
      printf (" Avem o infinitate de solutii pentru Y! \n");
    if (abs(b*p - n*c) > eps)
      printf (" X nu poate fi calculat! \n");
    else printf (" Avem o infinitate de solutii pentru X! \n");
  }
  else
  { // Sistem compatibil
    y = (m*c - a*p)/(m*b - n*a);
    x = (b*p - n*c)/(m*b - n*a);
    printf (" Solutiile sunt: x=%lf \t y=%lf \n", x, y);
  }
  getch();
}
```

R1_4. Să se scrie algoritmul pentru rezolvarea cu discuție a ecuației de gradul 2: $ax^2+bx+c=0$. Se dau pe mediul de intrare coeficienții *a, b, c* care pot avea orice valori reale reprezentabile în memoria calculatorului. (*Probleme similare:* P1_16.)

Rezolvare: Avem următoarele cazuri:

- $a=b=c=0$ - Ecuația are o infinitate de soluții
- $a=b=0,\ c\neq 0$ - Ecuația nu are nici o soluție.
- $a=0,\ b,\ c\neq 0$ - Ecuația este de gradul întâi şi are soluția *–c/b.*
- $a\neq 0$ - Definim $\Delta=b^2-4ac$

  Dacă $\Delta>0$ sau $\Delta=0$, atunci soluțiile sunt: $\frac{-b\pm\sqrt{\Delta}}{2a}$

  Dacă $\Delta<0$, atunci soluțiile $\frac{-b\pm i\sqrt{-\Delta}}{2a}$ sunt:

1_4.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

void main (void) {
  double a, b, c;
  double x1, x2, delta;
  const double eps = 0.001; // Pentru teste pe variabile
double
  clrscr();
  printf (" Introduceti a: ");
  scanf ("%lf", &a);
  printf (" Introduceti b: ");  scanf ("%lf", &b);
  printf (" Introduceti c: ");
  scanf ("%lf", &c);
  if (abs(a) < eps)      // Testam daca a == 0, de fapt
  { // Ecuatie de gradul intai
      if (abs(b) < eps)
        if(abs( c )  < eps)
            printf(" Ecuatia are o infinitate de solutii\n");
      else
        printf (" Ecuatia nu are solutii, a == b == 0.\n");
    else
    { printf (" Ecuatia este de gradul intai!\n");
      printf (" Radacina este: %lf\n", -c/b);
    }
  }
```



```
  else
  { // Ecuatie de gradul doi
    delta = b*b - 4*a*c;
    // Testam daca avem radacini complexe
    if (delta < -0.0) {
      printf ("x1=%lf+i*(%lf)\n", -b/(2*a), sqrt(-delta)/(2*a));
      printf ("x2=%lf-i*(%lf)\n", -b/(2*a), sqrt(-delta)/(2*a));
    }
    else
    { printf ("x1=%lf\n", (-b/(2*a)) + sqrt(delta)/(2*a));
      printf ("x2=%lf\n", (-b/(2*a)) - sqrt(delta)/(2*a));
    }
  }
  getch();
}
```

R1_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).

Rezolvare:

1_5.cpp

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

void main (void) {
  double x,y;
  clrscr();
  printf (" Introduceti x: ");
  scanf ("%lf", &x);
  printf (" Introduceti y: ");
  scanf ("%lf", &y);
    if (x >= 0) {
    if ( (x == 0) && (y == 0) ) {
      printf (" Punctul introdus este originea.");
      getch();
      // Iesim pentru a evita imbricarea de if-uri
      exit (0);
    }
```

```
    if (y >= 0) {
      // Daca se ajunge aici, se intra sigur in unul
      //    din if-urile de mai jos
      if (x == 0)
      printf (" Punctul se afla in cadranele 1-2.");
      // Nu pot fi ambele 0, pentru ca am testat mai sus
      if (y == 0)
      printf (" Punctul se afla in cadranele 1-4.");
      if ( (x > 0) && (y > 0) )
      printf (" Punctul se afla in cadranul 1.");
    }
    else
    { if (x == 0)
       printf (" Punctul se afla in cadranele 3-4.");
      else
        printf (" Punctul se afla in cadranul 4.");
    }
  }
  else /* x < 0 */
  { if (y == 0)
      printf (" Punctul se afla in cadranele 2-3.");
    if (y < 0)
      printf (" Punctul se afla in cadranul 3.");
    if (y > 0)
      printf (" Punctul se afla in cadranul 2.");
    }
  }
  getch();
}
```

R1_6. Dându-se n pe mediul de intrare, să se calculeze: $S = \sum_{i=1}^{n} (-1)^i i$
(*Probleme similare*: P1_1, P1_15)

Rezolvare:

1_6.cpp

```
#include <stdio.h>
#include <conio.h>

void main (void) {
  int n, suma = 0;
  printf (" Introduceti n: "); scanf ("%d", &n);
  for (int i = 1; i <= n; i++)
      if (i % 2)
        suma -= i;
      else
        suma += i;
  printf (" Suma este: %d\n", suma);
  getch();
}
```


R1_7. Abaterea medie pătratică a rezultatelor obținute prin determinări experimentale se poate calcula cu formula:

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

aplicabilă numai dacă s-au făcut cel puțin 2 măsurători.
Dându-se pe mediul de intrare N ($N \le 25$) și rezultatele celor N determinări să se calculeze abaterea medie pătratică.

Rezolvare:

1_7.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

void main (void) {
  // Initializam pe N cu o valoare in afara celor posibile
  int N = 26;
  double x;                 // o valoare determinata
experimental
  double s=0, sp=0;         // suma si suma patratelor
  double sigma;
  // Citire cu validare N
  clrscr();
  while (N > 25 || N < 0) {
      printf (" Introduceti N: ");
    scanf ("%d", &N);
  }
  for (int i = 0; i < N; i++)   {
      printf ("Introduceti determinarea %d: ", i + 1);
    scanf ("%lf", &x);
    s+=x;
    sp+=x*x;
  }
  if(N > 1) {
      sigma = sqrt((N*sp - s*s) / (N*(N-1)));
  printf (" Abaterea medie patratica este: %lf\n", sigma);
  getch();
  }
}
```

R1_8. Să se determine cel mai mare (max) precum şi cel mai mic (min) element dintr-un şir $a_0, a_1, \ldots a_{n-1}$.
Se dau pe mediul de intrare n precum şi cele n elemente ale şirului, care sunt citite pe rând într-o aceeaşi variabilă *a*. (*Probleme înrudite*: P1_17, P1_18)

Rezolvare: Ideea este ca pe măsură ce citim un număr *a* mai mare / mai mic decât maximul / minimul să actualizăm extremele.

1_8.cpp
```
#include <stdio.h>
#include <conio.h>
#include <values.h>

void main (void) {
  int n, a;
  int min = MAXINT,
      max = -MAXINT;
  // Citire date de intrare
  printf (" Introduceti n: "); scanf ("%d", &n);
  for (int i = 0; i < n; i++) {
    printf (" Elementul %d: ", i+1);
    scanf ("%d", &a);
    if (min > a)
      min = a;
    if (max < a)
      max = a;
  }
  printf (" Maximul este: %d\n", max);
  printf (" Minimul este: %d\n", min);
  getch();
}
```

R1_9. Se dau pe mediul de intrare notele obţinute de către studenţii unei grupe la un examen, precedate de numărul studenţilor. Să se determine dacă grupa este sau nu integralistă, precum şi procentajul de note foarte bune (8, 9, 10).

Rezolvare:

1_9.cpp
```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```



```
void main (void) {
  int nr_studenti, nota;
  int note_bune = 0;      // numarul de note foarte bune
  int integralista = 1;   // spune daca avem grupa integralista
  // citire date de intrare
  printf (" Introduceti numarul de studenti: ");
  scanf ("%d", &nr_studenti);
  for (int i = 0; i < nr_studenti; i++) {
    printf (" Nota studentului %d: ", i+1);
    scanf ("%d", &nota);
    if (nota < 5)
      integralista = 0;
    if (nota >= 8)
      note_bune += 1;
  }
  // afisare rezultate
  if (integralista)
    printf (" Grupa este integralista!\n");
  else
    printf (" Grupa NU este integralista!\n");
  printf (" Procentajul de note foarte bune este: %lf\n", \
        (double)note_bune/(double)nr_studenti);
  getch();
}
```

R1_10. Se consideră funcţia $f(x)=\ln(2x^2+1)$. Să se scrie un program pentru tabelarea pe intervalul [-10, 10] cu următorii paşi:
- 0.1 pentru $|x| \leq 1.0$
- 0.5 pentru $1.0 < |x| \leq 5.0$
- 1.0 pentru $5.0 < |x| \leq 10.0$.

Rezolvare:

1_10.cpp
```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

void main (void) {
  double x = -10, eps = 0.001;
  printf (" Valorile functiei ln(2*x^2 + 1) sunt:\n");
  while (x <= 10) {
    printf (" f(%lf) = %lf \n", x, log(2*x*x+1));
```

```
    // Pentru x == 1.0 trebuie sa incrementam cu 0.5
    if (fabs(x) <= 1.0 && fabs(x - 1.0) > eps)
      x += 0.1;
    else
      // Pentru x == 5.0 trebuie sa incrementam cu 1.0
      if (fabs(x) <= 5.0 && fabs(x - 5.0) > eps)
      x += 0.5;
      else x += 1.0;
    getch();
  }
}
```

R1_11. Să se calculeze şi să se afişeze valorile integralei: $I_k(x) = \int_0^x u^k e^u du$

pentru $k = 1, 2, ..., n$, în care n şi x sunt daţi, cunoscând că:

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

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

Rezolvare:

1_11.cpp

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main() {
  double s;  /* valoarea integralei */
  double x, t;
  double xk;  /* variabila in care retinem x^k */
  int n, p, k;
  printf("\n n=");
  scanf("%d", &n);
  printf("\n x=");
  scanf("%lf", &x);
  printf(" Cele %d integrale sunt: \n", n);
  xk = 1.0;
  for (k = 1; k <= n; k++)  /* calculam n integrale */
  { xk *= x;
    s = t = xk;  /* s - valoarea integralei */
    for (p = 1; p <= k; p++){   /* adunam k termeni */
      t *= -(k - p + 1) / x;    /* t - termenul curent */
      s += t;
    }
    s *= exp(x);
    printf(" Integrala I%d(%.2lf) este %lf \n", k, x, s);
  }
  getch();
}
```


R1_12. Şirurile $\{u_n\}$ şi $\{v_n\}$ generate cu relaţiile de recurenţă:

$u_n = (u_{n-1} + v_{n-1})/2$ şi $v_n = \sqrt{u_{n-1} v_{n-1}}$ pornind cu $u_0 = 1/|a|$, $v_0 = 1/|b|$, unde $a \neq 0$, $b \neq 0$ au o aceeaşi 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 şi b daţi, ca limită comună a celor două şiruri, determinată aproximativ cu precizia eps, în momentul în care distanţa între termenii celor două şiruri devine inferioară lui eps, adică $|u_n - v_n| < \text{eps}$. (*Probleme similare*: P1_2, P1_3, P1_4, P1_6, P1_7.)

Rezolvare: Termenii curenţi ai şirurilor se reţin în două variabile, u şi v. Având în vedere că la calculul noii valori a lui v ($v_n = \sqrt{u_{n-1} v_{n-1}}$) folosim valoarea veche a lui u, trebuie să salvăm această valoare înainte de a-l modifica pe u (într-o variabilă auxiliară).

1_12.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>

void main() {
  double a, b, eps, u, v, aux;
  printf("\n a = ");
  scanf("%lf", &a);
  printf("\n b = ");
  scanf("%lf", &b);
  printf("\n eps = ");
  scanf("%lf", &eps);
  u = 1 / fabs(a);
  v = 1 / fabs(b);
  while (fabs(u - v) > eps) {
  /*salvam vechea valoare a lui u(folosita in calculul lui
v):*/
    aux = u;
    u = (u + v) / 2;
    v = sqrt(aux * v);
  }
  printf(" Valaoarea integralei este %lf", u);
  getch();
}
```

R1_13. Se dau pe mediul de intrare un număr necunoscut de numere nenule terminate cu o valoare nulă. Să se stabilească dacă acestea:

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

Rezolvare: Se numără relaţiile de ordine între elementele vecine. Pot apărea următoarele situaţii:

- avem n-1 relaţii = $\Rightarrow$ şirul este constant
- avem n-1 relaţii > $\Rightarrow$ şirul este strict descrescător
- avem n+1 relaţii < $\Rightarrow$ şirul este strict crescător
- numărul relaţiilor < este 0 (în timp ce celelalte sunt nenule) $\Rightarrow$ şirul este descrescător
- numărul relaţiilor > este 0 (în timp ce celelalte sunt nenule) $\Rightarrow$ şirul este crescător
- numărul relaţiilor = este 0 (în timp ce celelalte sunt nenule) $\Rightarrow$ şirul este neordonat
- numărul relaţiilor >, = şi < sunt toate 0 $\Rightarrow$ şirul este vid sau are 1 element

Elementele din şir sunt citite pe rând. Trebuiesc reţinute numai două elemente succesive, în vederea comparării (`prec` şi `crt`). Vom folosi trei contoare de relaţii: `mic`, `egal` şi `mare`.

1_13.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
// Tipuri de siruri
#define S_CRESC     6
#define CRESC       5
#define S_DESCRESC  2
#define DESCRESC    3
#define CONST       4
#define NEORDONAT   1

void main (void) {
  int n=0;                    // numarul de elemente din sir
  int crt;                    // numarul curent citit
  int prec;                   // numarul citit anterior
  int mic=0, egal=0, mare=0;// contoare de relatii
  int tip;                    // tipul surului
  clrscr();
  // Facem citirile primelor date
  printf (" Introduceti primul numar: "); scanf ("%d", &crt);
  if (crt != 0) {
      n++;
      prec = crt;
      printf (" Introduceti al doilea numar: ");
      scanf ("%d", &crt);
      if (crt == 0) {
```



```
        printf ("Sirul are un singur element!\n");
        getch();
        exit (1);
      }
      else
        n++;
  }
  else
  { printf (" Sir vid\n");
    getch();
    exit(1);
  }
  // Sirul are cel putin doua elemente
  while (crt != 0) {
      if(prec < crt)
        mic++;
      else
        if(prec > crt)
            mare++;
        else
            egal++;
      prec = crt;
      printf (" Introduceti urmatorul numar: ");
      scanf ("%d", &crt);
      n++;
  };
  /* decrementam n, pentru ca ultimul element introdus
       (0) nu se ia in considerare: */
  n--;
  if(mic==n-1)  tip = S_CRESC;
  if(mare==n-1) tip = S_DESCRESC;
  if(egal==n-1) tip = CONST;
  if(mare==0 && mic && egal) tip = CRESC;
  if(mic==0 && mare && egal) tip = DESCRESC;
  if(mic && mare) tip = NEORDONAT;
  // Afisare rezultate
  switch (tip) {
  case S_CRESC:    printf (" Sirul este strict
crescator!\n");
                   break;
  case CRESC:      printf (" Sirul este crescator!\n");
break;
  case S_DESCRESC: printf (" Sirul este strict
escrescator!\n");
                   break;
  case DESCRESC:   printf (" Sirul este descrescator!\n");
                   break;
  case CONST:      printf (" Sirul este constant!\n"); break;
  case NEORDONAT:  printf (" Sirul este neordonat!\n");
break;
  }
  getch();
}
```

R1_14. Să se calculeze pentru n dat, f_n termenul de rangul n din şirul lui Fibonacci, cunoscând relaţia de recurenţă: $f_p = f_{p-1} + f_{p-2}$ pentru $p > 2$ şi $f_0 = 1$, $f_1 = 1$

Rezolvare: Pentru a calcula termenii şirului folosim 3 variabile: una în care reţinem termenul calculat la pasul curent (f) şi două în care reţinem cei doi termeni anteriori (f_1 şi f_2). După ce am determinat valoarea termenului curent ($f = f_1 + f_2$) trebuiesc actualizate şi valorile termenilor anteriori: $f_2 = f_1$ şi $f_1 = f$.
Am fi putut realiza aceste operaţii şi folosind doar două variabile, pentru a face economie, astfel: fie variabilele f şi f_1, în care sunt reţinuţi ultimii 2 termeni calculaţi (f_p, respectiv f_{p-1}). Facem atribuirea $f = f + f_1$ şi astfel în f avem valoarea termenului următor. Dar în f_1 trebuie să punem acum vechea valoare a lui f, care s-a pierdut. Pentru a o reconstitui calculăm $f_{nou} - f_1$, deoarece: $f_{nou} - f_1 = [f_{vechi} + f_1] - f_1 = f_{vechi}$ (am notat cu f_{nou} şi f_{vechi} valorile nouă, respectiv veche pentru f).

1_14.cpp

```
#include <stdio.h>

void main() {
    int n, i, f, f1, f2;
    printf("\n n=");
    scanf ("%d", &n);
    f = f1 = f2 = 1;
    /* in f1 retinem fp-1 si in f2 retinem fp-2 */
    for (i = 2; i <= n; i++) {
      f = f1 + f2;
      f2 = f1;
      f1 = f;
    }
    printf("\n f%d = %d \n", n, f);
}
```

R1_15. Pentru n dat să se calculeze suma: $S = \frac{1}{2} + \frac{1 \cdot 3}{2 \cdot 4} + \cdots + \frac{1 \cdot 3 \cdots (2n-1)}{2 \cdot 4 \cdots 2n}$

Rezolvare: Vom folosi două variabile (p_1 şi p_2) în care reţinem produsul numerelor impare (cel care apare la numărător), şi respectiv produsul numerelor pare.

1_15.cpp

```
#include <stdio.h>

void main() {
  int n, i;
  double s, p1, p2;
  printf("\n n =");
  scanf("%d", &n);
  p1 = 1.0; p2 = 2.0;
  s = p1 / p2;
  for (i = 2; i <= n; i++)  {
    p1 *= (2 * n - 1);
    p2 *= (2 * n);
    s += (p1 / p2);
  }
  printf("Valoarea sumei este S%d = %.3lf \n", n, s);
}
```

R1_16. Un număr perfect este un număr egal cu suma divizorilor săi, printre care este considerată valoarea 1 dar nu şi numărul.
Să se găsească toate numerele perfecte mai mici sau egale cu un număr dat pe mediul de intrare, şi să se afişeze fiecare număr astfel determinat, urmat de suma divizorilor lui. De exemplu numărul 6 are divizorii 1, 2 şi 3 este număr perfect deoarece: 6 = 1 + 2 + 3.

Rezolvare: Se calculează suma divizorilor şi se compară cu numărul – dacă sunt egale, numărul este perfect şi se afişează. Suma divizorilor se iniţializează la 1, deoarece am considerat că 1 este divizor implicit al fiecărui număr.

1_16.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

void main (void) {
  int k;
  int suma_div ;          // suma divizorilor
  int d;                  // un posibil divizor
  clrscr();
  printf (" Introduceti k: ");
  scanf ("%d", &k);
  // incercam toate numerele pana la k
  for (int x = 1; x <= k; x++)   {
    // calculam suma divizorilor lui x
```

```
    suma_div = 1;
    for (d = 2; d <= x/2; d++)
      if (x % d == 0)
        suma_div += d;
    if(suma_div==x) {
        printf ("%d=1", x);
        for (d = 2; d <= x/2; d++)
            if (x % d == 0)
              printf("+%d", d);
        printf ("\n");
      } //if
  } //for
  getch();
}
```

R1_17. Să se calculeze prin dezvoltare în serie, cu precizia eps dată, sin(x):

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

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

$\sin(x) = -\sin(-x)$	dacă $x < 0$	(1)
$\sin(x) = \sin(x - 2\pi n)$	dacă $x > 2\pi n$	(2)
$\sin(x) = -\sin(x - \pi)$	dacă $x > \pi$	(3)
$\sin(x) = \sin(\pi - x)$	dacă $x > \pi/2$	(4)

(*Probleme similare*: P1_5, P1_25.)

Rezolvare: Vom aplica relațiile (1) – (4) pe rând, modificând valoarea lui x pentru a obține o valoare din primul cadran: după ce aplicăm relația (1) x va fi pozitiv, după relația (2) va fi în intervalul [0, 2π] etc. ; eventualele schimbări de semn care pot apărea în valoarea funcției (dacă se aplică relațiile (1) și (3)) se reține într-o variabilă numită *semn*: aceasta este inițializată cu 1 și la aplicarea relațiilor (1) sau (3) este înmulțită cu –1. În fișierul math.h se găsesc definițiile mai multor constante matematice, printre care și n (denumită M_PI), pe care o vom utiliza în program.

1_17.cpp

```
#include <stdio.h>
#include <math.h>

void main(void) {
  double x, s, sv, t, eps;
  int k, n, semn;
  printf("\n x = ");
  scanf("%lf", &x);
```



```
  printf("\n eps = ");
  scanf("%lf", &eps);
  /* transformam x pentru a il aduce in primul cadran: */
  semn = 1; /* schimbarea de semn in urma transformarilor */
  if (x < 0.0)  /* sin(-x) = -sin(x) */
  { x = -x;
    semn = -semn;
  }
  while (x > 2 * M_PI) /* sin(x) = sin(x - 2*PI*n) */
      x -= 2 * M_PI;
  if (x > M_PI)  /* sin(x) = -sin(x - PI) */
  { x -= M_PI;
      semn = -semn;
  }
  if (x > M_PI/2)  /* sin(x) = sin(PI - x) */
      x = M_PI - x;
  /* calculam sin(x): */
  s = t = x; /* s - valoarea sumei; t - termenul curent */
  k = 1;
  do { sv = s;  /* salvam vechea valoare a lui s, pentru a
                   evalua eroarea */
       k += 2;
       t *= -(x * x / k / (k -1));
       s += t;
  } while (fabs(sv - s) >= eps);
  printf (" sin(x) = %lf \n", semn * s);
}
```

R1_18. Se consideră polinomul: $p_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_n$

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 $a_0, a_1, \ldots a_n$)

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

(*Probleme similare*: P1_19, P1_20)

Rezolvare:

a) Ne bazăm pe următoarele relații de recurență care se pot scrie pentru polinoamele de grad 0, 1, … n (schema lui Horner):

$$p_0(x) = a_0$$

$$p_1(x) = a_0 \cdot x + a_1 = x \cdot p_0(x) + a_1$$

$$p_2(x) = x \cdot (a_0 \cdot x + a_1) + a_2 = x \cdot p_1(x) + a_2$$

…

$$p_n(x) = x \cdot p_{n-1}(x) + a_n$$

Deci, vom folosi o variabilă (p) în care vom reține valoarea calculată până în pasul curent pentru polinom. În fiecare pas i, înmulțim valoarea respectivă cu x și apoi adunăm rezultatul obținut cu a_i. Observăm că este suficientă o singură variabilă a pentru citirea coeficienților (în pasul curent i avem nevoie doar de a_i).

b) Și în acest caz reținem valoarea polinomului într-o variabilă, la care, în pasul i, adunăm $a_{n-i}\, x^i$ (a_{n-i} este coeficientul citit) . x^i va fi păstrat în variabila xi.

1_18.cpp

```
#include <stdio.h>

void main() {
  float a, x;
  float p;  // valoarea polinomului
  float xi; // x la puterea i
  int n,i;
  printf("\n n=");
  scanf("%d",&n);
  printf(" x=");
  scanf("%f",&x);

  /* punctul a */
  for (i=0, p=0; i <= n; i++){
    printf("a%d=", i);
    scanf("%f", &a);
    p = p * x + a;
  }
  /* afisam valoarea polinomului: */
  printf("p(%.2f)=%.2f\n", x, p);
  /* punctul b */
  p=0;
  xi=1;   // initial xi = x^0
  for (i=0; i <= n; i++){
    printf("a%d=", n - i);
    scanf("%f", &a);
    p += a * xi;
    xi *= x; // xi ia valoarea x^(i+1)
  }
  printf("p(%.2f)=%.2f\n", x, p);
}
```

R1_19. Dându-se un număr întreg n, să se afișeze toți factorii primi ai acestuia precum și ordinele lor de multiplicitate. (*Probleme similare:* P1_21)

Rezolvare: Se împarte repetat numărul n la un posibil divizor d, cât timp se divide cu acesta, crescând corespunzător multiplicitatea divizorului. Când numărul nu se mai divide cu d, se afișează d împreună cu multiplicitatea lui (dacă aceasta a fost nenulă). Algoritmul se termină când numărul n devine 1.

1_19.cpp

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

void main (void) {
  int n;
  int mult=0;    // multiplicitatea unui divizor
  int div=2;     // un divizor incercat
  clrscr();
  printf (" Introduceti n: "); scanf ("%d", &n);

  while (1) {
      if(n%div==0){
        mult++;
        n/=div;
      }
      else
      { if(mult > 0)
            printf("%d %d\n", div, mult);
        mult = 0;
        if(div==2)
            div = 3;
        else
            div+=2;
        if (n==1)
            break;
      }
  }
  getch();
}
```

R1_20. 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.

Rezolvare: Pentru toate numerele cuprinse între 2 și n se calculează numărul de divizori nebanali și se reține maximul și numărul corespunzător. Dacă două numere au același număr maxim de divizori, se ia primul dintre ele.

1_20.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
```

```
void main (void) {
  int n;
  int nd;          // numarul de divizori ai unui candidat
  int nd_max = 0; // retine numarul maxim de divizori
  int imax = 0;    // retine numarul cu cei mai multi divizori
  // Citire date de intrare
  printf (" Introduceti numarul: "); scanf ("%d", &n);
  // se incearca toti candidatii pana la n
  for (int i = 2; i <= n; i++) {
      // calculeaza numarul de divizori ai lui i
      for( int div = 2, nd = 0; div <= n/2; div++)
        if(i % div == 0)
            nd++;
      // actualizeaza numarul maxim de divizori
      if (nd > nd_max) {
        nd_max = nd;
        imax = i;
      }
  }
  printf (" Numarul %d are maximul de diviori, adica %d\n",
imax, nd_max);
  getch();
}
```

R1_21. Se consideră ecuația: $ax^3+bx+c=0$. Fără a o rezolva, să se calculeze: $x_1^n + x_2^n + x_3^n$. Se dau: *a, b, c* și *n*

Rezolvare: Înmulțim ecuația cu x^{n-3}: $ax^n + bx^{n-2} + cx^{n-3} = 0$ și ținem cont de faptul că aceasta este satisfăcută de cele 3 rădăcini x_1, x_2 și x_3. Așadar:

$a(x_1^n+x_2^n+x_3^n) + b(x_1^{n-2}+x_2^{n-2}+x_3^{n-2}) + c(x_1^{n-3}+x_2^{n-3}+x_3^{n-3}) = 0$

Notând $s_n = x_1^n + x_2^n + x_3^n$, obținem relația de recurență:
$a.s_n + b.s_{n-2} + c.s_{n-3} = 0$, pentru care trebuiesc furnizate valorile inițiale:
$s_0 = x_1^0 + x_2^0 + x_3^0 = 3$
$s_1 = x_1 + x_2 + x_3 = 0$
$s_2 = x_1^2 + x_2^2 + x_3^2 = -2b$

1_21.cpp

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

void main (void) {
  int n;
  double a, b, c, s, s0, s1, s2;
```



```
  printf (" Introduceti, pe rand, n, apoi a, b si c: ");
  scanf("%d%lf%lf%lf", &n, &a, &b, &c);
  s0 = 3;
  s1 = 0;
  s2 = -2*b;
  for(int j = 3; j <= n; j++) {
      s = -(b * s1 + c*s0) / a;
      s0 = s1;
      s1 = s2;
      s2 = s;
  }
  printf("s(%d)=%6.2lf\n", n, s);
  getch();
}
```

R1_22. Să se calculeze x^n pentru x (real) și n (întreg) dați, folosind un număr cât mai mic de înmulțiri de numere reale.

Rezolvare: Se observă că x^n poate fi descompus într-un produs de factori în care pot apare x, x^2, x^4, x^8, . . . Un factor apare în produs dacă în reprezentarea binară a lui n cifra corespunzătoare este 1.

Vom forma deci pe rând cifrele din reprezentarea binară a lui n, începând cu cea mai puțin semnificativă și puterile lui x: x, x^2, x^4, x^8, . . . Puterea se înmulțește la produs dacă cifra corespunzătoare este 1. De exemplu: $x^{13} = x^8 x^4 x^1$ deoarece $13_2 = 1101$.

Cifrele reprezentării binare a lui n se obțin ca resturi ale împărțirii succesive a lui n la 2.

1_22.cpp

```
#include <stdio.h>
#include <conio.h>

void main (void) {
  int n, c, i, k;
  double x;
  long double rez;
  double px;      // puterea lui x
  clrscr();
  printf (" Introduceti x: "); scanf ("%lf", &x);
  printf (" Introduceti n: "); scanf ("%d", &n);
  for (i = 1, px=x, rez=1, k = n; k; k/=2) {
      c = k % 2;
    if(c)
      rez *= px;
    px *= px;
  };
  printf (" %lf ^ %d = %Lf\n", x, n, rez);
  getch();
}
```

R1_23. Să se calculeze funcția Bessel de speța I-a $J_n(x)$ știind că există relația de recurență: $J_p(x) = (2p-2)/x J_{p-1}(x) - J_{p-2}(x)$

$$J_0(x) = \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{x}{2}\right)^{2k}}{(k!)^2} \quad ; \quad J_1(x) = \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{x}{2}\right)^{2k+1}}{k!(k+1)!}$$

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

Rezolvare:

1_23.cpp

```
#include <stdio.h>
#include <math.h>

void main()
{ int n, k, p;
  double j;          /* valoarea functiei */
  double j0, j1;     /* valorile din iteratia curenta */
  double j0v, j1v; /* valorile din iteratia anterioara */
  double x, eps, term;
  printf("\n x = ");
  scanf("%lf", &x);
  printf("\n n = ");
  scanf("%d", &n);
  printf("\n eps = ");
  scanf("%lf", &eps);
  /* putem face validarea datelor de intrare(x!=0, n>=0, eps>
0)
     calculam j0 si j1: */
  term = 1.0;  /* termenul curent din suma */
  j0 = 1.0; j1 = x / 2;
  for (k = 1; ; k++) {
    j0v = j0; j1v = j1;
    term *= -(x * x / (4 * k * k));
    j0 += term;
    j1 += term * x / (2 * (k + 1));
    if (fabs(j0 - j0v) < eps && fabs(j1 - j1v) < eps) break;
  }
  /* calculam valoarea functiei: */
  for (p = 2; p <= n; p++) {
    j = ((2 * p - 2) / x) * j1 - j0;
    j0 = j1;
    j1 = j;
  }
  printf("Valoarea functiei Bessel: J%d(%.2lf) = %lf", n, x,
j);
}
```


R1_24. Să se obțină reprezentarea ca fracție zecimală a numărului m/n. Eventuala perioadă se afișează între paranteze.

Rezolvare: Deoarece numerele reale se reține în memorie în mod aproximativ, cu un număr finit de zecimale, rezultatul pe care ni-l furnizează calculatorul pentru împărțirea m/n nu ne ajută să determinăm dacă fracția m/n este periodică sau nu. Deci va trebui să simulăm noi împărțirea pentru partea zecimală a fracției dacă dorim să calculăm și perioada; prezentăm mai jos algoritmul folosit, împreună cu un exemplu ($m = 34$, $n = 28$):

1) Simplificăm fracția: aflăm cmmdc(m,n) (de exemplu, folosind algoritmul lui Euclid) și împărțim m și n prin această valoare;

Ex: `m = 34, n =28 => cmmdc(m, n) = 2`
`m = 17 ( = 34 / 2) n = 14 ( = 28 / 2)`

2) Calculăm și afișăm partea întreagă a fracției;

Ex: `[17 / 14] = 1 => afișăm "1."`

3) Determinăm lungimea părții neperiodice; aceasta este egală cu maximul dintre multiplicitățile factorilor 2 și 5 din descompunerea numitorului;

Ex: `n = 14 = 2^1 * 7^1 => partea neperiodică are lungime 1`

4) Simulând împărțirea cifră cu cifră, calculăm partea neperiodică:
- o nouă cifră a acesteia se obține prin împărțirea întreagă a restului parțial la numitor
- noul rest parțial este egal cu restul acestei împărțiri, înmulțit cu 10.

Inițial, restul parțial este (m % n)10 (restul împărțirii numărătorului la numitor, înmulțit cu 10);

Ex: Inițial: `Rest_parțial = (17 % 14) * 10 = 30`
Pas 1: `Cifră_cât = 30 / 14 = 2 => afișăm "2"`
`Rest_parțial = (30 % 14) * 10 = 20`

5) Dacă după ce am calculat toate cifrele din partea neperiodică, restul parțial este 0, nu există parte periodică și algoritmul s-a încheiat. Altfel, începem să determinăm și cifrele părții periodice (ne oprim atunci când obținem un rest parțial egal cu primul rest parțial)

Ex: Pas 1: `Afișăm "("`
Pas 2: `Cifră_cât = 20 / 14 = 1 => afișăm "1"`
`Rest_parțial = (20 % 14) * 10 = 60`
Pas 3: `Cifră_cât = 60 / 14 = 4 => afișăm "4"`
`Rest_parțial = (60 % 14) * 10 =40`
...
Pas n: `Afișăm ")"`

1_24.cpp

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

void main(){
  int m, n, i, auxm, auxn, r;
  int m2, m5, ncn, rp, rp1;
  printf("\n m = ");
  scanf("%d", &m);
  printf("\n n = ");
  scanf("%d", &n);
  assert(n != 0);
  /* aflam cmmdc(m,n) cu algoritmul lui Euclid: */
  auxm = m;
  auxn = n;
  do
  { r = auxm % auxn;
    auxm = auxn;
    auxn = r;
  } while (r);

  /* simplificam fractia: */
  m = m / auxm;
  n = n / auxm;
  printf("Fractia zecimala este ");
  /* afisam partea intreaga: */
  printf("%d.", m / n);
  if (m % n == 0)  /* partea zecimala a fractiei este 0 */
  { printf("0");
    exit(0);
  }
  m = m % n;
  /* calculam numarul de cifre din partea neperiodica: */
  m2 = 0;  /* multiplicitate 2 */
  for (auxn = n; auxn % 2 == 0; m2++, auxn /= 2)
    ;
  m5 = 0;  /* multiplicitate 5 */
  for (auxn = n; auxn % 5 == 0; m5++, auxn /= 5)
    ;
  ncn = (m2 > m5) ? m2 : m5;  /* numar cifre neperioada */
  /* afisam partea neperiodica: */
  rp = 10 * m;  /* rp - restul partial */
  for (i = 1; i <= ncn; i++)
  { printf("%d", rp / n); /* cifra curenta */
    rp = (rp % n) * 10;
  }
  /* afisam partea periodica, daca exista: */
  if (rp)  /* exista parte periodica */
  { printf("(");
    rp1 = rp;  /* salvam primul rest partial */
    do {
      printf("%d", rp / n);
      rp = (rp % n) * 10;
    } while ( rp != rp1);
    printf(")");
  }
}
```


Probleme propuse

P1_1. Să se calculeze coeficienții binomiali $C_n^1, C_n^2, \ldots C_n^p$ în care n și p sunt întregi pozitivi dați ($p \le n$), știind că există următoarea relație de recurență:
$C_n^k = \frac{(n-k+1)}{k} C_n^{k-1}$ pornind cu $C_n^0 = 1$.

P1_2. Pentru a, b și n dați ($a, b \in \mathbf{R}$, $n \in \mathbf{Z}$) să se calculeze x și y astfel ca: $x + i\,y = (a + i\,b)^n$, fără a folosi formula lui Moivre.

P1_3. Șirul $\{x_n\}$ generat cu relația de recurență $x_n = \frac{1}{2}\left(x_{n+1} + \frac{a}{x_{n+1}}\right)$ pornind cu $x_0 = \frac{a}{2}$ 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 șir,cu o precizie eps dată.

P1_4. Pentru calculul lui $\lg_2 x$ se generează șirurile $\{a_n\},\{b_n\}$ și $\{c_n\}$ cu relațiile de recurență:

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

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

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

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:

$$\lg_2 x = -\lg_2\left(\frac{1}{x}\right) \quad \text{pentru } x < 1 \qquad (1)$$

$$\lg_2 x = k + \lg_2\left(\frac{x}{2^k}\right) \quad \text{pentru } x \ge 2^k \qquad (2)$$

P1_5. 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: $x = i + f$ în care:
i = partea intreagă a lui x şi f = partea fracţionară a lui x. Rezultă $e^x = e^i \cdot e^f$

cu: $e^i = \underbrace{e\ e\ \dots\ e}_{i}$ pentru $i > 0$ $\quad e^i = \underbrace{\frac{1}{e}\ \frac{1}{e}\ \dots\ \frac{1}{e}}_{i}$ pentru $i < 0$

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

P1_6. Să se determine valoarea n pentru care: $S = \sum_{k=1}^{n} \frac{2}{\sqrt{4n^2 - k}}$

satisface conditia $|S - \pi/3| < \varepsilon$, in care eps este dat. Se ştie că: $\lim_{k \to \infty} S = \frac{\pi}{3}$

P1_7. Fie şirurile $\{a_n\}$, $\{b_n\}$, $\{c_n\}$ generate cu relaţiile de recurenţă:

$$a_n = \frac{(b_{n-1} + c_{n-1})}{2};\ b_n = \frac{(c_{n-1} + a_{n-1})}{2};\ c_n = \frac{(a_{n-1} + b_{n-1})}{2}$$

Se ştie că a_0 = alfa , b_0 = beta , c_0 = gama, alfa, beta, gama date. Cunoscând că cele trei şiruri sunt convergente şi au o limită comună, să se calculeze cu o precizie eps dată această limită.

P1_8. Cunoscând data curentă exprimată prin trei numere întregi reprezentând anul, luna, ziua precum şi data naşterii unei persoane exprimată în acelaşi mod, să se calculeze vârsta persoanei exprimată în ani, luni şi zile. Se consideră în mod simplificator că toate lunile au 30 de zile.

P1_9. Se citesc trei numere reale pozitive ordonate crescător. Să se verifice dacă acestea pot să reprezinte laturile unui triunghi şi în caz afirmativ să se stabilească natura triunghiului: isoscel, echilateral, dreptunghic sau oarecare şi să se calculeze aria sa.
Indicaţie: Trei segmente de lungime x, y şi z pot forma un triunghi dacă îndeplinesc relaţiile: $x + y < z$; $x + z < y$; $y + z < x$..
Se poate verifica dacă un triunghi este dreptunghic folosind teorema lui Pitagora. Aria unui triunghi se calculeaza cu formula:

$$S = \frac{1}{2} \begin{vmatrix} x_x & x_y & 1 \\ y_x & y_y & 1 \\ z_x & z_y & 1 \end{vmatrix}$$

unde $x = (x_x, x_y)$, $y = (y_x, y_y)$ şi $z = (z_x, z_y)$ sunt punctele care definesc vârfurile triunghiului.

P1_10. Cunoscând data curentă şi data naşterii unei persoane, exprimate fiecare sub forma unui triplet (an, lună, zi), să se afle vârsta persoanei în ani împliniţi.

P1_11. De pe mediul de intrare se citeşte un unghi exprimat în grade, minute, secunde. Să se convertească în radiani.

P1_12. Un număr întreg S reprezintă o durată de timp exprimată în secunde. Să se convertească în zile, ore, minute şi secunde utilizând în program cât mai puţine variabile.

P1_13. Să se scrie algoritmul pentru rezolvarea cu discuţie a ecuaţiei de gradul 1: $ax + b = 0$, cu valorile lui a şi b citite de pe mediul de intrare.

P1_14. Să se calculeze data revenirii pe pământ a unei rachete, exprimată prin an, lună, zi, oră, minut, secundă, cunoscând momentul lansării exprimat în acelaşi mod şi durata de zbor exprimată în secunde.

P1_15. Să se calculeze: $S = \sum_{i=1}^{n} i!$ când se cunoaşte n.

P1_16. Să se scrie algoritmul pentru rezolvarea a n ecuaţii de gradul 2. Se citesc de pe mediul de intrare valoarea lui n şi n tripleţi (a, b, c) reprezentând coeficienţii ecuaţiilor. Se recomandă realizarea unui program care să utilizeze cât mai puţine variabile.

P1_17. De pe mediul de intrare se citeşte un număr real b şi un şir de valori reale pozitive terminate printr-o valoare negativă (care nu face parte din şir).
Să se stabilească elementul din şir cel mai apropiat de b. Se va preciza şi poziţia acestuia. Termenii şirului vor fi păstraţi pe rând în aceeaşi variabilă a.

P1_18. De pe mediul de intrare se citeşte o listă de numere întregi pozitive terminate cu un număr negativ ce marchează sfârşitul listei. Să se scrie în dreptul fiecărei valori numărul prim cel mai apropiat mai mic sau egal cu numărul dat.

P1_19. Dându-se un număr întreg n să se afle cifrele reprezentării sale în baza 10 începând cu cifra cea mai semnificativă.

P1_20. Dându-se numărul real a $(0<a<1)$ să se determine primele n cifre ale reprezentării lui într-o bază b dată.

P1_21. 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 încerca divizibilitatea lui cu 2, 3, 4, ... x. Numărul este prim dacă nu se divide cu niciunul dintre aceste numere şi este neprim dacă are cel puţin un divizor printre ele.

P1_22. Se dau două numere întregi, primul reprezentând un an şi al doilea, numărul de zile scurse din acel an. Să se determine data (luna şi ziua).

P1_23. Se dă un întreg n. Să se calculeze şi să se afişeze:
- divizorii lui n
- divizorii primi şi multiplicităţile lor
- divizorul prim cu multiplicitatea maximă. Dacă există mai mulţi divizori cu multiplicitate maximă, se ia cel mai mare dintre ei.

P1_24 Modificaţi problema R1_17 astfel încât argumentul să fie redus la primul octant. Se ştie că: $\sin(x) = \cos\left(\frac{\pi}{2} - x\right)$ dacă $x \geq \frac{\pi}{4}$, iar seria Taylor pentru calculul cosinusului este: $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$

Capitolul 2
Funcţii

Breviar

Definire funcţie

```
tip_rezultat nume_functie( tip variabial_1, tip ariabila_2,... ) {
  declaratii;
  instructiuni;
}
```

Apelare funcţie

```
// apelarea unei funcţii care întoarce rezultatul void
nume_funcţie (listă_parametri_efectivi);
// apelarea unei funcţii care întoarce un rezultat
variabila = nume_funcţie (listă_parametri_efectivi);
```

Probleme rezolvate

R2_1. 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.

Rezolvare: Vom face interschimbări între numerele date, astfel încât să le aducem în ordinea dorită. Vom avea de făcut maximum trei interschimbări. Parametrii funcţiei de interschimbare se modifică, motiv pentru care vor fi transmişi prin referinţă.

2_1.cpp

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

// Interschimbare doua variabile
void swap (int &a, int &b) {
  int aux = a;
  a = b;
  b = aux;
}
```



```
void main (void) {
  int a, b, c;
  printf (" Introduceti pe a: "); scanf ("%d", &a);
  printf (" Introduceti pe b: "); scanf ("%d", &b);
  printf (" Introduceti pe c: "); scanf ("%d", &c);
  // Teste pentru modificarea ordinii variabilelor
  if (a > b)
    swap (a, b);
  if (a > c)
    swap (a, c);
  if (b > c)
    swap (b, c);
  // Afisare rezultate
  printf (" Sortat: a = %d \t b = %d \t c = %d.\n", a, b, c);
  getch();
}
```

R2_2. De pe mediul de intrare se citesc *n* valori întregi pozitive. Pentru fiecare element să se indice cel mai mare pătrat perfect mai mic sau egal cu el.

Rezolvare: Definim o funcţie `int patrat (int)`, care introduce pătratul perfect cel mai mare, mai mic sau egal cu numărul dat.

2_2.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

// Calculeaza cel mai mare patrat perfect mai mic sau egal cu a
int patrat (int a) {
  int i = 0;
  while (i * i <= a)
    i += 1;
  return (i * i > a) ? (i-1)*(i-1) : i*i;
}

void main (void) {
  int n, a;
  clrscr();
  printf (" Introduceti n: "); scanf ("%d", &n);
  // Prelucrare date si afisare rezultate
  for (int i = 0; i < n; i++) {
    printf (" Introduceti elementul %d: ", i + 1);
    scanf ("%d", &a);
    printf ("Patratul perfect mai mic sau egal cu x este:
%d\n",
            patrat(a));
  }
  getch();
}
```

R2_3. Să se verifice dacă un numar întreg citit de pe mediul de intrare este palindrom, adică se citeşte la fel de la stânga la dreapta şi 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.

Rezolvare: Vom face împărţiri repetate ale numărului dat la 10, construind cu ajutorul restului acestor împărţiri numărul invers. Acest algoritm este implementat în funcţia `palindrom.()`

2_3.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

// Verifica daca n este palindrom
int palindrom (int n) {
  int n_salv = n;          // salvam pe n
  int n_inv = 0;           // numarul rasturnat al lui n
  while (n) {
    n_inv = n_inv * 10 + n % 10;
    n /= 10;
  }
  return (n_salv == n_inv);
}
void main (void) {
  int n;
  printf (" Introduceti numarul: "); scanf ("%d", &n);
  // Afisare rezultate
  if (palindrom(n))
    printf (" Numarul este palindrom\n");
  else
    printf (" Numarul NU este palindrom\n");
  getch();
}
```

R2_4. Se citeşte un întreg *n* şi n perechi (*a, b*) de întregi. Să se afişeze acele perechi al căror cmmdc este un număr prim.

Rezolvare:

2_4.cpp

```
#include <stdio.h>
#include <conio.h>

// calcul cmmdc cu algoritmul lui Euclid
int cmmdc (int a, int b) {
  int r;
   do { r = a % b;
        a = b;
        b = r;
  } while (r);
  return a;
}

// stabileste daca n este prim
int prim(int n) {
  int div;
  for (div=2; div*div<=n; (div==2) ? div=3 : div+=2)
      if(n % div == 0) return 0;
  return 1;
}

void main (void) {
  int n;
  int a, b;
  // Citire date de intrare
  printf ("Introduceti numarul de perechi: ");
  scanf ("%d", &n);
  for(int i=0; i<n; i++) {
    printf (" Introduceti perechea %d: ", i);
    scanf("%d%d", &a,&b);
    if (prim(cmmdc(a, b)))
      printf("cmmdc(%d,%d)=%d\n", a, b, cmmdc(a,b));
  };
  getch();
}
```


R2_5. Se citeşte un întreg *n* şi *n* numere naturale. Să se afişeze acele numere care au indicatorul lui Euler o putere a lui 2.
Indicatorul lui Euler al unui număr *x* este numărul de numere naturale mai mici ca *x* şi prime cu el.

Rezolvare:

2_5.cpp

```
#include <stdio.h>
#include <conio.h>

// calcul cmmdc cu algoritmul lui Euclid recursiv
int cmmdc (int a, int b) {
  if(b==0)
    return a;
    return cmmdc(b, a % b);
}
```

```
// calculeaza indicatorul lui Euler al numarului n
int iEuler(int n) {
  int ind=0, j;
  for(j=2; j<n; j++)
    if(cmmdc(n, j)==1)
      ind++;
  return ind;
}
// stabileste daca n este o putere a lui 2
int putere2(int n){
  while(n % 2 == 0)
    n /= 2;
  return n==1;
}
void main (void) {
  int n;
  int x;
  // Citire date de intrare
  printf (" Introduceti numarul de numere: "); scanf ("%d",
&n);
  for(int i=0; i < n; i++){
      scanf("%d", &x);
      if(putere2(iEuler(x)))
        printf("%d\n",x);
  }
  getch();
}
```

R2_6. Să se stabilească codomeniul D al valorilor funcției: $f : [x_1, x_2] \to D$, $f(x)=ax^2+bx+c$, $a,b,c \in \mathbf{R}$, $a \neq 0$. Se cunosc a, b, c, x_1, x_2.

Rezolvare: Avem trei cazuri:

a) $x_1, x_2 < -\frac{b}{2a}$ în acest caz $D = [\min(f(x_1),f(x_2)),\ \max(f(x_1),f(x_2))]$

b) $x_1 < -\frac{b}{2a} < x_2$ - în acest caz trebuie să vedem dacă $a>0$ sau $a<0$; astfel:

- dacă $a > 0$, $D = [f\left(-\frac{b}{2a}\right), \max\ (f(x_1), f(x_2))]$
- dacă $a < 0$, $D = [\min(f(x_1),f(x_2)),\ f\left(-\frac{b}{2a}\right)]$

c) $x_1, x_2 > -\frac{b}{2a}$ - în acest caz $D=[\min(f(x_1),f(x_2)),\ \max(f(x_1), f(x_2))]$

Vom folosi o funcție care calculează $f(x)$, precum și două funcții care întorc minimul și maximul dintre două numere date ca parametri.

2_6.cpp

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

// Calculeaza valoarea functiei intr-un punct
double f(double a, double b, double c, double x) {
  return a*x*x+b*x+c;
}

// Calculeaza minimul a doua numere
double min (double a, double b) {
  return (a>b)?b:a;
}

// Calculeaza maximul a doua numere
double max (double a, double b) {
  return (a>b)?a:b;
}

void main (void) {
  double a,b,c,x1,x2;
  clrscr();
  a = 0;
  while (!a) {
    printf (" Introduceti a (diferit de 0): ");
    scanf ("%lf", &a);
  }
  printf (" Introduceti b: ");
  scanf ("%lf", &b);
  printf (" Introduceti c: ");
  scanf ("%lf", &c);
  printf (" Introduceti x1: ");
  scanf ("%lf", &x1);
  printf (" Introduceti x2: ");
  scanf ("%lf", &x2);
  // Daca ambele radacini sunt mai mari sau mai mici
  // decat -b/2a avem f(x1) si f(x2) extremitati
  if ( ((x1 < -b/(2*a)) && (x2 < -b/(2*a))) || \
       ((x1 > -b/(2*a)) && (x2 > -b/(2*a))) )
    printf (" Domeniul de valori este [%lf,%lf] .", \
          min (f(a,b,c,x1), f(a,b,c,x2)), \
          max (f(a,b,c,x1), f(a,b,c,x2)));
  else
  { if (a > 0) {
      // Altfel, f(-b/2a) este minimul functiei

      printf (" Domeniul de valori este [%lf,%lf] .", \
            f(a,b,c,-b/(2*a)), max (f(a,b,c,x1),
f(a,b,c,x2)));
    }
    else
```

```
    { // Aici, f(-b/2a) este maximul functiei
      printf (" Domeniul de valori este [%lf,%lf] .", \
          min (f(a,b,c,x1), f(a,b,c,x2)), f(a,b,c,-
b/(2*a)));
    }
  }
  getch();
}
```

R2_7. Să se rezolve ecuațiile $f_1(x)=x^3+px+q=0$ și $f_2(x)=rx^2+sx+t=0$, cu precizia eps dată, știind că are fiecare o rădăcină într-un intervalul $[a, b]$ precizat. Se va utiliza metoda înjumătățirii intervalului (*metoda bisecției*).

Rezolvare: Se împarte intervalul $[a, b]$ în două părți 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 eps sau valoarea $|f(m)| <$ eps.

Vom folosi două funcții, una care aplică propriu-zis acest algoritm (bisecție) și una care calculează valoarea funcției $f()$ într-un punct x.

Funcția bisectie () are 3 parametri: capetele intervalului inițial a și b și precizia eps. Ea poate localiza rădăcina *unei singure funcții* și anume, $f(x)$.

Dacă se dorește ca funcția de localizare a unei rădăcini să fie aplicabilă oricărei funcții aceasta trebuie dată în lista de parametri (al patrulea parametru), ca pointer la funcție, precizând tipurile parametrilor funcției și $f()$ tipul rezultatului întors de funcție. Dată fiind diversitatea formelor pe care le poate avea funcția $f()$ – polinom, ecuație transcendentă etc., vom considera această funcție cu un singur parametru x de tip float, cu valoare de tip float, adică `float f(float)`, parametrul apărând în funcția `bisecție()` sub forma `float (*f)(float)`. Funcția bisecție va fi apelată de două ori, cu ultimul parametru f_1, respectiv f_2.

Pentru a simplifica transmiterea parametrilor funcțiilor $f_1() – p$ și q și $f_2() – r, s, t$ între funcția `main()` și funcțiile respective, vom utiliza variabile globale.

2_7.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

float p, q;     //variabile globale communicate functiei f1
float r, s, t; //variabile globale communicate functiei f2

// Calculeaza valoarea functiei f1 intr-un punct x
float f1 (float x) {
  return x*x*x + p*x +q;
}
```



```
// Calculeaza valoarea functiei f2 intr-un punct x
float f2 (float x) {
  return  r*x*x + s*x +t;
}

// Aplica metoda bisectiei pentru o functie data ca parametru
// si intoarce radacina aproximativa x.
double bisectie(float a, float b, float eps,
float(*f)(float)) {
  float x, y;
  // Trebuie sa verificam daca nu cumva radacina
  //   este chiar mijlocul intervalului de intrare,
  //   pentru ca in acest caz nu se mai intra in ciclul
while.
  int passed = 0;
  while ( (b - a > eps) && (fabs( f((a+b)/2) ) > eps) ) {
    x = (a+b)/2;
    y = f(x);
    if ( f(a) * y < 0 )
      b = x;
    else
      a = x;
    passed = 1;
  }
  return (passed)?x:(a+b)/2;
}

void main (void) {
  float a, b;    // Limitele intervalului de separare a
radacinii
  float eps;     // Precizia calculului
  clrscr();
  printf (" Introduceti precizia: ");
  scanf ("%f", &eps);
  printf (" Introduceti parametrii functiei f1: p si q: ");
  scanf ("%f%f", &p, &q);
  printf (" Introduceti parametrii functiei f2: r,s si t: ");
  scanf ("%f%f%f", &r, &s, &t);
  printf (" Capetele intervalului pentru functia f1: ");
  scanf ("%f%f", &a, &b);
  printf (" Radacina este: %f\n", bisectie(a, b, eps, f1));
  printf (" Capetele intervalului pentru functia f2: ");
  scanf ("%f%f", &a, &b);
  printf (" Radacina este: %f\n", bisectie(a, b, eps, f2));
  getch();
}
```

R2_8. De pe mediul de intrare se citesc cifrele reprezentării unui număr întreg în baza 16, urmate de caracterul *H* (cifrele hexazecimale sunt 0,...,9, A, B, C, D, E, F). Să se calculeze şi să se afişeze reprezentarea numărului în baza 10.

Rezolvare: Un număr care are k cifre $n_0 n_1 \ldots n_{k-1}$ în baza 16 se scrie în baza 10 ca fiind:

$16^0 n_{k-1} + 16^1 n_{k-2} + \ldots + 16^{k-2} n_1 + 16^{k-1} n_0$.

Vom lua în considerare si faptul că utilizatorul poate introduce atât `'a'` cât şi `'A'` pentru a reprezenta pe 10 în baza 16, atât `'b'` cât şi `'B'` pentru 11, etc.

Funcţia `strlen()` folosită în program întoarce lungimea unui şir de caractere pe care îl primeşte ca parametru.

2_8.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <math.h>

// Scrie reprezentarea in baza 10 a lui n
int convert (char *n) {
  int result = 0;
  for (int i = strlen(n) - 1; i >= 0; i--) {
    if (n[i] >= 'a')
      result += (n[i] - 'a' + 10) * pow(16, strlen(n) - i -
1);
    else
      if (n[i] >= 'A')
      result += (n[i] - 'A' + 10) * pow(16, strlen(n) - i -
1);
      else
      if (n[i] >= '0')
        result += (n[i] - '0') * pow(16, strlen(n) - i - 1);
  }
  return result;
}

void main (void) {
  char n[10];
  clrscr();
  printf (" Introduceti numarul: "); scanf ("%s", n);
  printf (" Numarul in baza 10 este: %d", convert(n));
  getch();
}
```


R2_9. 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 abundent 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 $sd = 1 + 2 + 7 < 14$.

a) Definiţi o funcţie având ca parametru un număr întreg n, funcţie care întoarce ca rezultat –1, 0 sau 1 după cum numărul este deficient, perfect sau abundent.

b) Scrieţi o funcţie `main()` care citeşte două valori întregi x şi y şi clasifică toate numerele naturale cuprinse între x şi y în numere deficiente, perfecte sau abundente.

Rezolvare:

2_9.cpp

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#define DEFICIENT -1
#define PERFECT 0
#define ABUNDENT 1

/* calculul sumei divizorilor unui numar: */
int suma_divizori(unsigned int n) {
  int i, s = 1;
  for (i = 2; i <= n/2; i++)
    if (n % i == 0) s += i;
  return s;
}
/* functie care intoarce tipul numarului (DEFICIENT / PERFECT
/ ABUNDENT): */
int tip_numar(unsigned int n) {
  int s = suma_divizori(n);
  if (s < n) return DEFICIENT;
  if (s > n) return ABUNDENT;
  return PERFECT;
}

void main(void) {
  int x, y;
  int i;

  /* citirea si validarea datelor de intrare: */
  do {
    printf(" x = ");
```

```
    scanf("%d", &x);
    printf(" y = ");
    scanf("%d", &y);
    if (x <= 0 || y <= 0 || x > y)
      printf(" Trebuie ca 0 < x <= y  \n");
  } while (x <= 0 || y <= 0 || x > y);

  for (i = x; i <= y; i++)
    switch (tip_numar(i)) {
      case DEFICIENT:
           printf("%d este DEFICIENT\n",i);
           break;
        case PERFECT:
           printf("%d este PERFECT\n",i);
           break;
        case ABUNDENT:
           printf("%d este ABUNDENT\n",i);
      }
  getch();
}
```

R2_10. Dându-se numărul întreg a să se determine primele n cifre ale reprezentării lui într-o bază b dată.

Rezolvare:

Vom avea în vedere resturile repetate ale împărțirii lui a la b, până când câtul este 0. Spre exemplu, dacă vrem să convertim pe $28_{(10)}$ în baza 2 avem:

28 % 2 = 0, 28 / 2 = 14
14 % 2 = 0, 14 / 2 = 7
7 % 2 = 1, 7 / 2 = 3
3 % 2 = 1, 3 / 2 = 1
1 % 2 = 1, 1 / 2 = 0.

Reprezentarea în baza b este dată de citirea resturilor în ordine inversă (din acest motiv funcția de calcul trebuie să fie recursivă). Pentru exemplul dat, reprezentarea în baza 2 este 11100.

2_10.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

// Scrie reprezentarea in baza b a lui n
```



```
void convert (int n, int b) {
  if (n / b > 0)
    convert (n / b, b);
  if (b < 10)
    printf ("%d", n % b);
  else
    if (n % b > 10)
      printf ("%c", (n % b) - 10 + 'a');
    else printf ("%d", n % b);
}
void main (void) {
  int n, b;
  clrscr();
  printf (" Introduceti numarul: "); scanf ("%d", &n);
  printf (" Introduceti baza: "); scanf ("%d", &b);
  printf (" Numarul in baza %d este: ", b);
  convert(n,b);
  getch();
}
```

R2_11. Scrieți un program C care conține următoarele funcții:

a) Funcție care calculează cmmdc a două numere naturale.
b) Funcție care simplifică o fracție rațională prin cmmdc al numărătorului și numitorului. Funcția are doi parametri pointeri la numărătorul și numitorul fracției și nu întoarce nici un rezultat.
c) Funcție care adună două fracții raționale, obținând tot o fracție rațională. Funcția are 6 parametri: 4 întregi reprezentând cele două fracții care se adună și doi pointeri către numărătorul și numitorul fracției raționale rezultat.

Funcția `main()`:

1. Citește un întreg n și n fracții raționale, pe care le simplifică
2. Calculează suma p/q a celor n fracții și afișează numerele p și q.

Rezolvare: Vom aduna fracțiile, pe măsură ce le citim, la suma deja obținută (pentru a nu fi nevoie să le memorăm pe toate). La sfârșit simplificăm și fracția p/q (care reprezintă suma lor).

Funcția `assert` folosită în program testează dacă o expresie primită ca parametru este adevărată și în caz contrar încheie execuția programului.

2_11.cpp

```
#include <stdio.h>
#include <conio.h>
#include <assert.h>
/*
 * cel mai mare divizor comun al numerelor a si b
```

```
 * (calculat cu algoritmul lui Euclid):
 */
int cmmdc(int a, int b) {
  if(b==0)
    return a;
    return cmmdc(b, a % b);
}
/* simplificarea fractiei a / b : */
void simplifica(int &a, int &b) {
  int d;
  d = cmmdc(a, b);
  a /= d;
  b /= d;
}
/*
 * adunarea a doua fractii:
 * as / bs <- a1 / b1 + a2 / b2
 */
void aduna(int a1, int b1, int a2, int b2, int &as, int &bs)
{
  simplifica(a1, b1);
  simplifica(a2, b2);
  as = a1 * b2 + a2 * b1;
  bs = b1 * b2;
  simplifica(as, bs);
}
void main() {
  int n;     // numarul de fractii
  int a, b;  // numaratorul si numitorul fractiei citite
  int p, q;  // numaratorul si numitorul rezultatului
  printf("\n Numarul de fractii = ");
  scanf("%d", &n);
  p = 0; q = 1;
  for (int i = 1; i <= n; i++) {
    printf("Numarator%d = ", i);
    scanf("%d", &a);
    printf("Numitor%d = ", i);
    scanf("%d", &b);
    assert(b != 0);
    simplifica(a, b);
    aduna(a, b, p, q, p, q);
  }
  simplifica(&p, &q);
  printf ("Rezultatul este : %d / %d", p, q);
  getch();
}
```


R2_12. Să se scrie în C:

a) O funcție pentru calculul valorii funcției $f(x)$:

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

b) O funcție pentru calculul integralei definite prin metoda trapezelor, Cu n pași egali, pe intervalul $[a, b]$, după formula

$$\frac{h}{2}\left(f(a)+f(b)+2\sum_{i=1}^{n-1} f(a+ih) \right) \quad \text{unde} \quad h=\frac{b-a}{n}.$$

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

Rezolvare: Vom implementa direct formulele date în funcțiile `f` și `int_f`.

2_12.cpp

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
double f(double x) {
  if (x <= -1)
    return -x;
  if (x >= 1)
    return x;
  else return sqrt(1 - x*x);
}

double int_f (double a, double b, int n, double (*f)(double))
{
  double h = (b - a) / (double)n;
  double result = f(a) + f(b);
  for (int i = 1; i <= n-1; i++)
    result += (2*f(a+(double)i*h));
  return result*(h/2);
}
```

```
void main (void) {
  double x;
  int n;
  double a, b, eps;
  double int_f_1, int_f_2;
  printf (" Introduceti x: ");
  scanf ("%lf", &x);
  printf (" Limita inferioara a intervalului de integrare:
");
  scanf ("%lf", &a);
  printf (" Limita superioara a intervalului de integrare:
");
  scanf ("%lf", &b);
  printf (" Precizia de integrare: ");
  scanf ("%lf", &eps);
  // Calculam f(x) si afisam rezultatul
  printf (" f(%lf) = %lf\n", x, f(x));
  // Calculam integrala si afisam rezultatul
  int_f_1 = int_f(a, b, 10, &f);
  int_f_2 = int_f(a, b, 20, &f);
  n = 20;
  while ( fabs(int_f_2 - int_f_1) > eps ) {
    int_f_1 = int_f_2;
    n += 10;
    int_f_2 = int_f(a, b, n, &f);
  }
  printf (" Integrala din f pe [%.2lf,%.2lf] este: %lf\n", a, b,
          int_f_2);
  getch();
}
```

R2_13. 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.

Rezolvare: Vom apela repetat funcția astfel:

$cmmdc(n_1, n_2, \ldots, n_n) = cmmdc\ (cmmdc(n_1, n_2, \ldots, n_{n-1}), n_n)$

$cmmdc(n_1, n_2, \ldots, n_{n-1}) = cmmdc(cmmdc(n_1, n_2, \ldots, n_{n-2}), n_{n-1})$ etc.

2_13.cpp

```
#include <stdio.h>
#include <conio.h>
// Calculeaza cmmdc-ul a doua numere
int cmmdc (int a, int b) {
  if (a > b) return cmmdc (a-b, b);
  if (a < b) return cmmdc (a, b-a);
  return a;
}
void main (void) {
```



```
  int lista[30], n;
  int result;           // Cmmdc-ul celor n numere
  // Citire date de intrare
  printf (" Introduceti n: "); scanf ("%d", &n);
  for (int i = 0; i < n; i++) {
    printf (" Numarul %d: ", i+1);
    scanf ("%d", &lista[i]);
  }
  // Calculul cmmdc-ului si afisarea rezultatului
  result = cmmdc (lista[0], lista[1]);
  for (i = 2; i < n; i++)
    result = cmmdc (result, lista[i]);
  printf (" Cmmdc-ul tuturor numerelor este: %d\n", result);
  getch();
}
```

R2_14. 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.

Rezolvare: Vom folosi o funcție care verifică dacă un număr este prim. Ne vom deplasa, pe rând, cu câte un număr la stânga și la dreapta pe axa numerelor naturale până ce găsim numărul căutat. Astfel, dacă numărul este 27, vom testa pe 28 și 26, apoi pe 29 și 25, oprindu-ne când am găsit un număr prim (aici 29).

2_14.cpp

```
#include <stdio.h>
#include <conio.h>

// Verifica daca un numar este prim
int prim (int k) {
  for (int i = 2; i < k/2; i++)
    if (k % i == 0) return 0;
  return 1;
}

void main (void) {
  int lista[30], n;
  // Pentru calculul numarului prim cel mai apropiat
  int index;
  // Citire date de intrare
  printf (" Introduceti n: "); scanf ("%d", &n);
```

```
  for (int i = 0; i < n; i++) {
     printf (" Numarul %d: ", i+1);
     scanf ("%d", &lista[i]);
   }
    // Calculul numarului prim cel mai apropiat
   for (i = 0; i < n; i++) {
     printf (" Cel mai aproape de %d : ", lista[i]);
     index = 0;
     while (!prim(lista[i] - index) && !prim(lista[i] +
index)) {
       index += 1;
       if (prim(lista[i] + index))
       printf ("%d ", lista[i] + index);
       if (prim(lista[i] - index))
       printf ("%d ", lista[i] - index);
     }
     // Trebuie sa afisam chiar numarul, daca el este prim
     if (!index)
       printf ("%d", lista[i]);
     printf ("\n");
   }
   getch();
}
```

Probleme propuse

P2_1. O pereche de numere naturale a şi b se numesc numere prietene, dacă suma divizorilor unuia dintre numere este egală cu celălalt număr (şi invers). De exemplu 220 şi 284 sunt numere prietene deoarece:
$sd(220) = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$
$sd(284) = 1 + 2 + 4 + 71 + 142 = 220$

a) Scrieţi o funcţie având ca parametru un număr natural, care întoarce suma divizorilor numărului.

b) Scrieţi o funcţie având ca parametri două numere naturale, care întoarce 1 sau 0, după cum cele două numere sunt sau nu prietene.

c) Scrieţi o funcţie `main()`, care în intervalul x, y dat găseşte toate perechile de numere prietene şi le afişează.

P2_2. Se consideră funcţia $f(x) = \ln(1 + x^2)$. Să se scrie un program care tabelează funcţia pe n intervale, fiecare interval fiind precizat prin capetele $a[i]$ şi $b[i]$ şi pasul de afişare $h[i]$.

P2_3. Să se scrie toate descompunerile unui număr par ca o sumă de două numere prime. Se va utiliza o funcţie care stabileşte dacă un număr este sau nu prim.

P2_4. Se citesc mai multe numere pozitive, terminate printr-o valoare negativă. Dintre acestea să se afişeze acelea care sunt egale cu suma cuburilor cifrelor lor. De exemplu: $153 = 1^3 + 5^3 + 3^3$. Numerele astfel găsite se afişează câte 5 pe o linie. Se va defini o funcţie pentru calculul sumei cuburilor cifrelor unui număr dat ca parametru.

P2_5. De la tastatură se citesc mai multe valori întregi şi pozitive, terminate printr-o valoare negativă. După fiecare valoare citită se va afişa aceasta, urmată de cubul cel mai apropiat, între ele afişându-se un număr de caractere '*' egal cu diferenţa dintre ele. Afişarea se va face ordonat (adică începând cu valoarea cea mai mica). Se va defini şi utiliza o funcţie care calculează cubul cel mai apropiat de un număr dat ca parametru.
Exemplu:

```
25
25**27
14
8******14
64
6464
-4
```

P2_6. Se citeşte de la tastatură un număr întreg şi pozitiv n. Să se scrie un program care determină numărul cuprins între 2 şi n are suma divizorilor nebanali maximă (1 şi n nu sunt consideraţi divizori). Se va defini o funcţie care calculează suma divizorilor unui număr natural dat ca parametru. Dacă există mai multe asemenea numere se va afişa numai primul dintre ele.
Exemplu:

```
100
96 are suma divizorilor 155
```

P2_7. De la tastatură se introduc: un număr întreg pozitiv p şi mai multe numere de asemeni întregi şi pozitive, terminate printr-o valoare negativă.
Să se scrie un program care, după fiecare număr introdus îl mai afişează o dată, dacă acesta are exact p divizori nebanali. Se va defini o funcţie care calculează numărul de divizori nebanali a unui număr dat ca parametru. La sfârşit se va afişa câte asemenea numere s-au găsit.
Exemplu:

```
4
6
12
12 are 4 divizori
25
16
20
20 are 4 divizori
-1
S-au gasit 2 numere avand cate 4 divizori nebanali
```

P2_8. De la tastatură se introduc mai multe numere întregi şi pozitive, terminate printr-o valoare negativă. După fiecare număr introdus, se va verifica dacă este termen din şirul lui Fibonacci, afişându-se în caz afirmativ poziţia pe care o ocupă în şirul Fibonacci. La sfârşit se va afişa numărul de numere Fibonacci găsite. Se va defini o funcţie care verifică dacă un număr aparţine şirului Fibonacci, funcţie care întoarce poziţia numărului în şir sau -1, dacă nu aparţine şirului.

P2_9. a) Definiţi o funcţie având ca parametri doi întregi (`an` şi `lună`), care întoarce ca rezultat ultima zi din lună.

b) Definiţi o funcţie cu 3 parametri: `an, luna, zi`, care întoarce rezultatul 1/0, după cum data este corectă sau nu.

c) Scrieţi o funcţie `main()` care:
- citeşte o dată
- stabileşte dacă este corectă sau nu
- în caz că este corectă, calculează a câta zi din an reprezintă acea dată.

Capitolul 3
Tablouri unidimensionale (vectori) şi pointeri

Breviar

vector alocat static `tip nume[dimensiune];`

declarare pointer `tip *nume;`

vector alocat dinamic in C

```
tip *var_pointer=(tip*) malloc(dimensiune);
tip *var_pointer=(tip*) calloc(dimensiune, unitate);
```

eliberare memorie dinamica in C `free var_pointer;`

vector alocat dinamic in C++ `tip *var_pointer=new tip[dimensiune];`

eliberare memorie dinamică în C++ `delete [] var_pointer;`

Probleme rezolvate

R3_1. Dându-se un număr întreg *n* să se afişeze 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 condiţii ca LXXXXVIIII şi nu ca XCIX.

Rezolvare: Vom folosi un vector cifrom pentru valorile fiecărui simbol cu cifre romane. Ideea este de a scădea din *n* valoarea cifrom[*i*], cu *i* maxim astfel încât diferenţa să fie pozitivă. Algoritmul se termină când diferenţa este 0.

3_1.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
```

```
const char letters[] = {"MDCLXVI"};
const int cifrom[] = {1000, 500, 100, 50, 10, 5, 1};

// Scrie reprezentarea cu cifre romane a lui n
void convert (int n) {
  for (int i = 0; i < strlen(letters) && n; i++)
    while (n >= cifrom[i]) {
      printf ("%c", letters[i]);
      n -= cifrom[i];
    }
}

void main (void) {
  int n;
  printf (" Introduceti numarul: "); scanf ("%d", &n);
  printf (" Numarul cu cifre romane este: ");
  convert(n);
  getch();
}
```

R3_2. Să se scrie algoritmul pentru "casierul automat" care citeşte de pe mediul de intrare suma (întreagă) datorată de un client şi calculează "restul" pe care acesta îl primeşte în număr minim de bancnote şi monezi de 100000, 50000, 10000, 5000, 1000, 500, 100, 50, 25, 10, 5, 3 si 1 leu considerând că suma plătită este cel mai mic multiplu de 100000 mai mare decât suma datorată.

Rezolvare: Vom folosi o funcţie care întoarce cel mai mic multiplu de 100000 mai mare decât suma datorată (pentru a afla suma plătită). Plata se va face "iterativ": se vor da cât mai multe bancnote de 100000 posibile, apoi cât mai multe bancnote de 50000 posibile, apoi de 10000, etc. până se epuizează suma datorată.

3_2.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

// Calculeaza cel mai mic multiplu de 100000
//   al sumei date ca parametru
long multiplu (long suma) {
  if (suma % 100000 == 0)
    return suma;
  return ((suma / 100000) + 1) * 100000;
}
```



```
void main (void) {
  long suma;
  int i;
  // Valorile monezilor
  const long valori[13] = {100000, 50000, 10000, 5000, 1000,
\
                           500, 100, 50, 25, 10, 5, 3, 1};
  // Cantitatile din fiecare moneda
  int cantitati[13];
  // Cel mai mic multiplu de 100000 al sumei de plata
  long platit;
  // Restul de plata
  long rest;
  clrscr();
  printf (" Introduceti suma de plata: ");
  scanf ("%ld", &suma);
  // Prelucrarea datelor
  platit = multiplu (suma);
  rest = platit - suma;
  printf (" Restul de plata este: %ld\n", rest);
  // Resetam cantitatile din fiecare bancnota
  for (i = 0; i < 13; i++)
    cantitati[i] = 0;
  // Calculam cantitatile din fiecare bancnota
  for (i = 0; (i < 13) && (rest > 0); i++) {
    cantitati[i] = rest / valori[i];
    rest %= valori[i];
  }
    for (i = 0; i < 13; i++)
    if (cantitati[i])
      printf (" Avem %d bancnote de valoare %ld\n", \
              cantitati[i], valori[i]);
  getch();
}
```

R3_3. Scrieţi un program C care conţine:

- O funcţie care scade două polinoame date prin grad şi tabloul coeficienţilor.
  Un polinom de gradul n are forma: $P_n(x) = a_0 + a_1x + \ldots + a_nx_n$.
- Lista de parametri conţine gradele şi tablourile coeficienţilor celor două polinoame care se scad şi ca rezultate gradul şi tabloul coeficienţilor polinomului diferenţă. Funcţia nu întoarce nici un rezultat.
- O funcţie care normalizează un polinom dat prin grad şi tabloul coeficienţilor (modifică gradul, astfel încât coeficientul puterii celei mai mari să fie nenul).
- O funcţie `main()`, în care se citesc două polinoame, date fiecare prin grad şi tabloul coeficienţilor, se scad, se normalizează polinomul diferenţă şi se afişează.

Rezolvare: Diferența dintre două polinoame p_1 și p_2 se calculează astfel:

- gradul polinomului diferență este maximul dintre gradele celor două polinoame
- coeficienții se determină scăzând coeficienții lui p_2 din cei ai lui p_1 (dar trebuie să ținem cont și de faptul că polinoamele pot avea grade diferite)

3_3.cpp

```
#include <stdio.h>
#include <conio.h>

/* diferenta a doua polinoame:
 * p1, p2 - coeficeintii polinoamelor;
 * pd - coeficientii polinomului diferenta
 * n1, n2, nd - gradele polinoamelor
 */
void diferenta_pol(float p1[], int n1, float p2[], int n2,
float pd[], int& nd){
  int i, j;
  if (n1 > n2) { // descazutul are gradul mai mare
    nd = n1;
    /* copiem primii coeficienti ai descazutului în polinomul
diferenta: */
    for (i = n1; i > n2; i--)
      pd[i] = p1[i];
  }
  else  // scazatorul are gradul mai mare sau gradele sunt
egale
  { nd = n2;
    for (i = n2; i > n1; i--)
      pd[i] = -p2[i];
  }
  for (j = i; j >= 0; j--)
    pd[j] = p1[j] - p2[j];
}


/* normalizarea unui polinom: */
void normalizeaza(float p[], int& grad) {
  for (; p[grad] == 0 && grad >= 0; grad--)
    ;
}

void main(void) {
  int n1, n2, nd;
  int i;
  float p1[100], p2[100], pd[100];
  printf("Gradul primului polinom = ");
  scanf("%d", &n1);
```



```
  printf("Introduceti coeficientii pentru primul
polinom:\n");
  for (i = n1; i >= 0; i--) {
    printf("a%d = ", i);
    scanf("%f", &p1[i]);
  }
  printf("Gradul celui de-al doilea polinom = ");
  scanf("%d", &n2);
  printf("Coeficientii pentru al doilea polinom:\n");
  for (i = n2; i >= 0; i--) {
    printf("a%d = ", i);
    scanf("%f", &p2[i]);
  }
  diferenta_pol(p1, n1, p2, n2, pd, nd);
  normalizeaza(pd, nd);
  printf("Polinomul diferenta este: \n");
  if (nd < 0)
    printf ("Polinomul nul");
  else
    for (i = nd; i >= 0; i--)
      printf(" a%d = %6.2f ", i, pd[i]);
  getch();
}
```

R3_4. Să se rezolve ecuația $P(x) = 0$, prin metoda tangentei, pornind cu un $x^{(0)}$ dat, și calculând $x^{(k+1)} = x^{(k)} - \dfrac{P(x^{(k)})}{P'(x^{(k)})}$ cu o precizie dată eps, care se atinge când $|x^{(k+1)} - x^{(k)}| < \text{eps}$.

Rezolvare: Metoda tangentei (denumită și metoda Raphson – Newton) presupune cunoașterea unei aproximații inițiale (notată în enunț cu $x^{(0)}$) pentru rădăcina ecuației. Pornind de la această valoare inițială se calculează, cu formula dată, alte aproximații $x^{(1)}$, $x^{(2)}$,... care ar trebui să fie din ce în ce mai apropiate de rădăcina exactă (acest lucru poate să nu se întâmple dacă valoarea lui $P'(x)$ este foarte apropiată de 0 sau dacă aproximația inițială nu era suficient de bună). Ne vom opri din calculul valorilor $x^{(k)}$ când diferența dintre două valori consecutive devine mai mică decât precizia. Se poate pune și condiția de a nu depăși un anumit număr de iterații, pentru cazul în care, din motivele arătate mai sus, metoda nu conduce la o aproximație bună pentru rădăcină și apare riscul unui ciclu infinit.

În program avem nevoie de o funcție care să calculeze valoarea unui polinom într-un punct și, de asemenea, de o funcție care să deriveze un polinom. Determinarea aproximațiilor succesive se va face într-un ciclu în cadrul căruia reținem atât valoarea calculată la pasul curent (`x_crt`), cât și pe cea de la pasul anterior (`x_prec`) (deoarece este conținută în condiția de ieșire).

3_4.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
#include <conio.h>
/*
 * calculul valorii unui polinom:
 * p - coeficientii polinomului; n - gradul polinomului
 * x - punctul in care se calculeaza
 */
float val_polinom(float p[], int n, float x) {
  int i;
  float val;
  val = p[n];
  for (i = n - 1; i >= 0; i--)
    val = val * x + p[i];
  return val;
}
/*
 * derivarea unui polinom:
 * p, n - coeficientii si gradul polinomului
 * pd, nd - coeficientii si gradul polinomului derivat
 */
void deriveaza_polinom(float p[], int n, float pd[], int& nd)
{
  int i;
  if (n == 0) {
    nd = 0;
    pd[0] = 0;
  }
  else
  { nd = n - 1;
    for (i = nd; i >= 0; i--)
      pd[i] = (i + 1) * p[i+1];
  }
}

void main(void) {
  float p[20], pd[20]; //coeficientii polinomului initial si
//derivat
  float eps;  // precizia
  float x_crt; // aproximatia curenta
  float x_prec;  // aproximatia din pasul anterior
```



```
  int n, nd;  // gradul polinomului initial,respectiv derivat
  char c;
  cout << "Gradul polinomului (>=1) = ";
  cin >> n;
  cout << " Introduceti coeficientii polinomului:" << endl;
  for (int i = n; i >= 0; i--) {
    cout << "a" << i << " = ";
    cin >> p[i];
  }
  cout << " Precizia = ";
  cin >> eps;
  cout << " Valoarea de pornire = ";
  cin >> x_crt;
  deriveaza_polinom(p, n, pd, nd);
  cout << "Aproximatiile succesive pentru radacini:" << endl;
  do {
    x_prec = x_crt;
    // noua aproximatie pentru radacina:
    x_crt = x_prec - val_polinom(p, n, x_prec) /
                     val_polinom(pd, nd, x_prec);
    cout << setprecision(5) << x_crt << endl;
  } while (fabs(x_crt - x_prec) >= eps);
  cout << "Aproximatia finala este " << setprecision(5) <<
x_crt
       << endl;
  getch();
}
```

R3_5. Un număr de bare ($N \le 100$) sunt date prin lungimile lor. Se dau de asemenea P categorii de lungimi (sau standarde) între care trebuie să se încadreze lungimile pieselor ($P \le 10$).
O categorie de lungimi este precizată prin 2 limite: una minimă şi cealaltă maximă. Presupunem că aceste categorii de lungimi formează intervale disjuncte.

- O piesă i se încadrează în categoria de lungimi j dacă:
  $LMIN[j] \le L[i] \le LMAX[j]$.
  Să se calculeze şi să se afişeze:
- numărul de piese din fiecare clasă
- dimensiunea medie a pieselor din fiecare clasă de lungimi
- numărul de rebuturi şi lungimile barelor rebutate.

Rezolvare:

3_5.cpp

```
#include <stdio.h>
#include <conio.h>

#define MAX_N 100
#define MAX_P 10

int main(void) {
    int n,p;
    float bare[MAX_N],lmin[MAX_P],lmax[MAX_P]; //date de
intrare
    int categ[MAX_N]; // categoria in care intra fiecare bara
                      // vom initializa acest vector cu -1
    int nr_piese[MAX_P]; // numarul de piese din fiecare
clasa
    // lungimea totala a pieselor din fiecare clasa
    // (aceasta va fi folosita pentru determinarea mediei)
    float lungime[MAX_P] ;
    int rebuturi; // numarul de rebuturi
    int i,j;
    //citire
    printf("n="); scanf("%d",&n);
    for (i=0;i<n;i++) {
        printf("Lungimea barei %d:",i);
        scanf("%f",&bare[i]);
        categ[i]=-1; // nu cunoastem din ce categorie face
parte
    }
    printf("p="); scanf("%d",&p);
    for (i=0;i<p;i++) {
        printf("LMin[%d]=",i); scanf("%f",&lmin[i]);
        printf("LMax[%d]=",i); scanf("%f",&lmax[i]);
        nr_piese[i]=0;
        lungime[i]=0;
    }
    // calculam vectorii categ, nr_piese si lungime
    rebuturi=n; //initial toate piesele sunt rebuturi
    for (i=0;i<n;i++)
        for (j=0;j<p;j++)
            if (bare[i]>=lmin[j] && bare[i]<=lmax[j]) {
               categ[i]=j;
               nr_piese[j]++;
               lungime[j]+=bare[i];
               rebuturi--;
               break;
            }
```



```
    // afisez numarul de piese din fiecare clasa si
dimens.medie
    for (i=0;i<p;i++) {
        float medie=lungime[i]/nr_piese[i];
        printf("In categoria %d exista %d piese; dimensiunea
               medie este %.2f\n", i, nr_piese[i], medie);
    }
    // afisez numarul si lungimea pieselor rebutate
    printf("Rebuturi: %d\n",rebuturi);
    for (i=0;i<n;i++)
        if (categ[i]==-1)
           printf("Rebut: %f \n",bare[i]);

    getch();
    return 0;
}
```

R3_6. Doi vectori x şi y au n, respectiv m elemente reale distincte ($m, n \le 10$). Să se creeze un nou vector z cu conţinând elementele celor doi vectori. Elementele comune din cei doi vectori apar în z o singură dată (*reuniunea* elementelor mulţimilor reprezentate de cei doi vectori).

Rezolvare: Mulţimea reuniune conţine atât elementele mulţimii x cât şi elementele mulţimii y, dar o singură dată. De aceea trebuie să fim atenţi să nu introducem elemente duplicat în vectorul z.

Algoritmul este următorul:

1) Se introduc în vectorul z toate elementele vectorului x (acestea sunt conţinute în reuniune şi sunt distincte din ipotezele problemei).
2) Se parcurg pe rând toate elementele vectorului y şi se verifică dacă apar în vectorul x. În caz negativ sunt introduse în vectorul z.

3_6.cpp

```
#include <stdio.h>

// Citeste un vector de elemente reale de la tastatura
// Argumente: *v - pointer la adresa vectorului
//            *n - pointer la adresa variabilei ce retine
numarul de elemente al //                    sirului
void citeste_vector(float *v, int *n) {
    int i;
    printf("Numarul de elemente:");
    scanf("%d",n);
    for (i=0;i<*n;i++) {
        printf("Elementul %d:",i);
        scanf("%f",&v[i]);
    }
}
```

```
// Realizeaza reuniune multimilor reprezentate de vectorii x
si y.
// n si m reprezinta numarul de elemente al vectorilor.
// Se creaza vectorul z ce va avea k elemente.
void reuniune(float *x, int n, float *y, int m, float *z, int
*k) {
    int i,j;elemente;
    int gasit;
    elemente=0;
    // adaug elmentele vectorului x in vectorul z
    for (i=0;i<n;i++)
        z[elemente++]=x[i];
    // adaug si elementele vectorului y
    for (i=0;i<m;i++) {
        gasit=0;
        for (j=0;j<n;j++) //verific daca y[i] este in
vectorul x
            if (x[j]==y[i]) {
                gasit=1;
                break;
            }
        if (!gasit) // in caz negativ este adaugat in
vectorul z
            z[elemente++]=y[i];
    }
    *k=elemente;
}

// Afiseaza pe ecran elementele vectorului z.
void afisare(float *z, int k) {
    int i=0;
    printf("Z=[ ");
    for (i=0;i<k;i++)
        printf("%.2f ",z[i]);
    printf("]\n");
}

int main(void) {
    float x[10]; int n;
    float y[10]; int m;
    float z[20]; int k; // poate avea maxim 20 de elemente !
    citeste_vector(x,&n); // Citire vector x
    citeste_vector(y,&m); // Citire vector y
    reuniune(x,n,y,m,z,&k);
    afisare(z,k);
    return 0;
}
```


R3_7. Se citesc n ($n \le 100$) coordonate reale x, y ale unor puncte în plan şi se creează cu acestea două tablouri x şi y.

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

Rezolvare: Aria determinată de punctele i, j, k este: $S = \frac{1}{2}\begin{vmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{vmatrix}$

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

3_7.cpp

```
#include <stdio.h>

#define MAX_N 100 // numarul maxim de puncte
// calculeaza aria unui triunghi determinat de punctele
// (x1,y1) (x2,y2) (x3,y3)
float arie(float x1, float y1, float x2, float y2, float x3,
float y3) {
    return (x1*y2+x2*y3+x3*y1-x3*y2-y3*x1-y1*x2)/2;
}

int main(void) {
    int n;
    float x[MAX_N],y[MAX_N];
    int i,j,k;
    float arie_max=-1,a; // Aria maxima este initial -1
    printf("n=");scanf("%d",&n); // Citire n
    // Citire coordonate puncte
    for (i=0;i<n;i++) {
        printf("x[%d]=",i);
        scanf("%f",&x[i]);
        printf("y[%d]=",i);
        scanf("%f",&y[i]);
    }
    // Se determina punctele coliniare
    printf("Tripletele de puncte coliniare:\n");
    for (i=0;i<n;i++)
      for (j=i+1;j<n;j++)
        for (k=j+1;k<n;k++) {
          a=arie(x[i],y[i],x[j],y[j],x[k],y[k]);
          if (a==0) // arie=0 -> punctele sunt coliniare
            printf("(%.2f,%.2f) (%.2f,%.2f) (%.2f,%.2f)\n",
                   x[i],y[i],x[j],y[j],x[k],y[k]);
          else if (arie_max<a) // Determin aria maxima
                 arie_max=a;
        }
```

```
    // Se determina triunghiurile de arie maxima
    printf("Triunghiurile de arie maxima:\n");
    for (i=0;i<n;i++)
        for (j=i+1;j<n;j++)
            for (k=j+1;k<n;k++) {
                a=arie(x[i],y[i],x[j],y[j],x[k],y[k]);
                if (a==arie_max)

printf("(%.2f,%.2f)(%.2f,%.2f)(%.2f,%.2f)\n",
                        x[i],y[i],x[j],y[j],x[k],y[k]);
            }
    return 0;
}
```

R3_8. Scrieți un program C care conține:

- O funcție care stabilește dacă o valoare întreagă y se află sau nu printre cele n componente întregi ale unui vector x. Funcția are ca parametri pe y, n și x și întoarce rezultatul 1 dacă y se află în x și 0 în caz contrar.
- O funcție care citește de la tastatură o secvență de întregi terminată prin −1 (care nu aparține secvenței) și creează cu aceste valori un vector. Funcția întoarce ca rezultat numărul de componente al vectorului.
- O funcție care șterge dintr-un vector x cu n componente, valoarea din poziția p și compactează vectorul, actualizându-i și lungimea.
- O funcție `main()` care citește doi vectori a și b și creează vectorul diferență $a - b$ în locul vectorului a, folosind funcțiile definite mai sus.

Rezolvare: Pentru eliminarea elementului de pe poziția p dintr-un vector, mutăm elementul al p +1- lea în locul lui, elementul al p +2- lea în locul celui de-al p +1- lea și așa mai departe, până la ultimul element. Este importantă ordinea în care facem aceste mutări (încercați să vă gândiți ce s-ar întâmpla dacă am începe de la sfârșitul vectorului, parcurgându-l în sens invers, în loc să începem cu poziția p).

Diferența dintre doi vectori a și b se determină astfel: căutăm în b fiecare element al lui a și dacă îl găsim, îl eliminăm din a. La sfârșit, în locul vectorului a vom avea vectorul diferență.

3_8.cpp

```
#include <stdio.h>
#include <conio.h>
#define NMAX 30  // nr. maxim de elemente dn vectori
```



```
/* citire de numere de la tastatura si memorarea lor intr-un
vector: */
int creeaza_vector(int vect[]) {
  int x;  // elementul curent citit de la tastatura
  int n; // nr. de elemente care se vor citi
  n = 0;
  printf("Introduceti elementele tabloului(-1 la sfarsit):
\n");
  do
  { printf("Element nou : ");
    scanf("%d", &x);
    if (x != -1 )
      vect[n++] = x;
  } while (x != -1);
  return n;
}

/* cautarea elementului y in vectorul (de n elemente) x: */
int cauta(int y, int n, int x[]) {
  int i;
  for (i = 0; i < n; i++)
    if (x[i] == y) return 1;
  return 0;
}

/* eliminarea elementului de pe pozitia p din vectorul x: */
/* (se reactualizeaza lungimea n a vectorului) */
void elimina(int p, int x[], int& n) {
  int j;
  for (j = p; j < n - 1; j++)
    x[j] = x[j + 1];
  n--;
}

void main() {
  int a[NMAX], b[NMAX];
  int na, nb;  // lungimile vectorilor a si b
  int i;
  printf ("******* Creare vector a: *********\n");
  na = creeaza_vector(a);
  printf ("******* Creare vector b: *********\n");
  nb = creeaza_vector(b);
  i = 0;
  /* cautam in b fiecare element din a; daca il gasim, il
eliminam din a: */
    while (i < na)
    if (cauta(a[i], nb, b)) elimina(i, a, na);
    else i++;
  printf("Vectorul diferenta este: \n");
  if (na == 0) printf ("Vector vid\n");
  else
    for (i = 0; i < na; i++)
      printf(" %d ", a[i]);
  getch();
}
```

R3_9. Să se scrie un program C care conține:

- O funcție care determină poziția elementului minim dintr-un vector *x*, având *n* elemente, începând căutarea dintr-o poziție dată i. Parametrii funcției sunt: *x*, *n*, *i*, iar funcția întoarce poziția elementului minim situat între *x*[*i*] și *x*[*n*].
- O funcție având ca parametri un tablou *x* și doi întregi *p* și *q*, care interschimbă *x*[*p*] și *x*[*q*].
- O funcție `main()` care citește un întreg *n* și un vector *a* cu *n* componente și îl ordonează crescător (*sortare prin metoda selecției*), folosind funcțiile de mai sus. În acest scop se folosește proprietatea că în vectorul ordonat (*x*[0], *x*[1],..., *x*[*n*-1]) orice subvector (*x*[i], ..., *x*[*n*-1]) are elementul din vârf *x*[*i*] minim, pentru *i* = 0, ..., *n*-2. Dacă se constată că poziția *m* a elementului minim diferă de *i*, atunci se interschimbă *x*[i] cu *x*[*m*].

Rezolvare:

3_9.cpp

```
#include <stdio.h>
#include <conio.h>

/* gasirea minimului dintre elementele x[i], x[i+1], ...,
x[n-1]: */
int gaseste_min (float x[], int n, int i) {
  float min;
  int k;
  int poz_min; /* pozitia pe care se afla elementul minim */
  min  = x[i]; poz_min = i;
  for (k = i + 1; k < n ; k++)
    if (min > x[k]) {
      min = x[k];
      poz_min = k;
    }
   return poz_min;
}
/* interschimbarea elementelor x[p] si x[q]: */
void interschimba (float x[], int p, int q) {
  float aux;
  aux = x[p];
  x[p] = x[q];
  x[q] = aux;
}
```



```
void main(void) {
  float a[20];
  int n, i;
  int poz;
  printf("Numarul de elemente din vector = ");
  scanf("%d", &n);
  printf("Introduceti elementele vectorului:\n");
  for (i = 0; i < n; i++) {
    printf("a[%d] = ", i);
    scanf("%f", &a[i]);
  }
  for (i = 0; i < n - 1; i++) {
    poz = gaseste_min(a, n, i);
    interschimba(a, i, poz);
  }
  printf("Vectorul sortat:\n");
  for (i = 0; i < n; i++)
    printf("% 6.2f ", a[i]);
  getch();
}
```

R3_10. Să se ordoneze crescător un vector ce contine *n* numere întregi ($n \leq 100$) utilizând metoda contorizării inversărilor (denumită și *metoda bulelor*). Atât *n* cât și numerele vor fi citite de la tastatură.

Rezolvare:

3_10.cpp

```
#include <stdio.h>

#define MAX_N 100 // numarul maxim de elemente ale vectorului

void swap(int i, int j, int* x){
  int temp = x[i];
  x[i]=x[j];
  x[j]=temp;
}

// Sortare vector folosind METODA BULELOR
void bule(int n, int* v){
  int sortat, i;
  do { sortat=1; // initial, vectorul este presupus sortat
       for (i=0; i < n-1; i++)
          if (v[i] > v[i+1]) { // daca gasesc o inversiune
             swap(i, i+1, v);  // interschimb elementele
             sortat=0;         // marchez inversiunea
          }
    } while (!sortat); // daca nu am gasit nici o inversiune
                       // vectorul este sortat
}
```

```
int main(void) {
    int n, i;
    int v[MAX_N];
    printf("n="); scanf("%d",&n); // citire n
    // citire elemente vector
    for (i=0; i<n; i++) {
        printf("v[%d]=",i);
        scanf("%d",&v[i]);
    }
    bule(n, v);
    // Afisare
    printf("Vectorul sortat: [ ");
    for (i=0; i<n; i++)
        printf("%d ",v[i]);
    printf("]\n");
    return 0;
}
```

R3_11. De pe mediul de intrare se citesc mai multe numere reale pozitive terminate cu o valoare negativă. Să se introducă aceste valori într-un tablou v pe măsura citirii lor, astfel încât tabloul să fie ordonat crescător. Tabloul va fi afişat pe ecran la terminarea programului.

Exemplu: • Se citesc numerele: 7, 5.2, 3.4, 1, 0,9.2, -1
• Se afişează: 0, 1, 3.4, 5.2, 7, 9.2

Rezolvare:

Se creează o funcţie care adaugă un element într-un vector astfel încât vectorul să rămână sortat. Funcţia va executa următorii paşi:

1) Determină poziţia unde va fi introdus elementul nou.
2) Deplasează la dreapta vectorul cu o poziţie, începând cu poziţia determinată anterior.
3) Copiază elementul în poziţia determinată la pasul 1).

3_11.cpp

```
#include <stdio.h>

#define MAX_N 100

// Adauga un element in vectorul v, astfel incat acesta sa
ramana sortat.
// n - numarul de elemente al vectorului inainte de adaugare
// element - elementul ce trebuie adaugat
```



```
void adauga(float *v, int *n, float element) {
    int i=0,j;
    // gasesc locul unde trebuie inserat
    while (v[i]<element && i<*n)
        i++;
    // deplasez vectorul la dreapta cu o pozitie,
    //incepand cu pozitia i
    for (j=*n;j>i;j--)
        v[j]=v[j-1];
    // copiez elementul in pozitia corecta
    v[i]=element;
    (*n)++; // numarul de elemente al vectorului este
incrementat
}

void main(void) {
    int n=0,i;
    float v[MAX_N],element;
    // citesc elementele de la tastatura
    while (1) {
        printf("Introduceti un element: ", i);
        scanf("%f", &element);
        if (element < 0) break;
        adauga(v, &n, element);
    }
    // afisez vectorul sortat
    printf("Vectorul sortat: ");
    for (i=0;i<n;i++)
        printf("%.2f ",v[i]);
    printf("\n");
}
```

R3_12. Se consideră un vector x cu n componente, ordonat strict crescător, şi 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 şir (*căutare binară*). Şirul rezultat se va tipări cu câte 5 elemente pe linie.

Rezolvare: În general, căutarea binară a unei valori y într-un vector ordonat x cu n elemente se face astfel: comparăm y cu al $\frac{n}{2}$ - lea element al lui x; dacă acest element este egal cu y, căutarea s-a terminat; dacă este mai mare decât y, înseamnă că y trebuie căutat printre elementele din prima jumătate a lui x; altfel, elementul este mai mic decât y şi vom căuta printre elementele din ultima jumătate a lui x (printr-un procedeu asemănător). Definim astfel o funcţie recursivă de căutare binară, care întoarce: o valoare pozitivă, reprezentând poziţia în x a valorii găsite y (caz în care nu trebuie făcută inserare), sau o valoare negativă, dacă valoarea căutată nu a fost găsită. Înlăturând semnul – obţinem poziţia unde trebuie făcută inserarea.

3_12.cpp

```
#include <stdio.h>
#include <conio.h>
#define MAX_N 100

// cautare binara recursiva a valorii y in vectorul x
// intoarce pozitia (pozitiva) in care gaseste pe y in x
// sau o valoare negativa indicand pozitia unde trebuie
inserata
// valoarea
int CB(int i, int j, double y, double *x) {
  if(i > j) return -i;
  int 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);
}

// insereaza un element in vectorul x, astfel incat acesta sa
ramana sortat.
// n - numarul de elemente al vectorului inainte de inserare
// y - elementul ce trebuie adaugat
int insert(int n, double y, double *x) {
       int i = CB(0, n-1, y, x);
       if(i < 0){
         i=-i;
         // deplasare la dreapta cu o pozitie, incepand din i
         for (int j=n; j>i; j--)
             x[j]=x[j-1];
         // copiez elementul in pozitia corecta
         x[i]=y;
         return (n+1); // creste numarul de elemente
       }
       return n;      // altfel, ramane acelasi numar de
elemente
}

void main(void) {
       int n=1,i;
       double x[MAX_N], y;
       // citesc elementele de la tastatura
       while (1) {
         printf("Introduceti un element: ");scanf("%lf", &y);
         if (y < 0) {
           if (n == 1)
             n = 0;
           break;
         }
         n = insert(n, y, x);
       }
```



```
       // Afisez vectorul sortat
       printf("Vectorul sortat: ");
       // vectorul a fost deplasat la dreapta cu o unitate
       for (i = 1; i < n; i++) {
         if ((i-1) % 5 == 0)
             printf("\n");
         printf("\t%.2lf",x[i]);
       }
     printf("\n");
     getch();
}
```

R3_13. Să se ordoneze crescător un vector x cu n componente utilizând *sortarea prin inserție*. Vectorul se consideră partiționat în două zone:

- zonă ordonată formată inițial din primul element
- zonă neordonată formată din restul elementelor

Se extrage în mod repetat primul element din zona neordonată și se inserează în zona ordonată astfel încât relația de ordine să se păstreze. Șirul inițial și cel obținut prin ordonare vor fi afișate cu câte 10 elemente pe linie.

Rezolvare:

3_13.cpp

```
#include <iostream.h>
#include <conio.h>

void insertie(int n, int* v){
  for(int k=1; k < n; k++){
       int temp = v[k];
       int j = k;
       while(j > 0 && temp <=v[j-1]){
         v[j] = v[j-1];
         j--;
       };
       v[j] = temp;
  }
}

// Functie pentru afisarea unui vector
void print_vec (int *v, int n) {
  for (int j=0; j<n; j++){
       if(j%10==0)
         cout << endl;
       cout << "   " << v[j];
  }
  cout << endl;
}
```

```
void main(void){
  int n, j;
  cout << "n = ";
  cin >> n;
  int* v = new int[n];

  cout << "Introduceti elementele vectorului:" << endl;
  for(j=0; j<n; j++) {
      cout << " Elementul " << j << ": ";
      cin >> v[j];
      }
  print_vec (v, n);
  insertie(n, v);
  print_vec (v, n);
  getch();
}
```

R3_14. Să se sorteze un vector folosind algoritmul lui Shell (*Shell Sort*). În acest scop se sortează mai întâi elementele subşirurilor situate la distanţa d. Se reduce apoi d şi se repetă sortarea până când $d = 1$, Intervalele d se generează, de exemplu, cu relaţia $d = 3d + 1$. De exemplu, dacă $n = 1000$, se porneşte cu $d = 364$ şi se reduce cu $d = \frac{d-1}{3}$.

Rezolvare:

3_14.cpp

```
...
void Shell(int n, int* v){
  int d=1;
  while(d <= n/3)
    d = 3 * d + 1;
  int i, j, temp;
  while(d > 0){
    for(i=d; i < n; i++){
      temp = v[i];
      j = i;
      while(j>d-1 && v[j-d]>=temp){
        v[j] = v[j-d];
        j-=d;
      };
      v[j] = temp;
    };
  d = (d - 1) /  3;
  }
}
...
```


R3_15. Dându-se o valoare x şi un tablou a cu n elemente, să se separe acest tablou în două partiţii astfel încât elementele din prima partiţie să fie mai mici sau egale cu x, iar cele din a doua partiţie strict mai mari decât x.

Rezolvare:

3_15.cpp

```
//Interschimba elementele de pe pozitiile s si d din vectorul a.
void swap (int s, int d, int *a){
  int aux = a[s];
  a[s] = a[d];
  a[d] = aux;
}

//Realizeaza partitionarea.
int partitie(int n, int* a, int x){
  int s, d;
  while(1){
      s = -1; d = n;
      while(a[++s] < x && s <= d);
      while(d > 0 && a[--d] > x);
      if(s == d)
        break;
      else
        swap(s, d, a);
  }
  swap(s, d, a);
  return s;
}
```

R3_16. Să se sorteze un vector folosind algoritmul sortării rapide (*Quicksort*).

Rezolvare: Vom alege mai întâi un element din vector drept pivot (putem lua primul sau ultimul element, sau elementul median, ca în cazul de faţă, sau un element aleator etc.). Şirul de sortat este separat în două partiţii: partea stîngă – cu elemente mai mici decât pivotul, şi dreapta – cu elemente mai mari sau egale cu pivotul. În acest moment, pivotul se află pe poziţia în care ar trebui să fie în vectorul sortat şi mai rămâne să ordonăm partiţiile din stânga şi dreapta lui, aplicând acelaşi algoritm.

Procesul de partiţionare continuă (recursiv), până când partiţiile ajung de lungime 1. Funcţia de partiţionare este uşor modificată faţă de problema **R3_15**, pentru a permite partiţionarea unei porţiuni de tablou. Funcţia întoarce poziţia primului element din partiţia dreaptă. Pentru a realiza sortarea unui vector, se va apela funcţia QuickSort.

3_16.cpp

```
...
void QS(int, int, int*);
int partitie(int, int, int, int*);
void swap (int s, int d, int *a) {
  int aux = a[s];
  a[s] = a[d];
  a[d] = aux;
}

void QuickSort(int n, int* a){
  QS(0, n-1, a);
}

void QS(int s, int d, int* a){
  if(s >= d) return;
  int x = a[(s+d)/2];
  int m = partitie(s, d, x, a);
  QS(s, m-1, a);
  QS(m+1, d, a);
}

int partitie(int s, int d, int x, int* a){
  int st, dr;
  while(1){
      st = s-1; dr = d + 1;
      while(a[++st] < x && st <= dr);
      while(dr > 0 && a[--dr] > x);
      if(st == dr)
        break;
      else
        swap(st, dr, a);
  }
  swap(st, dr, a);
  return st;
}
```

R3_17. Se dau două şiruri *x* şi *y* ordonate strict crescător, având *M* şi respectiv *N* elemente. Să se construiască un şir *z* ordonat strict crescător conţinând elementele şirurilor *x* şi *y* o singură dată. (*"interclasare de şiruri"*).

Rezolvare:

3_17.cpp

```
...
void interclas(int m, int* x, int n, int* y, int& p, int* z){
  int ix=0, iy=0;
  p=0;
  while(ix < m && iy < n)
    if(x[ix] < y[iy])
      z[p++] = x[ix++];
```



```
      else
        if (x[ix] > y[iy])
            z[p++] = y[iy++];
        else {
            z[p++] = y[iy++];
            ix++;
            }
  while(ix < m)
    z[p++] = x[ix++];
  while(iy < n)
    z[p++] = y[iy++];
}
...
```

R3_18. Să se sorteze un vector prin metoda interclasării (*mergesort*).

Rezolvare: Vectorul este împărţit în două părţi cât mai egale. Se sortează acestea şi apoi cele două jumătăţi se interclasează.

Sortarea celor două jumătăţi se face în mod recursiv, în sensul că ele sunt împărţite la rândul lor în jumătăţi, până când mărimea partiţiilor devine 1.

Funcţia de interclasare scrisă în **R3_16** se modifică, în sensul că cele două tablouri de interclasat sunt plasate unul în continuarea celuilalt, iar şirul interclasat creat este plasat peste şirurile din care provine.

În funcţia de interclasare se alocă un tablou local în care se comasează şirurile de interclasat, care apoi este copiat peste acestea. Funcţia are ca parametri poziţiile de început din cele două partiţii, poziţia de sfârşit a partiţiei din dreapta şi tabloul cu elemente de sortat.

3_18.cpp

```
void merge(int, int, int, int*);
void MS(int, int, int*);
void MergeSort(int n, int* a){
  MS(0, n-1, a);
}
// Functia care se apeleaza recursiv.
void MS(int s, int d, int* a){
  if(s < d) {
    int m = (s+d)/2;
    MS(s, m, a);
    MS(m+1, d, a);
      merge(s, m+1, d, a);
    }
}
```

```
// Functia de interclasare.
// a - vectorul in care se afla partitiile de interclasat
// stars, stard, stopd - limitele partitilor in cadrul lui a
void merge(int stars, int stard, int stopd, int* a){
  int is=stars, id=stard, stops=stard-1, n=stopd-stars+1;
  int it=0;
  int* temp = new int[n];            //alocare de memorie
pentru tabloul temp
  while(is <= stops && id <= stopd) //interclasare partitii
in temp
    if(a[is] < a[id])
      temp[it++] = a[is++];
    else
      temp[it++] = a[id++];
  while(is <= stops)
    temp[it++] = a[is++];
  while(id <= stopd)
    temp[it++] = a[id++];
  for(int i=0; i < n; i++)           //copiere peste
tablourile initiale
    a[stars+i] = temp[i];
  delete [] temp;                    //eliberarea memoriei
alocate
}

// Functie pentru afisarea unui vector
void print_vec (int *v, int n){
  for (int j=0; j<n; j++){
      if(j%5==0)
        cout << endl;
      cout << "\t" << v[j];
  }
  cout << endl;
}
void main(void){
  int n, j;
  cout << "n = ";
  cin >> n;
  int* v = new int[n];

  cout << "Introduceti elementele vectorului v1:" << endl;
  for(j=0; j<n; j++) {
      cout << " Elementul " << j << ": ";
      cin >> v[j];
      }
  print_vec (v, n);
  MergeSort(n, v);
  print_vec (v, n);
  getch();
}
```


R3_19. Dându-se trei numere întregi reprezentând data unei zile (an, lună, zi), să se stabilească a câta zi din an este aceasta.

Rezolvare: Definim doi vectori ce conţin numărul de zile din fiecare lună şi numele fiecărei luni (de fapt, vom defini trei vectori, deoarece numărul de zile din luna februarie este diferit în anii normali şi cei bisecţi). Nu rămâne decât să facem o simplă adunare.

3_19.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

void main (void) {
  // Numarul de zile din fiecare luna (an normal si bisect)
  const int zile[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
                      30, 31};
  const int zile_b[] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31,
                        30, 31};
  // Numele fiecarei luni
  const char nz[12][15] = {" Ianuarie ", " Ferbruarie ", \
                           " Martie", " Aprilie ", " Mai ", \
                          " Iunie ", " Iulie ", " August ", \
                           " Septembrie ", " Octombrie ", \
                           " Noiembrie ", " Decembrie "};

  int an, luna, zi;
  int nr_zilei = 0;
  int index;          // Indexul lunii curente
  // Citim datele de intrare
  printf (" Introduceti anul: "); scanf ("%d", &an);
  printf (" Introduceti luna: "); scanf ("%d", &luna);
  printf (" Introduceti ziua: "); scanf ("%d", &zi);
  // Testam daca anul este bisect
  if ((an % 4 == 0) && ((an % 100 != 0) || (an % 400 == 0)))
{
    // Adaugam lunile anterioare
    for (index = 0; index < luna-1; index++)
      nr_zilei += zile_b[index];
    // Adaugam numarul zilei din luna curenta
    nr_zilei += zi;
  }
  else
  { // Adaugam lunile anterioare
    for (index = 0; index < luna-1; index++)
      nr_zilei += zile[index];
    // Adaugam numarul zilei din luna curenta
    nr_zilei += zi;
  }
  // Afisare rezultate
  printf (" Ati introdus ziua de %d %s, a %d-a zi din %d\n",\
        zi, nz[luna-1], nr_zilei, an);
  getch();
}
```

R3_20. *N* copii identificați prin numerele 1, 2, ..., *N* joacă următorul joc: se așază în cerc în ordinea 1, 2, .. , *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 ș.a.m.d.
Folosindu-se identificarea inițială să se stabilească ordinea de ieșire a copiilor din joc.
Exemplu: *n* = 5, p = 2, *k* = 1. Ordinea de eliminare este: 3, 5, 2, 1, 4.

Rezolvare: Pentru a ști ce copii au fost eliminați se creează vectorul copii cu următoarea semnificație:

- Dacă `copii[k] = 1`, atunci copilul *k* este în cerc
- Dacă `copii[k] = 0`, atunci copilul *k* a fost eliminat

Inițial toți copiii sunt în cerc, deci vectorul este inițializat cu 1.
Programul simulează acest joc. La fiecare pas, începând din poziția curentă (cea a ultimului copil eliminat) se numără *p* copii care nu au fost eliminați, apoi se elimină ultimul dintre aceștia.

3_20.cpp

```
#include <stdio.h>
#define MAX_N 100 // Numarul maxim de copii
int main(void) {
    int n,k,p;
    int copii[MAX_N]; // copii[k]=1 -> copilul este in cerc
                      // copii[k]=0 -> copilul a fost
eliminat
    int i,j;
    // Citesc n,k,p
    printf("n=");scanf("%d",&n);
    printf("k=");scanf("%d",&k);
    printf("p=");scanf("%d",&p);
    // Initializez vectorul copii
      for (i=0;i<n;i++)
            copii[i]=1;
      for (i=0;i<n;i++) { // i indica numarul de copii
eliminati
            for (j=0;j<p;j++) // j numara de la 1 la p
                  do { //urmatorul copil care nu a fost
eliminat
                     k++;
                     if (k==(n+1)) k=1;
                  } while (!copii[k - 1]);
            printf("Copil eliminat: %d\n", k);
            copii[k - 1]=0;
    }
    return 0;
}
```


R3_21. Se citește de la tastatură un șir de numere întregi. Se cere să se determine elementele distincte.
Exemplu: În șirul [2 2 5 4 5 1 2] elementele distincte sunt: 2 5 4 1.

***Rezolvare*:** Înainte de a tipări un element $v[i]$ pe ecran se verifică dacă acesta a mai fost afișat. Cu alte cuvinte dacă există un element $v[j]$ în vector astfel încât: $v[i]=v[j]$ și $j<i$

3_21.cpp

```
#include <stdio.h>
#define MAX_N 100 // n maxim

int main(void) {
    int n;
    int v[MAX_N];
    int i,j;
    // citesc n si vectorul
    printf("n=");scanf("%d",&n);
    for (i=0;i<n;i++) {
        printf("v[%d]=",i);
        scanf("%d",&v[i]);
    }
    for (i=0;i<n;i++) {
        int gasit=0; // verific daca a mai fost afisat
        for (j=0;j<i;j++)
            if (v[i]==v[j]) {
               gasit=1;
               break;
            }
        if (!gasit)
           printf("%d ",v[i]);
    }
    return 0;
}
```

R3_22. Dintr-un șir dat să se determine lungimea și poziția subșirului strict crescător cel mai lung, format din elemente alăturate din șirul dat. Șirul dat se tipărește în ecou cu câte 10 elemente pe linie. Subșirul de lungime maximă se afișează cu 5 elemente pe linie.

Rezolvare: Se parcurg elementele șirului începând cu al doilea element și atâta timp cât elementul curent și elementul anterior se află în relația ">=" se incrementează lungimea subșirului crescător. În momentul când această relație nu se mai respectă se verifică dacă am descoperit un subșir de lungime mai mare.

Observații:
1) Subșirul crescător de lungime maximă este inițializat cu subșirul ce conține primul element al șirului (având un singur element, îl putem considera crescător).
2) Se adaugă un ultim element la șirul dat având valoarea -MAXINT (cel mai mic număr întreg posibil), pentru a include cazul în care ultimul element al șirului este în subșirul căutat.

3_22.cpp

```
#include <stdio.h>
#include <conio.h>
#include <values.h>
#define MAX_N 100

void main(void) {
     int v[MAX_N],n;
     int inceput=0, lungime=1; // inceputul si lungime subsir
     int i,l=1;
     // citesc n si elementele sirului
     printf("n=");scanf("%d",&n);
     for (i=0;i<n;i++) {
         printf("v[%d]=",i);
         scanf("%d",&v[i]);
     }
     // afisez sirul cu ecou, cate 10 elemente pe linie
     for (i=0;i<n;i++) {
         if (i % 10 == 0)
            printf("\n");
         printf("%d ",v[i]);
     }
     // determin lungimea subsirului crescator de lungime
maxima
     v[n++]=-MAXINT;
     for (i=1;i<=n;i++)
         if (v[i]<v[i-1]) {
            if (l>lungime) {
               inceput=i-1;
               lungime=l;
            }
            l=1;
         } else
           l++;
     // afisez subsirul, cate 5 elemente pe linie
     for (i=inceput;i<inceput+lungime;i++) {
         if ( (i-inceput) % 5 == 0)
            printf("\n");
         printf("%d ",v[i]);
     }
     printf("\n");
     getch();
}
```


R3_23. Un număr întreg este reprezentat prin cifrele sale: $c[0]$, $c[1]$, ..., $c[n-1]$, ($c[0]$ fiind cifra cea mai semnificativă).
Să se calculeze câtul $q[0]$, $q[12]$, ... , $q[m-1]$ obținut prin împărțirea numărului dat prin numărul p.

Rezolvare: Pentru a obține cifrele câtului, se împarte la p fiecare rest parțial la care se adaugă câte o cifră a deîmpărțitului. Primul rest parțial la care se adaugă prima cifră a deîmpărțitului este 0.

Câtul va avea n cifre, din care primele (cele mai semnificative) pot fi 0.

Numărul efectiv de cifre ale câtului se obține prin normalizare adică prin deplasarea la stânga a cifrelor, cu scăderea lungimii până când prima cifră devine diferită de 0.

3_23.cpp

```
#include <stdio.h>
void main(void) {
  int    n, l, i, d, rest, p;
  int    c[100], q[100];
  printf("n=");
  scanf("%d", &n); // numarul de cifre al deimpartitului
  printf("introduceti cele %d cifre ale deimpartitului\n" , n);
  for (i = 0; i < n; i++) // citire deimpartit
    scanf("%d", &c[i]);
  printf("\n");
  printf("introduceti impartitorul\n");
  scanf("%d", &p);
  // ecoul datelor
  for (i = 0; i < n; i++)
    printf("%d", c[i]);
  printf(" impartit la %d = ", p);
  rest = 0;
  for (i = 0; i < n; i++) {  // impartire partiala
      d = 10 * rest + c[i];
      q[i] = d / p;
      rest = d % p;
  }
  // normalizare cit
  l = n;
  while (q[0] == 0 && l > 1)  {
    for (i = 0; i <  l - 1; i++ )
      q[i] = q[i+1]; // deplasare stinga
    l--;
  }
  // afisare cit
  for (i = 0; i < l; i++)
    printf("%d", q[i]);
  printf("\n");
}
```

R3_24. Un număr întreg lung este memorat într-un vector (fiecare cifră este reținută într-un element al vectorului). Fiind date două numere lungi, se cere să se genereze produsul acestor numere.

Rezolvare: Înmulțirea se realizează ca "pe hârtie", adică se înmulțește deînmulțitul pe rând, cu fiecare din cifrele înmulțitorului. Fiecare rezultat este deplasat spre cea mai semnificativă cifră cu un număr de poziții egal cu rangul cifrei. Rezultatul final se obține prin adunarea rezultatelor parțiale (după ce acestea au fost deplasate corespunzător).

Cifrele produsului se calculează folosind formula:

$$Cifra_{produs}[i+k-1] = Cifra_a[i] \cdot Cifra_b[k] + Transport + Cifra_{produs}[i+k-1]$$

unde contorul i parcurge cifrele primului număr, iar cu k cifrele celui de-al doilea număr.

Observații:

1) Indicele $i + k - 1$ provine din deplasare; pentru înmulțirea cu prima cifră a înmulțitorului vom avea indicele $i + 1 - 1 = i$, pentru înmulțirea cu a doua cifră $i + i$ și așa mai departe;
2) Rezultatul va avea cel mult $NrCifre_a + NrCifre_b$ cifre;
3) În cazul în care $Cifra_{produs}[i+k-1] > 9$ se reține ultima cifră, prima cifră devenind transport;
4) La început se inițializează toate cifrele rezultatului cu zero.

Exemplu: 123 · 89 = 10947

Vom înmulți mai întâi 123 cu cifra cea mai puțin semnificativă (9):

123	3	2	1	-
	9			
Transport	0	2	2	1
Rezultat vechi	0	0	0	0
Rezultat nou	9 · 3 = 7 Transport = 2	9 · 2 + 2 = 0 Transport = 2	9 · 1 + 2 = 1 Transport = 1	1

Apoi înmulțim 123 cu următoarea cifră a înmulțitorului (9):

123		3	2	1	-
		8			
Transport	0	0	2	1	1
Rezultat vechi	7	0	1	1	0
Rezultat nou	7	8 · 3 = 4 Transport = 2	8 · 2 + 2 + 1 = 9 Transport = 1	8 · 1 + 1 + 1 = 0 Transport = 1	1

3_24.cpp

```
#include <stdio.h>
#include <conio.h>
#include <mem.h>

struct NUMAR {
    int n; // numarul de cifre al numarului
    unsigned char v[100]; // cifrele vectorului,incepand cu
cmps
};

struct NUMAR citeste() {
    char ch=0,i;
    struct NUMAR a;
    unsigned char x[100];
    a.n=0;
    //citire cifre numar, terminate prin Enter(Cod ASCII 13)
    while ((ch=getche())!=13)
        x[a.n++]=ch-'0';

    // copiez cifrele din  x in vectorul a.v, inversand
cifrele
    // pentru a obtine pe prima pozitie cifra cmps
    for (i=0;i<a.n;i++)
        a.v[i]=x[a.n-i-1];
    return a;
}
void afiseaza(struct NUMAR a) {
    for (int i=a.n-1;i>=0;i--)
        printf("%d",a.v[i]);
}

struct NUMAR produs(struct NUMAR a, struct NUMAR b) {
    struct NUMAR c;
    int i,j,transport=0;
    // verific daca unul din termeni este 0
    if ( (a.n==1 && a.v[0]==0) || (b.n==1 && b.v[0]==0) ) {
        c.n=1;
        c.v[0]=0;
        return c;
    }
    c.n=a.n+b.n; // numarul de cifre al produsului
    memset(c.v,0,c.n); // initializez cifrele numarului c cu
0
    // efectuez inmultirea
    for (i=0;i<a.n;i++) {
        transport=0;
```

```
        for (j=0;j<b.n || transport>0;j++) {
            transport+=c.v[i+j];
            if (j<b.n)
                transport+=a.v[i]*b.v[j];
            c.v[i+j]=transport % 10;
            transport/=10;
        }
    }
    // verific daca exista zerouri la inceput si le elimin
    while (c.v[c.n-1]==0)
        c.n--;
    return c;
}
void main(void) {
    struct NUMAR a,b,c;
    // citesc 2 numere
    printf("\n a=");a=citeste();
    printf("\n b=");b=citeste();
    // calculez produsul
    c=produs(a,b);
    // afisez produsul
    printf("\n c=");afiseaza(c);
}
```

R3_25. Două polinoame sunt date prin gradele lor şi tablourile coeficienţilor după puterile descrescătoare ale lui *x*. Să se calculeze şi să se afişeze polinoamele cât şi rest ale împărţirii celor două polinoame.

Rezolvare: Programul va simula împărţirea manuală a două polinoame. În cursul împărţirii, polinomul deîmpărţit se va modifica; în el se păstrează pe rând polinoamele resturi parţiale.

La terminarea împărţirii tabloul polinomului deîmpărţit va conţine (în poziţiile mai puţin semnificative) polinomul rest.

Coeficientul curent din polinomul cât se obţine prin împărţirea primului coeficient al restului parţial curent la primul coeficient din împărţitor.

3_25.cpp

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

#define EPS 0.00001
void cit_pol(int n, double* a, char* mesaj){
  printf(mesaj);
  for(int i = 0; i<= n; i++)
    scanf("%lf", &a[i]);
}
```



```
void af_pol(int n, double* a, char* mesaj){
  printf(mesaj);
  for (int i = 0; i <= n; i++) {
      if (i > 0 && a[i] > 0.0)
        printf("+");
      if (fabs(a[i]) > EPS)
        printf("%5.2lf *x^ %d", a[i], n-i);
  }
  printf("\n");
}
void main(void) {
  double a[20], b[20], c[20], r[20];
  int m, n, q, i, j;
  printf("m="); scanf("%d", &m);
  cit_pol(m, a, "coeficenti deimpartit\n");
  af_pol(m, a, "polinom deimpartit\n");
  printf("n="); scanf("%d", &n);
  cit_pol(n, b, "coeficenti impartitor\n");
  af_pol(n, b, "polinom impartitor\n");
  q = m-n;
  for (i = 0; i <= q; i++) {
      c[i] = a[i]/b[0];  // coeficent cit
      for (j = 0; j <= n; j++)  // urmatorul rest partial
        a[i+j] = a[i+j] - c[i]*b[j];
  }
  for (i = 0; i < n; i++)
    r[i] = a[q+i+1];
  af_pol(q, c, "polinom cat\n");
  af_pol(n - 1, r, "polinom rest\n");
  getch();
}
```

R3_26. Să se genereze numerele prime până la o limită dată N ($N \leq 500$) folosind "*sita lui Eratostene*". Într-un vector cu N elemente de tip boolean, iniţializate la valoarea "adevărat", iau valoarea "fals" elementele din poziţiile multiple de k număr prim. Poziţiile elementelor care în final au valoarea "adevărat" sunt numere prime.

Rezolvare:

3_26.cpp

```
#include <stdio.h>
#define MAX_N 500
#define TRUE  1
#define FALSE 0

void main(void) {
    char v[MAX_N];
    int n,i,j;
```

```
    printf("n=");scanf("%d",&n); // Citesc n
    // initializez vectorul v
    for (i=0;i<n;i++)
        v[i]=TRUE;
    // Ciurul lui Eratostene
    for (i=2;i<n;i++) {
        j=i*2;
        while (j<n) {
            v[j]=FALSE;
            j+=i;
        }
    }
    // Afisare
    printf("Numerele prime mai mici decat %d:",n);
    for (i=2;i<n;i++)
        if (v[i]==TRUE)
            printf("%d ",i);
}
```

R3_27. Să se calculeze coeficienții polinomului Cebâşev 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.

Rezolvare: Fie a, b, c tablourile coeficenților polinoamelor $T_{k-2}(x)$, $T_{k-1}(x)$ şi $T_k(x)$, adică:

$$T_{k-2}(x)=\sum_{j=0}^{k-2} a_j \cdot x^j \; ; \quad T_{k-1}(x)=\sum_{j=0}^{k-1} b_j \cdot x^j \; ; \quad T_k(x)=\sum_{j=0}^{k} c_j \cdot x^j \; .$$

Dacă înlocuim aceste expresii în relația de recurență şi identificăm coeficenții în raport cu puterile lui x, se obțin relații de recurență între coeficenții a_j, b_j şi c_j de forma:

$c_0 = -a_0$

$c_j = 2 \cdot b_{j-1} - a_j$ pentru $j = 1 : k-2$

$c_{k-1} = 2 \cdot b_{k-2}$

$c_k = 2 \cdot b_{k-1}$

care se aplică pornind de la valorile inițiale: $a_0 = 1$; $b_0 = 0$; $b_1 = 1$;

3_27.cpp

```
#include <stdio.h>
#include <conio.h>
#include <mem.h>
#define MAX_N 500
```



```
void main(void) {
  int n,k,j;
  int a[20], b[20], c[20];
  printf("n="); scanf("%d", &n);
  a[0] = 1;
  b[0] = 0;
  b[1] = 1;
  // aplicare formula de recurenta
  for (k = 2; k <= n; k++) {
    c[0] = -a[0];
    for (j = 1; j <  k-1; j++)
      c[j] = 2 * b[j-1] - a[j];
    c[k-1] = 2 * b[k-2];
    c[k] = 2 * b[k-1];
    // actualizare a si b pentru iteratia urmatoare
    //a <- b
    memcpy(a, b, k * sizeof(int));
    //b <- c
    memcpy(b, c, (k + 1) * sizeof(int));
  }
  for (k = n; k >= 0; k--) {
    if (k < n && c[k] > 0)
      printf("+");
    if (c[k])
      printf("%d *x^ %d", c[k], k);
  }
  printf("\n");
  getch();
}
```

Probleme propuse

P3_1. Într-un şir x cu n componente reale, să se determine media aritmetică a elementelor pozitive situate între primul element pozitiv şi ultimul element negativ al şirului, excepând aceste elemente. Cazurile speciale vor fi clarificate prin mesaje corespunzătoare.

P3_2. Doi vectori x şi y au n, respectiv m elemente reale distincte ($m, n \le 10$). Să se creeze un nou vector z cu elementele comune ale celor doi vectori. (*intersecția* elementelor mulțimilor reprezentate de cei doi vectori).

P3_3. Se citeşte o valoare întreagă n ($0 < n \le 100$) şi n valori reale cu care se creează un vector x. Scrieţi un program C care calculează şi afişează:

- Valoarea medie $x_m = \frac{1}{n}\sum_{i=0}^{n-1} x_i$
- Abaterea medie pătratică: $x_p = \sqrt{\frac{\sum_{i=0}^{n-1}(x_i - xm)^2}{n(n-1)}}$
- Numărul de componente care depăşesc valoarea medie
- Să se creeze un vector y cu componentele din x mai mari decât valoarea medie şi să se afişeze câte 5 elemente pe o linie.

P3_4. Să se calculeze coeficienţii: C_0 , C_1 , ..., C_n , pentru n dat, ştiind că:

$\frac{C_0}{p+1} + \frac{C_1}{p} + \ \ldots\ \frac{C_p}{1} = 1$, pentru oricare $p = 0, 1, 2, ..., n$.

P3_5. Considerăm metoda de sortare din problema **R3_10** (metoda bulelor). Constatăm că acesteia i se pot aduce unele îmbunătăţiri. Astfel, se observă că în urma primei parcurgeri a listei, elementul maxim ajunge în ultima poziţie, ceea ce înseamnă că la a 2-a parcurgere sunt suficiente numai $n-2$ comparaţii, la a 3-a parcurgere $n-3$ ş.a.m.d. Putem deci reduce complexitatea algoritmului (numărul de comparaţii), făcând treceri cu amplitudine descrescătoare.
Algoritmul rezultat are o anumită asimetrie: astfel, în urma primei treceri cel mai mare element ajunge în ultima poziţie, în următoarea trecere următorul element ca ordin de mărime ajunge în penultima poziţie ş.a.m.d., în timp ce un element mai mic avansează o singură poziţie în listă în urma unei treceri. Acest dezechilibru se datoreşte faptului că trecerile prin listă se fac într-un singur sens; de sus în jos, şi ar putea fi compensat facând treceri alternative: sus-jos, jos-sus.
Scrieţi o funcţie de sortare îmbunătăţită, folosind aceste două observaţii.

P3_6. Dându-se o valoare întreagă n, să se genereze reprezentarea fracţiilor zecimale $\frac{1}{2^k}$ unde k = 1, 2, ..., n.

P3_7. Să se stabilească dacă o valoare dată y se află printre cele n componente ale unui vector dat x şi în caz afirmativ se va afişa poziţia componentei din x egală cu y, iar în caz negativ se va afişa un mesaj corespunzător. Se va utiliza o căutare secvenţială.

Indicaţie: Căutarea secvenţială a lui y în vectorul x presupune compararea între y şi $x[k]$ până la găsirea valorii căutate: $y = x[k]$ sau până la epuizarea tuturor componentelor lui x .

P3_8. Se spune că şirul x cu k elemente "intră" în şirul s cu n elemente $k \le n$, dacă există un subşir contiguu $s_i, s_{i+1}, \ldots, s_{i+k}$ cu proprietatea că elementele lui sunt identice cu elementele şirului x. Să se stabilească numărul de "intrări" ale lui x în s şi poziţiile în s ale elementelor de unde începe intrarea.

P3_9. Doi vectori a şi b au câte n componente fiecare, precizate pe mediul de intrare. Să se calculeze unghiul dintre ei exprimat în radiani.

P3_10. Să se calculeze cel mai mare divizor comun cmmdc a două numere date m şi n utilizând următoarea metodă:
- Se creează tablouri cu factorii primi ai celor două numere şi multiplicităţile lor
- Se selectează în cmmdc factorii primi comuni la puterea cea mai mică.

P3_11. Să se calculeze cel mai mare divizor comun a două numere date m şi n prin următoarea metodă:
- Pentru fiecare număr se creează un tablou cu toţi divizorii acestuia, inclusiv divizorii banali
- Se selectează apoi într-un tablou divizorii comuni şi se ia cel mai mare dintre aceştia.

P3_12. Dându-se un vector V cu n date experimentale (numere reale) să se scrie:
a) O funcţie care calculează media elementelor nenule dintr-un vector dat
b) O funcţie care înlocuieşte cu zero toate elementele dintr-un vector dat care sunt mai mari sau egale în modul faţă de o valoare dată.
c) Un program care citeşte un vector V cu maxim 200 numere reale, calculează media componentelor vectorului folosind funcţia de la punctul a), elimină prin anulare elementele din vector mai mari în modul ca dublul mediei, folosind funcţia b) şi afişează media elementelor rămase.

P3_13. a) Să se definească o funcţie întreagă care stabileşte câte elemente ale unui vector sunt mai mari decât o valoare dată.
b) Să se definească o funcţie având ca parametri: un vector, numărul de elemente ale acestuia, o valoare dată şi un al doilea vector conţinând elementele din primul care sunt mai mari decât valoarea dată; funcţia creează cel de-al doilea vector.
c) Să se scrie un program care:
- Citeşte n valori ($n \le 100$) reprezentând temperaturi şi le afişează
- Afişează numărul de temperaturi reci (<15) şi calde (≥15) şi valorile temperaturilor din fiecare categorie.

P3_14. a) Să se scrie o funcţie care numără câte componente dintr-un vector (având maxim 100 componente) sunt egale cu o valoare dată întreagă.
b) Într-un bloc există n apartamente ($n \le 100$). Fiecare apartament are cel mult 4 camere. Folosind funcţia de la punctul precedent să se stabilească numărul de apartamente cu cel mult 2 camere precum şi numărul de apartamente cu 4 camere.
Se dau: n, numărul de apartamente din bloc şi un vector x, în care $x[i]$ conţine numărul de camere al apartamentului i.

P3_15. a) Să se definească o funcție care stabilește numărul de elemente dintr-un tablou situate într-o vecinătate $(x-v,\ x+v)$ a unei valori date x.
b) Dându-se un tablou cu n elemente ($n \le 100$) și o lungime l, să se determine intervalul având această lungime în care sînt situate cele mai multe elemente din tablou. Menționăm că tabloul nu este ordonat.

P3_16. Să se definească o funcție care determină cifrele reprezentării unui număr dat într-o bază dată. Utilizând această funcție se va afișa un număr citit de pe mediul de intrare sub forma unui șir de caractere în bazele 2, 3, 10.

P3_17. Să se scrie în C:
a) O funcție care determină poziția (indicele) primului element dintr-un vector, care nu respectă ordinea crescătoare în vector. Dacă nu există un asemenea element (vectorul este ordonat), rezultatul funcției este 0.
b) O funcție care schimbă între ele valorile a două variabile.
c) Un program care citește un șir de numere reale (de lungime cunoscuta), ordonează crescător șirul folosind subprogramele de la punctele a) și b), și afișează șirul ordonat (șirul are maximum 500 de numere).

P3_18. Să se definească o funcție pentru integrarea ecuației diferențiale $y'=f(x,y)$ pe intervalul dat $[a, b]$, folosind n puncte de diviziune și cunoscând valoarea inițială y_a a soluției în punctul a, prin metoda lui Euler.
Indicație: În metoda Euler, soluția $y=y(x)$ a ecuației diferențiale se aproximează printr-un vector cu $n+1$ componente Y în care:
$Y[0]= y_a$

$Y[i+1]=Y[i]+h\cdot f(a+i\cdot h,\ Y[i])$ pentru $i = 0 \ldots n-1$, unde $h=\dfrac{b-a}{n}$

P3_19. Un polinom având toate rădăcinile reale este dat prin gradul său n și tabloul A al celor $n + 1$ coeficenți ai săi. Se dau de asemeni $n + 1$ valori ordonate $B[i]$ care separă cele n rădăcini, adică $B[i]<x[i]<B[i+1]$.
Să se localizeze cele n rădăcini ale polinomului cu precizie eps dată.
Se vor utiliza două funcții: una care localizează o rădăcină cu o precizie dată, rădăcină separată într-un interval (u, v) dat folosind metoda injumătățirii intervalului, iar cealaltă care calculează valoarea unui polinom într-un punct dat.

P3_20. Să se calculeze coeficienții polinomului obținut prin dezvoltarea produsului:

$$\prod_{j=0}^{n-1}\left(x+a_j\right)=x^n+\sum_{i=1}^{n}b_i\cdot x^{n-i}$$

Se va defini și utiliza o funcție care calculează coeficienții polinomului obținut prin înmulțirea polinomului dat prin $x + c$.

Se va modifica apoi programul pentru a calcula dezvoltarea: $\prod_{j=0}^{n-1}\left(x+a_j\right)^{m_j}$

dacă se cunosc n, a_j și m_j cu $j = 0 \ldots (n-1)$.

P3_21. Să se scrie un program care citește un întreg n și un tablou A având n elemente distincte și calculează valoarea medianei acelui tablou, adică elementul din tablou pentru care diferența între numărul valorilor mai mari și numărul valorilor mai mici decât acesta este minim.

P3_22. Să se definească funcții pentru realizarea adunării și înmulțirii unor întregi reprezentați în precizie multiplă într-o bază dată B. Un întreg în precizie multiplă este reprezentat printr-un tablou având un număr de componente egal cu lungimea numărului. Funcțiile vor realiza:
- adunarea a doi întregi de lungimi diferite
- înmulțirea a doi întregi cu n, respectiv m cifre obținând un produs cu $m + n$ cifre.

Dându-se două numere x și y reprezentate în precizie multiplă să se utilizeze funcțiile definite mai sus pentru a calcula suma și produsul lor.

P3_23. Un polinom de gradul n: $P(x)=a_0\cdot x^n+\ldots+a_n$ este cunoscut prin gradul său n și tabloul a al celor $n + 1$ coeficienți: $a_0, a_1, \ldots, a_n$.
Se cunosc de asemenea un întreg p și p valori: $z_0, z_1, \ldots, z_{p-1}$, posibile rădăcini ale polinomului.
Să se determine (și să se afișeze) care dintre cele p valori reprezintă rădăcini ale polinomului și ce multiplicitate are fiecare.
Se vor defini și folosi funcții pentru:
- calculul valorii polinomului într-un punct.
- calculul coeficienților polinomului derivat.

Indicație: Dacă z este o rădăcină cu ordinul de multiplicitate k, atunci:
$P(z)=0,\ P'(z)=0,\ P''(z)=0,\ \ldots,\ P^{(k+1)}(z)=0$
Datele (n, p, $a_0, \ldots, a_n, z_0, \ldots, z_{p-1}$) se citesc dintr-un fișier binar de reali. Există mai multe seturi de date. Numele fișierului binar este dat ca parametru al comenzii. Nu se impune nici o restricție asupra lui n și p, adică tablourile a și z sunt alocate dinamic.

P3_24. Definiți o funcție care inițializează prin citire un întreg n și n componente ale unui vector. Funcția are doi parametri: (un pointer la) numărul de componente și valorile componentelor.
Definiți o funcție care stabilește dacă o valoare dată aparține unui vector dat cu număr dat de componente. Funcția are 3 parametri: valoarea, numărul de componente și vectorul și întoarce rezultatul 1/0.
Definiti o functie `main()` care:
- citește 2 vectori
- creează vectorul intersecție al celor 2 vectori și îl afișează
- creează vectorul diferență al celor 2 vectori și îl afișează
- creează vectorul reuniune al celor 2 vectori și îl afișează

P3_25. Scrieți o funcție, care pe baza unui vector x cu n componente, crează un nou vector y din componentele din x având p apariții. Funcția întoarce numărul componentelor tabloului y.
Scrieți o funcție `main()`, care citește un vector ($n \le 100$) și pe baza lui creeaza tabloul elementelor distincte și tabloul elementelor cu două apariții și le afișează.

P3_26. Scrieţi o funcţie care calculează produsul dintre un număr fracţionar subunitar foarte lung şi un întreg b ($1 < b \le 10$). Deînmulţitul (numărul lung) este reprezentat printr-un tablou având n componente cifre (partea fracţionară). Partea fracţionară a produsului va fi de asemenea un număr lung (păstrat într-un tablou cu n cifre). Funcţia are ca parametri pe n, b, tabloul cifrelor deînmulţitului, tabloul cifrelor părţii fracţionare a produsului şi întoarce ca rezultat întreg - partea întreagă a produsului, un număr foarte lung.

P3_27. Un număr întreg lung, reprezentat într-o bază $2 \le b \le 16$ este memorat într-un vector de 20 de caractere, aliniat la dreapta şi completat la stânga cu caracterul '0'.

a) Definiţi funcţia:

```
int corect(int b, char x[]);
```

care determină dacă numărul lung este corect reprezentat în baza b.

b) Definiţi funcţia:

```
void aduna(int b, char x[], char y[], char z[]);
```

care adună 2 numere lungi x şi y reprezentate în baza b şi plasează rezultatul ca al treilea parametru

c) Definiţi funcţia:

```
int imparte(char[], int, char[]);
```

care împarte numărul lung dat ca prim parametru, cu întregul (cuprins între 2 şi 16) dat ca al doilea parametru. Câtul împărţirii este memorat ca număr lung în parametrul 3, iar restul împărţirii (o valoare întreagă între 0 şi 15) reprezintă rezultatul întors de către funcţie.

P3_28. Pentru a sorta crescător un vector de n întregi prin *metoda grupelor de monotonie*, se procedează astfel:

- Se iau pe rând elementele vectorului de sortat, care se copiază într-un nou vector (pe care îl vom numi grupă de monotonie), cât timp elementele se află în relaţia "$\le$"
- Dacă un element din vector nu respectă relaţia, în raport cu ultimul element din grupa de monotonie, se va crea cu acest element o nouă grupă de monotonie
- După repartizarea tuturor elementelor pe grupe de monotonie, se interclasează aceste grupe.

Scrieţi un program care citeşte un vector cu n componente şi îl sortează, folosind metoda descrisă mai sus.
Vom face presupunerea că pot fi cel mult 10 grupe de monotonie (în caz de depăşire, programul semnalează eroare şi se opreşte).
Alocarea de memorie se va face dinamic.

Indicaţie: Grupele de monotonie vor fi reprezentate printr-o matrice cu n linii şi 10 coloane.

Poziţia ultimului element dintr-o grupă de monotonie se păstrează într-un vector. Acest vector se iniţializează cu –1.
La interclasare se va folosi de asemenea un vector, care păstrează poziţia curentă în fiecare grupă de monotonie.
Prima grupă de monotonie se va iniţializa cu primul element din şir.

P3_29. a) Definiţi o funcţie care rezolvă un sistem de 2 ecuaţii cu 2 necunoscute.
Funcţia are 2 parametri:

- un tablou cu 6 elemente conţinând coeficienţii celor două ecuaţii,
- un tablou cu 2 elemente conţinând soluţiile sistemului.

Funcţia întoarce rezultatul:
1 - dacă sistemul este compatibil determinat,
2 - dacă sistemul este compatibil nedeterminat,
3 - dacă sistemul este incompatibil.

b) Definiţi o funcţie care citeşte n valori reale, pe care le depune într-un tablou dat ca parametru al funcţiei. Celălalt parametru este numărul n.

c) Definiţi o funcţie care calculează produsul scalar a 2 vectori a şi b. Funcţia are ca parametri pe n şi vectorii a şi b.

d) Scrieţi o funcţie `main()` care:

- Citeşte un întreg n.
- Citeşte, folosind funcţia de la punctul. b) cele n abscise şi cele n ordonate a n puncte din plan.
- Calculează şi afişează dreapta de regresie, sau dă un mesaj corespunzător, în caz că aceasta nu poate fi calculată.

Indicaţie: Dreapta de regresie $y = ax + b$ are proprietatea că pentru punctele din plan:
$E = \text{SUMA}(y[i] - a \cdot x[i] - b)^2$ este minimă.
Coeficienţii a şi b se determină prin rezolvarea sistemului:
$n \cdot a + \text{SUMA}(x[i]) \cdot b = \text{SUMA}(y[i])$
$\text{SUMA}(x[i]) \cdot a + \text{SUMA}(x[i] \cdot x[i]) \cdot b = \text{SUMA}(x[i] \cdot y[i])$
Pentru calculul sumelor, folosind funcţia produs scalar, unul din parametri va fi un vector iniţializat cu unităţi.

P3_30. a) Definiţi o funcţie având ca parametru un întreg fără semn reprezentând un an, care întoarce 1/0 după cum anul este sau nu bisect.

b) Definiţi o funcţie având ca parametru un vector cu 3 elemente întregi (an, lună, zi) care intoarce ca rezultat ultima zi din luna corespunzătoare datei. Funcţia foloseşte 2 vectori statici cu 12 elemente fiecare, corespunzând numărului de zile din fiecare lună pentru an bisect şi nebisect (sau o matrice cu 2 linii şi 12 coloane).

c) Definiţi o funcţie care primeşte ca parametru un vector cu 3 elemente reprezentând o dată şi întoarce 1/0 după cum data este sau nu corectă. O dată corectă însemnează an > 0, 0 < luna ≤ 12, 0 < zi ≤ ultima.

d) Definiţi o funcţie având ca parametri 2 vectori (cu 3 elemente) reprezentând 2 date, funcţie care întoarce rezultatul –1, 0 sau 1 după cum prima dată este înaintea celei de a doua, cele două date sunt egale, respectiv prima dată este după a doua.

e) Definiţi o funcţie `main()` care:

- Citeşte 2 date calendaristice şi le validează (dacă nu sunt corecte, afişează un mesaj şi se opreşte).
- Stabileşte succesiunea în timp a celor două date, afişând unul din mesajele: *înainte*, *egale*, *după*.

P3_31. O corespondență între două mulțimi este dată prin n perechi de valori ($x[i]$, $y[i]$). Să se stabilească dacă această corespondență definește o funcție, și în caz afirmativ să se verifice dacă funcția este bijectivă.
Se recomandă definirea și folosirea a 2 funcții: una care testează dacă dependența este funcție, și cealaltă care testează injectivitatea.

P3_32. O fracție rațională este reprezentată printr-un vector cu două componente: numărătorul și numitorul.
a) Definiți o funcție care simplifică o fracție rațională (prin cmmdc):

```
void simplif(long []);
```

tablou cu 2 elemente (numărătorul și numitorul). După simplificare rezultatele se pun în același tablou.

b) Definiți o funcție care adună două fracții raționale:

```
void aduna(long[], long[]);
```

- primul parametru conține prima fracție (și la sfârșit suma)
- al doilea parametru conține a doua fracție rațională

Rezultatul adunării este pus în locul primei fracții.
c) Definiți o funcție `main()` care:
- Citește un întreg n (cel mult 20)
- Citește n fracții raționale $\frac{a[i]}{b[i]}$
- Simplifică fracțiile.

P3_33. Se știe că polinomul definit prin relația de recurență:
$P_{k+1}(x) + 2 \cdot x \cdot P_k(x) + P_{k-1}(x) = 0$, $P_0(x) = 1$, $P_1(x) = x$
are toate rădăcinile reale și simple, cuprinse în $[a, b]$. Mai mult, rădăcinile polinomului $P_{k-1}(x)$ determină intervale de separare pentru rădăcinile lui $P_k(x)$: $x_j^{(k)} \in (x_j^{(k-1)}, x_{j+1}^{(k-1)})$ cu $x_0^{(k)} = a$, $x_{k+1}^{(k)} = b$.
Folosind această proprietate de separare, pentru a, b și n dați să se determine rădăcinile lui $P_n(x)$. Localizarea unei rădăcini se face prin bisecție (înjumătățirea intervalului).

P3_34. Fie x un vector cu n componente: $x_0, x_1, \ldots, x_{n-1}$. Calculați sumele Viète:

$s_0 = x_0 + x_1 + \ldots + x_{n-1}$
$s_1 = x_0x_1 + x_0x_2 + \ldots + x_0x_{n-1} + \ldots + x_{n-2}x_{n-1}$
. . .
$s_{n-1} = x_0x_1 \ldots x_{n-1}$

P3_35. Se dau două șiruri x și y având câte n elemente fiecare. Se dau de asemenea un număr necunoscut de valori alfa≠0 terminate cu o valoare nulă. Pentru fiecare valoare alfa să se calculeze: beta $= \sum_{i=0}^{n-1} y_i \cdot \prod_{\substack{j=0 \\ j \neq i}}^{n-1} \frac{\text{alfa} - x_j}{x_i - x_j}$ și să se afișeze alfa și beta.

P3_36. N puncte în plan sunt date prin coordonatele lor x_k ,y_k cu $k = 1 : n$, în ordinea crescătoare a absciselor. Pentru un număr neprecizat de abscisă alfa≠0 să se calculeze ordonatele beta corespunzătoare astfel:
- dacă alfa ≤ x_k atunci beta = y_k
- dacă alfa < x_0 atunci beta = y_0
- dacă alfa > x_{n-1} atunci beta = y_{n-1}
- dacă x_k < alfa < x_{k+1} atunci beta = $y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(\text{alfa} - x_k)$

Prima valoare alfa = 0 marchează sfârșitul datelor.

P3_37. Un polinom este dat prin gradul său n și tabloul coeficenților q.
Să se afle coeficienții polinomului cât b obținuți prin împărțirea polinomului dat prin $x + a$, unde a este de asemeni dat.

Indicație: Identificând obținem relațiile:

$$\sum_{k=0}^{n} q_k \cdot x^k = (x+a) \cdot \sum_{k=0}^{n-1} b_k \cdot x^k = \sum_{k=0}^{n-1} (b_k \cdot x^{k+1} + a \cdot b_k \cdot x^k) = \sum_{k=1}^{n} b_{k-1} \cdot x^k + \sum_{k=0}^{n-1} a \cdot b_k \cdot x^k$$

$$q_0 + \sum_{k=1}^{n-1} q_k \cdot x^k + q_n \cdot x^n = a \cdot b_0 + \sum_{k=1}^{n-1} (a \cdot b_k + b_{k-1}) \cdot x^k + b_{n-1} \cdot x^n$$

de unde: $b_{n-1} = q_n$; $b_{k-1} = q_k - a\, b_k$ pentru $k = n-1 : 1$.

Capitolul 4
Tablouri multidimensionale

Breviar

Alocarea statică a tablourilor:

```
tip nume[dim1] [dim2]... [dimn];
```

Exemplu:

```
int var[10][10][10]; // tablou tridimensional
```

Alocarea dinamică a unei matrice:

```
double **alocare_mat(int l, int c) {
  double **m;
  m= (double**) calloc(l, sizeof(double*));
  for (int i = 0; i < l; i++)
    m[i] = (double*) calloc(c, sizeof(double));
  return m;
}
```

Eliberarea zonei de memorie ocupate de o matrice:

```
void elibereaza_mat(double **m, int l) {
  for (int i = 0; i < l; i++)
    free(m[i]);
  free(m);
}
```

Exemplu:

```
// ........................
double **matrice;
matrice = alocare_mat(10,100);
// .... folosire matrice ....
elibereaza_mat(matrice,10);
// ........................
```


Probleme rezolvate

R4_1. a) Să se definească o funcție care calculează produsul scalar a doi vectori, adică:

$$\bar{x}^T \bar{y} = \sum_{i=0}^{n-1} x_i y_i$$

b) Să se definească o funcție care calculează produsul diadic a doi vectori:

$$\bar{\bar{x}}\bar{y}^T = \begin{bmatrix} x_0 y_0 & x_0 y_1 & \cdots & x_0 y_{n-1} \\ x_1 y_0 & x_1 y_1 & \cdots & x_1 y_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n-1} y_0 & x_{n-1} y_1 & \cdots & x_{n-1} y_{n-1} \end{bmatrix}$$

c) Să se scrie un program care citește: un număr întreg n ($n \le 10$), 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: $B = A - \frac{u \cdot v^T}{u^T \cdot v}$

Rezolvare: Vom implementa direct formulele date în funcțiile *produs_scalar* și *produs_diadic*.

4_1.cpp

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

// Intoarce produsul scalar a doi vectori de dimensiune n
int p_scalar (int *x, int *y, int n) {
  int suma = 0;
  for (int i = 0; i < n; i++)
      suma += x[i] * y[i];
  return suma;
}

// Intoarce produsul diadic a doi vectori de dimensiune n
int **p_diadic (int *x, int *y, int n) {
  int **rezultat;
  int i, j;
  rezultat = (int**) malloc (n * sizeof(int*));
  for (i = 0; i < n; i++) {
      rezultat[i] = (int*) malloc (n * sizeof(int));
    for (j = 0; j < n; j++)
        rezultat[i][j] = x[i] * y[j];
  }
  return rezultat;
}
```

```
void main (void) {
  int *x, *y, n;
  int **a;
  double **b;
  int prod_scalar;
  int **prod_diadic;
  int i, j;
  printf (" Introduceti n: "); scanf ("%d", &n);
  // alocare dinamica pentru vectori
  x = (int*) malloc (n * sizeof(int));
  y = (int*) malloc (n * sizeof(int));
  // alocare dinamica pentru matrici
  a = (int**) malloc (n * sizeof(int*));
  for (i = 0; i < n; i++)
    a[i] = (int*) malloc (n * sizeof(int));
  b = (double**) malloc (n * sizeof(double*));
  for (i = 0; i < n; i++)
    b[i] = (double*) malloc (n * sizeof(double));
  for (i = 0; i < n; i++) {
    printf (" x[%d] = ", i+1);
    scanf ("%d", &x[i]);
    printf (" y[%d] = ", i+1);
    scanf ("%d", &y[i]);
  }
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) {
      printf (" a[%d,%d] = ", i+1, j+1);
      scanf ("%d", &a[i][j]);
    }
  // Prelucrarea datelor
  prod_scalar = p_scalar (x, y, n);
  prod_diadic = p_diadic (x, y, n);
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      b[i][j] = (double)a[i][j] -   \
            (double)prod_diadic[i][j] / (double)prod_scalar;
    // Afisarea rezultatelor
  printf (" Rezultat: \n");
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
      printf (" %lf", b[i][j]);
    printf ("\n");
  }
  getch();
}
```


R4_2. Să se realizeze un program care simulează jocul "viața", 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 nici unul, va muri la începutul generației următoare (din cauza izolării)
- orice organism dintr-o celui cu patru sau mai mulți vecini în generația prezentă va muri la începutul generației următoare (din cauza supra-populației).

Rezolvare: Vom introduce de la tastatură generația de început. Apoi, pentru calculul generației următoare avem nevoie de o funcție care întoarce numărul de vecini vii ai unei celule. Astfel, ținând cont de regulile exprimate, se determină dacă celula va fi sau nu vie în generația următoare.

Funcția care calculează numărul de vecini vii trebuie să țină cont de faptul că o celulă de pe frontieră are numai 5 vecini, iar una dintr-un colț al matricii are 3 vecini.

4_2.cpp

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

// Intoarce numarul de vecini vii pentru o celula (i,j)
int vecini (int **matrice, int n, int i, int j) {
  int p, q;
  // Trebuie sa tinem cont de faptul ca
  //   celula poate fi pe frontiera...
  int start_i, start_j, stop_i, stop_j;
  int vii = 0;
  // Calculam limitele de cautare pentru celula (i,j)
  if (i == 0) start_i = 0;
  else start_i = i-1;
  if (i == n-1) stop_i = n-1;
  else stop_i = i+1;
  if (j == 0) start_j = 0;
  else start_j = j-1;
  if (j == n-1) stop_j = n-1;
  else stop_j = j+1;
```

```
  // Calculam numarul de vecini vii
  for (p = start_i; p <= stop_i; p++)
    for (q = start_j; q <= stop_j; q++)
        if (matrice[p][q]) vii++;
  return vii;
}
void main (void) {
  int **matrice, n, g;
  int i, j, pas;
  // Trebuie sa calculam generatia urmatoare
  //   folosind mereu aceeasi matrice
  int **backup;
  // Citire date de intrare
  printf (" Introduceti nr. de generatii: "); scanf ("%d", &g);
  printf (" Introduceti dimens.generatiei: "); scanf ("%d", &n);
  matrice = (int**) malloc (n * sizeof(int*));
  backup = (int**) malloc (n * sizeof(int*));
  printf (" Introduceti generatia de inceput: \n");
  for (i = 0; i < n; i++) {
    matrice[i] = (int*) malloc (n * sizeof(int));
    backup[i] =  (int*) malloc (n * sizeof(int));
    for (j = 0; j < n; j++) {
        printf (" matrice[%d,%d] = ", i+1, j+1);
        scanf ("%d", &matrice[i][j]);
        backup[i][j] = matrice[i][j];
    }
  }
  // Prelucrarea datelor si afisarea rezultatelor
  for (pas = 0; pas < g; pas++) {
      for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
            if ( (vecini(backup, n, i, j) >= 4) ||
                 (vecini(backup, n, i, j) <= 1) )
              matrice[i][j] = 0;
            if (vecini(backup, n, i, j) == 3)
              matrice[i][j] = 1;
        }
      printf (" Generatia %d: \n", pas+1);
      for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf (" %d", matrice[i][j]);
            backup[i][j] = matrice[i][j];
        }
        printf ("\n");
      }
    getch();
  }
}
```


R4_3. Să se scrie în C:

a) O funcție care verifică dacă două linii date *i* și *j* dintr-o matrice pătrată (*n* x *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 distincte, atunci se afișează numerele liniilor și coloanelor ce conțin elementele nule din matrice, folosind funcția de la punctul b)

Rezolvare: Vom implementa direct funcțiile *egale* și *nule*.

4_3.cpp

```
#include <stdio.h>
#include <conio.h>
#define TRUE  1
#define FALSE 0

// Verifica daca liniile i si j sunt egale
int egale (int mat[30][30], int i, int j, int n) {
  int k;
  for (k = 0; k < n; k++)
    if (mat[i][k] != mat[j][k])
        return FALSE;
  return TRUE;
}
// Afiseaza liniile si coloanele care contin elemente nule
void nule (int mat[30][30], int n) {
  int i, j;
  int found;
  // Cautam liniile ce contin elemente nule
  for (i = 0; i < n; i++) {
      found = FALSE;
    for (j = 0; j < n && !found; j++)
      if (!mat[i][j]) {
            found = TRUE;
            printf (" Linia %d are elemente nule.\n", i+1);
        }
  }
  // Cautam coloanele ce contin elemente nule
  for (i = 0; i < n; i++) {
      found = FALSE;
      for (j = 0; j < n && !found; j++)
        if (!mat[j][i]) {
            found = TRUE;
            printf (" Coloana %d are elemente nule.\n", i+1);
        }
  }
}
```

```
void main (void) {
  int mat[30][30], n;
  int i, j;
  int doua = 0;                 // Doua linii identice
  int toate = 1;                // Toate liniile identice
  // Citire date de intrare
  printf (" Introduceti dimens matricii: "); scanf ("%d", &n);
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) {
      printf (" mat[%d,%d] = ", i+1, j+1);
      scanf ("%d", &mat[i][j]);
    }
  // Prelucrarea datelor si afisarea rezultatelor
  for (i = 0; i < n-1 && (toate || !doua); i++)
    for (j = i+1; j < n && (toate || !doua); j++)
      if (egale(mat,i,j,n))
            doua = 1;
      else toate = 0;
  if (toate) {
    printf (" Toate liniile sunt egale! \n");
    nule (mat, n);
  } else
        if (doua)
            printf (" Doua linii sunt egale! \n");
        else printf (" Nu avem linii egale! \n");
  getch();
}
```

R4_4. a) Să se definească o funcție care calculează diferența între elementul maxim și elementul minim ale unei linii date dintr-o matrice.
b) Să se scrie un program care citește numerele naturale l și c (l, $c < 10$) o valoare reală eps și matricea A având l x c elemente, și afișează liniile din matrice pentru care diferența dintre extreme este inferioară valorii eps.

Rezolvare:

4_4.cpp

```
#include <stdio.h>
#include <conio.h>

#define TRUE  1
#define FALSE 0

// diferenta dintre  elementul maxim si cel minim de pe o
linie // a matricii;
```



```
// nc  - nr. de coloane din matrice;
// linie - numarul liniei dprelucrat
float diferenta_extreme(float a[][10], int linie, int nc) {
  float min, max;
  int i;
  min = max = a[linie][0];
  for (i = 1; i <nc; i++) {
    if (min > a[linie][i]) min = a[linie][i];
    if (max < a[linie][i]) max = a[linie][i];
  }
  return max - min;
}

// afisarea unei linii din matrice:
void afiseaza_linie(float a[][10], int linie, int nc) {
  int i;
  for (i = 0; i < nc; i++)
    printf(" %6.2f ", a[linie][i]);
  printf("\n");
}

void main(void) {
  int l, c;
  int i, j;
  int gasit; // s-au gasit linii care indeplinesc conditia
  float eps, a[10][10];
  printf ("Nr. de linii = ");
  scanf("%d", &l);
  printf ("Nr. de  coloane = ");
  scanf("%d", &c);
  printf("Introduceti elementele matricei:\n");
  for (i = 0; i < l; i++)
    for (j = 0; j < c; j++) {
      printf("a[%d][%d] = ", i, j);
      scanf("%f", &a[i][j]);
    }
  printf("eps = ");
  scanf("%f", &eps);
  gasit = FALSE;
  printf("Liniile cu diferenta extreme mai mica decat eps\n");
  for (i = 0; i < l; i++)
      if (diferenta_extreme(a, i, c) < eps) {
        afiseaza_linie(a, i, c);
        gasit = TRUE;
      }
 if (!gasit) printf("Nu exista astfel de linii\n");
  getch();
}
```

R4_5. Într-o matrice dată A cu l linii şi c coloane să se permute circular dreapta fiecare linie i cu i poziţii. Se va utiliza o funcţie care permută circular dreapta componentele unui vector cu un număr de poziţii.

Rezolvare: Vom implementa direct funcţia `permut`.

4_5.cpp

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

// Permuta elementele vectorului v de dimensiune n cu i
pozitii
int *permut (int *v, int c, int i) {
  int *aux;
  aux = (int *) malloc (c * sizeof(int));
  for (int j = 0; j < c; j++)
      aux[(j+i)%c] = v[j];
  free(v);
  return aux;
}

void main (void) {
  int **mat, l, c;
  int i, j;
  // Citire date de intrare
  printf (" Introduceti nr. de linii: "); scanf ("%d", &l);
  printf (" Introduceti nr. de coloane: "); scanf ("%d", &c);
  mat = (int**) malloc (l * sizeof(int*));
  for (i = 0; i < l; i++) {
    mat[i] = (int*) malloc (c * sizeof(int));
    for (j = 0; j < c; j++) {
      printf (" mat[%d,%d] = ", i+1, j+1);
      scanf ("%d", &mat[i][j]);
    }
  }

  // Prelucrarea datelor si afisarea rezultatelor
  for (i = 0; i < l; i++)
    mat[i] = permut (mat[i], c, i+1);
  printf (" Matricea dupa rotatii este: \n");
  for (i = 0; i < l; i++) {
    for (j = 0; j < c; j++)
      printf ("%d ", mat[i][j]);
    printf ("\n");
  }
  getch();
}
```


R4_6. Doi vectori x şi y cu câte n componente fiecare, ($n \le 20$) se află în relaţia $x \le y$ dacă $x_i \le y_i$, pentru $i: = 0 .. n - 1$.

- Să se definească o funcţie care primind ca parametri două linii i şi k ale unei matrici, stabileşte dacă acestea se află în relaţia "$\le$".
- Să se definească o funcţie care, primind ca parametri numerele a două linii dintr-o matrice calculează diferenţa lor, depunând rezultatul într-un vector.
- Să se scrie un program care citeşte un număr întreg n ($n \le 20$) şi o matrice cu n linii şi coloane. Programul va afişa pentru toate liniile care nu se găsesc în relaţia "$\le$"diferenţa acestora.

Rezolvare: Vom implementa direct funcţiile `lt` şi `line_dif`.

4_6.cpp

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

#define TRUE  1
#define FALSE 0

//Intoarce TRUE daca liniile l_1 si l_2 sunt in relatia l_1<=l_2
int mai_mic (int **mat, int n, int l_1, int l_2) {
  for (int j = 0; j < n; j++)
      if (mat[l_1][j] > mat[l_2][j])
        return FALSE;
  return TRUE;
}

// Intoarce diferenta intre liniile l_1 si l_2 ale matricii mat.
int *line_dif (int **mat, int n, int l_1, int l_2) {
  int *dif;
  dif = (int *) malloc (n * sizeof(int));
  for (int j = 0; j < n; j++)
    dif[j] = mat[l_1][j] - mat[l_2][j];
  return dif;
}

// Afiseaza un vector
void print_line (int *v, int n) {
  for (int i = 0; i < n; i++)
    printf ("%d ", v[i]);
  printf ("\n");
}
```

```
void main (void) {
  int **mat, n;
  int i, j;
  // Citire date de intrare
  printf ("Introduceti dimensiunea matricii: ");
  scanf ("%d", &n);
  mat = (int**) malloc (n * sizeof(int*));
  for (i = 0; i < n; i++) {
    mat[i] = (int*) malloc (n * sizeof(int));
    for (j = 0; j < n; j++) {
      printf (" mat[%d,%d] = ", i+1, j+1);
      scanf ("%d", &mat[i][j]);
    }
  }
   // Prelucrarea datelor si afisarea rezultatelor
  printf ("Linii care NU sunt in relatia <= au diferente: \n");
  for (i = 0; i < n; i++)
    for (j = i+1; j < n; j++)
      if (!mai_mic(mat, n, i, j)) {
        printf (" Liniile %d si %d: ", i+1, j+1);
        print_line (line_dif(mat, n, i, j), n);
      }
  getch();
}
```

R4_7. Să se definească o funcție care stabilește dacă doi vectori dați ca parametri sunt sau nu ortogonali.
Să se scrie un program care citește o matrice pătrată cu *n* linii și coloane și stabilește dacă matricea este sau nu ortogonală pe linii. În caz afirmativ calculează matricea inversă.

Rezolvare: Doi vectori x și y de dimensiune n sunt ortogonali, dacă $\sum_{i=1}^{n} x_i y_i = 0$

Se știe că pentru o matrice ortogonală, matricea inversă se obține transpunând matricea dată și împărțind fiecare coloană cu pătratul normei euclidiene a ei. O matrice este ortogonală, dacă oricare două linii diferite sunt ortogonale.

Vom folosi următoarele funcții:

- `orto` – întoarce TRUE dacă vectorii dați ca parametru sunt ortogonali, verificând relația de mai sus
- `print_matrix` – afișează o matrice
- `norm` – calculează pătratul normei euclidiene a unui vector, după formula $\sum_{i=1}^{n} x_i^2$
- `inversa` – calculează inversa. Aici ne vom folosi de faptul că liniile matricei inițiale sunt coloane pentru matricea inversă.

4_7.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#define TRUE  1
#define FALSE 0

// Aloca dinamic memorie pentru matrice
double** aloc_mat(int n){
  double** a;
  a = new double*[n];
  for (int i = 0; i < n; i++)
    a[i] = new double[n];
  return a;
}

void elib_mat(int n, double** a){
  for(int i = 0; i < n; i++)
    delete [] a[i];
  delete []a;
}

// Intoarce TRUE daca vectorii v_1 si v_2 sunt ortogonali
int orto (double *v_1, double *v_2, int n) {
  double sum = 0;
  for (int i = 0; i < n; i++)
    sum += (v_1[i] * v_2[i]);
  return (fabs(sum) < 0.001);
}

// Afiseaza o matrice de numere reale
void print_matrix (double **mat, int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << mat[i][j] << " ";
    cout << endl;
  }
}

// Intoarce patratul normei euclidiene a unui vector
double norm (double *v, int n) {
  double sum = 0;
  for (int i = 0; i < n; i++)
    sum += (v[i] * v[i]);
  return sum;
}
```

```
// Intoarce matricea inversa a matricii ortogonale mat
void inversa (double **mat, double **inv, int n) {
  double col_norm;
  int i, j;
  for(i = 0; i < n; i++){
    col_norm = norm (mat[i], n);
    for (j = 0; j < n; j++)
      mat[i][j] /= col_norm;
  };
  for(i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      inv[j][i] = mat[i][j];
}

void main (void) {
  double **mat, **inv;
  int i, j, n;
  clrscr();
  // Citire date de intrare
  cout << " Introduceti dimensiunea matricii: ";
  cin  >> n;
  mat = aloc_mat(n);
  inv = aloc_mat(n);
  for(i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      cin >> mat[i][j];
  //   Verificam daca matricea este ortogonala
  int ortogonal = TRUE;
  for (i = 0; i < n-1 && ortogonal; i++)
      for (j = i+1; j < n && ortogonal; j++)
        if (!orto(mat[i], mat[j], n))
            ortogonal = FALSE;
  // Inversa este...
  if (ortogonal) {
    inversa (mat, inv, n);
    cout << " Matricea inversa este : \n";
    print_matrix (inv, n);
  }
  else
    cout << " Matricea nu este ortogonala!\n";
  elib_mat(n, mat);
  elib_mat(n, inv);
  getch();
}
```


R4_8. Dându-se o bază $\underline{x}_1, \underline{x}_2, \ldots \underline{x}_n$, să se construiască o bază ortonormată $\underline{y}_1, \underline{y}_2, \ldots \underline{y}_n$ folosind algoritmul Gramm-Schmidt.

Rezolvare: Bazele sunt formate din n vectori de dimensiune n, deci le vom reprezenta ca matrici pătrate.

Vom construi mai întâi o bază ortogonală, pe care o vom norma (pentru a obține una ortonormată).

Într-o bază ortogonală $\underline{y}_1, \underline{y}_2, \ldots \underline{y}_n$ oricare doi vectori $\underline{y}_i$ și $\underline{y}_j$ sunt ortogonali (adică produsul lor scalar este nul), condiție de care ne vom folosi în felul următor:

- pornim cu $\underline{y}_1 = \underline{x}_1$
- pentru a determina $\underline{y}_k$, cu $k > 1$, îl scriem sub forma: $\underline{y}_k = \underline{x}_k + \sum_{j=1}^{k-1} a_{kj}\,\underline{y}_j$

(unde $k = 2, 3, \ldots, n$)

Aici a_{kj} sunt coeficienți care se calculează punând condiția de ortogonalitate $(\underline{y}_k, \underline{y}_1) = 0$;

valorile care se obțin pentru ei sunt: $a_{kj} = \dfrac{(\underline{x}_k, \underline{y}_j)}{(\underline{y}_j, \underline{y}_j)}$

Determinarea lui $\underline{y}_k$ cu formula de mai sus se face în program apelând de k ori funcția `adauga()`.

Vectorii $\underline{y}_1, \underline{y}_2, \ldots, \underline{y}_n$ obținuți astfel formează o bază ortogonală. Pentru a avea o bază ortonormată, împărțim elementele fiecărui vector la norma acestuia (adică la produsul scalar dintre vectorul respectiv și el însuși). De asemenea, pentru a nu calcula norma unui vector y_i de fiecare dată când împărțim un element al lui la aceasta, vom reține normele vectorilor $\underline{y}_i$ ($i = 1, \ldots, n$) într-un tablou.

4_8.cpp

```
#include <iostream.h>
#include <math.h>
#include <conio.h>
#define NMAX 100  // dimensiunea maxima a bazelor

/* afisarea unei baze x de dimensiune n: */
void afişează_baza(int n, float x[][NMAX])  {
  int i, j;
  for (i = 0; i < n; i++)  {
    cout << i << " :";
    cout.precision(2);
    for (j = 0; j < n; j++)
      cout << x[i][j] <<" ";
    cout << endl;
  }
}
```

```
/* produsul scalar al vectorilor x si y de dimensiune n: */
float prod_scalar(int n, float x[], float y[])  {
  int i;
  float p = 0.0;
  for (i = 0; i < n; i++)
    p += x[i] * y[i];
  return p;
}

/* efectuarea operatiei x[i] <- x[i] + a*y[i] pe elementele
vectorilor x, y: */
void adauga(int n, float x[], float y[], float a) {
  int i;
  for (i = 0; i < n; i++)
    x[i] += a * y[i];
}

void main(void) {
  int n;  //dimensiunea bazei
  int i, j, k;
  // baza initiala si cea ortonormata
  float x[NMAX][NMAX], y[NMAX][NMAX];
  // vector in care se retin produsele scalare y[k]*y[k]
  float c[NMAX];
  float a;
  cout <<" n = ";
  cin >> n;
  cout << "Introduceti baza: " << endl;
  for (i = 0; i < n; i++)    {
    cout << "Vectorul " << i << ":" << endl;
    for (j = 0; j < n; j++)   {
      cout << "x[" << i << "][" << j << "] = ";
      cin >> x[i][j];
    }
  }
  /* determinăm baza ortogonala: */
  for (k = 0; k < n; k++)  {
    for (i = 0; i < n; i++)
      y[k][i] = x[k][i];
    for (j = 0; j < k; j++)   {
      a = - prod_scalar(n, x[k], y[j]) / c[j];
      adauga(n, y[k], y[j], a);
    }
    c[k] = prod_scalar(n, y[k], y[k]);
  }
  cout <<"Baza ortonormala:" << endl;
  afişează_baza(n, y);
  /* normam baza: */
```



```
  for (k = 0; k < n; k++)
    c[k] = sqrt(c[k]);
  for (k = 0; k < n; k++)
    for (j = 0; j < n; j++)
      y[k][j] = y[k][j] / c[k];
  cout << "Baza ortonormata:" << endl;
  afişează_baza(n, y);
  getch();
}
```

R4_9. Se dă o matrice de elemente întregi având l linii şi c coloane (l<10, c<10, c>3).
Să se afişeze liniile în care există cel puţin trei elemente având minim cinci divizori nebanali. Se va defini şi utiliza o funcţie care stabileşte câţi divizori nebanali are un număr dat.

Rezolvare:

4_9.cpp

```
#include <stdio.h>
#include <conio.h>

/* calculeaxa numarul de divizori nebanali ai numarului n: */
int nr_divizori(int n) {
  int i;
  int cnt = 0;
  for (i = 2; i <= n / 2; i++)
    if (n % i == 0) cnt++;
  return cnt;
}

void main(void) {
  int L, C;  // dimensiunile matricei
  int mat[10][10];
  int i, j;
  int contor; // numarul de elemente din linia curenta care
indeplinesc conditiile
  printf("Dimensiuni: ");
  scanf("%d%d", &L, &C);
  printf("Introduceti elementele matricei: \n");
  for (i = 0; i < L; i++)
    for (j = 0; j < C; j++)   {
      printf("mat[%d][%d] = ", i, j);
      scanf("%d", &mat[i][j]);
    }
```

```
printf("Liniile cu cel putin 3 elemente\n",
        "care au minim 5 divzori nebanali:\n");
  /* parcurgem fiecare linie: */
  for (i = 0; i < L; i++)   {
      contor = 0;
      for (j = 0; j < C; j++)
        if (nr_divizori(mat[i][j]) >= 5) contor++;
      if (contor >= 3)  {
        printf("\n Linia %d :", i);
        for (j = 0; j < C; j++)
        printf("%5d ", mat[i][j]);
    }
  }
  getch();
}
```

R4_10. Să se calculeze coeficienții b_i, $i = 0 : n-1$ din dezvoltarea produsului:

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

Se dau n și coeficienții $a_0, a_1, \ldots, a_{n-1}$.

Rezolvare: Dacă se dezvoltă parantezele și se identifică coeficenții puterilor lui x se obțin relațiile:

$$\begin{aligned}
b_0 &= a_0 + a_1 + \ldots + a_{n-1}\\
b_1 &= a_0\cdot a_1 + a_0\cdot a_2 + \ldots + a_{n-2}\cdot a_{n-1}\\
&\ldots\ldots\ldots\ldots\\
b_{n-1} &= a_0\cdot a_1\cdot \ldots \cdot a_{n-1}
\end{aligned}$$

Vom calcula aceste sume stabilind în prealabil relații de recurență. În acest scop formăm matricea triunghiular superioară:

$$X = \begin{bmatrix}
a_0 & a_0+a_1 & a_0+a_1+a_2 & \cdots & a_0+a_1+a_2+\cdots+a_{n-1}\\
0 & a_0a_1 & a_0a_1+a_0a_2+a_1a_2 & \cdots & a_0a_1+a_0a_1+\cdots+a_{n-2}a_{n-1}\\
0 & 0 & a_0a_1a_2 & \cdots & \\
0 & 0 & & \cdots & \\
0 & 0 & 0 & \cdots & a_0a_1a_2\cdots a_{n-1}
\end{bmatrix}$$

în care coloana n reprezintă chiar sumele căutate.

Elementele matricei se calculează astfel:

$x_{00} = a_0$

$x_{0j} = x_{0,j-1} + a_j$ j>0

$x_{ii} = x_{i-1,i-1}\cdot a_i$ i>0

$x_{ij} = x_{i,j-1} + a_j\cdot x_{i-1,j-1}$ i>0, j>0

4_10.cpp

```
#include <stdio.h>
#define N 20

void main(void){
  double a[N], b[N];
  double x[N][N];
  int i, j, n;
  printf("n="); scanf("%d", &n);
  printf("Vectorul a\n");
  for (i=0; i < n; i++)
    scanf("%lf", &a[i]);
  x[0][0] = a[0];
  for (i=1; i < n; i++) {
    x[0][i] = x[0][i-1] + a[i];
    x[i][i] = x[i-1][i-1] * a[i];
  }
  for (i=1; i < n-1; i++)
    for (j=i+1; j < n; j++)
      x[i][j] = x[i][j-1] + a[j] * x[i-1][j-1];
  for (i=0; i < n; i++)
    b[i] = x[i][n-1];
  for (i=0; i < n; i++)
    printf("b[%d]=%lf\n", i, b[i]);
  printf("\n");
}
```

R4_11. Să se rezolve sistemul:

$$\begin{cases}
a_{00}\cdot x_0 + a_{01}\cdot x_1 + \cdots + a_{0,n-1}\cdot x_{n-1} = b_0\\
a_{11}\cdot x_1 + \cdots + a_{1,n-1}\cdot x_{n-1} = b_1\\
\ldots\\
a_{n-1,n-1}\cdot x_{n-1} = b_{n-1}
\end{cases}$$

cunoscut prin n, matricea a și vectorul b.

Rezolvare: Aceste sisteme se rezolvă prin substituție (dacă $a_{ii} \neq 0$) folosind relațiile:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} \qquad i = n-1:0$$

4_11.cpp

```
#include <stdio.h>
void STS(int n, double** a, double* b, double* x){
  for (int i = n - 1; i >= 0; i--){
      double s = 0;
      for(int j = i + 1; j < n; j++)
        s += a[i][j] * x[j];
        x[i] = (b[i] - s) / a[i][i];
  }
}

void cit_sist(int n, double** a, double* b){
  for(int i = 0; i < n; i++){
      printf("Ecuatia %d\n", i+1);
      for (int j = i; j < n; j++)
        scanf("%lf", &a[i][j]);
      printf("Termenul liber\n");
      scanf("%lf", &b[i]);
  }
}

double** alocmat(int n){
  double** a = new double* [n];
  for(int i=0; i<n; i++)
    a[i]= new double[n];
  return a;
}

void main(void){
  double **a, *b, *x;
  int i, j, n;
  printf("n="); scanf("%d", &n);
  a = alocmat(n);
  b = new double[n];
  x = new double[n];
  cit_sist(n, a, b);
  STS(n, a, b, x);
  for (i=0; i < n; i++)
    printf("x[%d]=%lf\n", i, x[i]);
  delete [] x;
  delete [] b;
  for(i=0; i<n; i++)
    delete [] a[i];
  delete [] a;
}
```


R4_12. Să se rezolve sistemul:

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + \ldots + a_{0,n-1}x_{n-1} = b_0 \\ a_{10}x_0 + a_{11}x_1 + \ldots + a_{1n-1}x_{n-1} = b_1 \\ \ldots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \ldots + a_{n-1,n-1}x_{n-1} = b_{n-1} \end{cases}$$

cunoscut prin n, matricea a şi vectorul b.

Rezolvare: Sistemul se aduce la formă superior triunghiulară în modul următor: se înmulţeşte prima linie pe rând cu $-\frac{a_{10}}{a_{00}}$, $-\frac{a_{20}}{a_{00}}$, ... $-\frac{a_{n-1,0}}{a_{00}}$, şi se adună la liniile 2, 3,..., n. În urma acestei operaţii s-a eliminat necunoscuta x_0 din ecuaţiile 2 : n.

În mod asemănător se elimină $x_1, x_2, \ldots, x_{n-1}$.

4_12.cpp

```
void Gauss(int n, double** a, double* b){
  for(int p=0; p< n-1; i++){
    for(int i=p+1; i < n; i++){
      double c = -a[i][p]/a[p][p];
      for(j=p; j < n; j++)
        a[i][j]+=c*a[p][j];
    b[i]+= c*b[p];
    }
  }
}
```

R4_13. Să se construiască un pătrat magic de dimensiune n (cu n impar), adică o matrice cu n linii şi n coloane având elemente numerele naturale 1, 2, ..., n^2 astfel încât sumele elementelor pe linii, pe coloane şi pe cele două diagonale să fie identice.

Rezolvare: Se introduc pe rând valorile 1, 2 ,... n^2 în elementele $a[i][j]$ ale pătratului magic. Următoarea poziţie (i, j) se stabileşte în funcţie de valoarea k ce urmează a fi plasată în matrice astfel: dacă $k\%n = 1$ atunci poziţia următoare va avea coloana j cu o unitate mai mică faţă de poziţia curentă, în caz contrar se cresc atât i cât şi j cu câte o unitate modulo n . Poziţia iniţială va fi: $i_0 = \frac{n+1}{2}$ şi $j_0 = n+1$.

Astfel pentru n = 5 pătratul magic este:

	1	2	3	4	5
1	11	10	4	23	17
2	18	12	6	5	24
3	25	19	13	7	1
4	2	21	20	14	8
5	9	3	22	16	15

Pentru reprezentarea grafică a pătratului magic vom utiliza caracterele: '–', pentru desenarea liniilor, '|' pentru desenarea coloanelor şi '+' pentru marcarea colţurilor pătratului.

4_13.cpp

```
#include <stdio.h>

void main(void){
  int a[19][19];
  int n, k, i, j;
  // Citire cu validarea datelor
  do {
      printf("Introduceti n < 19, impar: ");
    scanf("%d", &n);
  } while ( n % 2 == 0 || n > 20);
  // Generarea patratului magic
  i = (n + 1) / 2;
  j = n + 1;
  for (k = 1; k <= n * n; k++) {
    if (k % n == 1)
      j--;
      else {
        i++; if (i > n) i = 1;
        j++; if (j > n) j = 1;
      }
      a[i][j] = k;
  }
  // Desenarea patratului magic
  printf("+---");
  for (j = 1; j < n; j++)
    printf("----");
  printf("+\n");
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++)
      printf("|%3d", a[i][j]);
    printf("|\n");
    if (i < n) {
      printf("|---");
      for (j = 1; j < n; j++)
        printf("+---");
      printf("|\n");
      } else {
        printf("+---");
        for (j = 1; j < n; j++)
            printf("----");
        printf("+\n");
      }
  }
}
```


Probleme propuse

P4_1. Să se scrie un program care citeşte un număr întreg *n* şi o matrice pătrată *A* cu *n* linii şi coloane şi afişează numerele liniilor având în prima poziţie elementul minim şi în ultima poziţie elementul maxim din linie. Se vor afişa de asemenea numerele coloanelor având în prima poziţie elementul maxim şi în ultima elementul minim.

P4_2. Definiţi o funcţie care calculează produsul scalar a doi vectori cu elemente reale. Funcţia are ca parametri lungimea comună a celor doi vectori, pointeri la cei doi vectori şi întoarce ca rezultat produsul scalar.
Definiţi o funcţie care transpune o matrice pătrată cu elemente reale, peste ea. Funcţia are ca parametri dimensiunea matricii şi un pointer la matrice şi nu întoarce nici un rezultat.
Definiţi o funcţie care afişează la terminal o matrice pătrată cu elemente reale. Funcţia are ca parametri dimensiunea matricii şi un pointer la matrice şi nu întoarce nici un rezultat.
Definiţi o funcţie care înmulţeşte două matrici prin următoarea metodă:

- se transpune matricea înmulţitor
- se calculează elementul din poziţia (*i*, *j*) al matricei produs ca produsul scalar dintre liniile *i* şi *j* din cele două matrici.

Funcţia are ca parametri: dimensiunea comună a matricilor, pointeri la cele două matrici care se înmulţesc şi la matricea produs.
Definiţi o funcţie `main()` care:

- citeşte două matrici
- calculează şi afişează matricile şi produsul lor folosind funcţiile definite mai sus.

P4_3. Pentru o matrice dată A cu l linii şi c coloane să se afişeze toate cuplurile (i, j) reprezentând numere de linii având elementele respectiv egale. Se va defini şi utiliza o funcţie care stabileşte dacă doi vectori sunt sau nu egali.

P4_4. Se dau două matrici A şi B având n linii şi coloane fiecare. Să se stabilească dacă una dintre ele este sau nu inversa celeilalte. Se va defini şi utiliza o funcţie pentru a înmulţi două matrici.

P4_5. Un punct în şa într-o matrice este un element maxim pe coloană şi minim pe linia pe care se află sau minim pe coloană şi maxim pe linia sa. Utilizând funcţii care verifică dacă un element este minim/maxim pe linia/coloana sa, să se determine punctele în şa dintr-o matrice cu elemente distincte.

R4_6. Să se definească o funcţie care stabileşte dacă o linie specificată a unei matrici este sau nu o secvenţă ordonată crescător.
Să se definească o funcţie, care pentru o linie specificată a unei matrici determină elementul minim şi elementul maxim din acea linie.
Se citeşte o matrice pătrată A cu n linii şi coloane ($n \le 10$). Să se afişeze pentru fiecare linie, care nu reprezintă un şir de valori crescătoare: numărul liniei, elementul maxim şi elementul minim.

Indicaţie: O linie i dintr-o matrice reprezintă o secvenţă ordonată crescător dacă: $A_{i,j} \le A_{i,j+1}$, pentru $j = 1 \ldots n-1$.

P4_7. Să se scrie un program care verifică dacă o matrice pătrată cu n linii şi coloane este sau nu ortogonală, adică $A \cdot A' = I_n$. Se vor defini funcţii pentru: alocarea de memorie pentru o matrice, transpunerea unei matrice în alta, înmulţirea a două matrici, compararea unei matrici cu matricea unitate.

P4_8. Se dă o matrice a cu l linii şi c coloane de elemente reale. Să se verifice dacă matricea este ortogonală pe linii, şi în caz afirmativ să se calculeze inversa sa. Se ştie că inversa unei matrici ortogonale se obţine transpunând matricea şi împărţind fiecare coloană cu pătratul normei euclidiene a ei. Se vor defini şi utiliza obligatoriu funcţii pentru:
- verificarea dacă două linii ale unei matrici sunt ortogonale (au produsul scalar egal cu 0)
- transpunerea unei matrici
- calculul normei euclidiene a unei coloane date a unei matrici date.

R4_9. Doi vectori sunt ortogonali dacă produsul lor scalar este nul. Definiţi o funcţie care stabileşte dacă doi vectori sunt sau nu ortogonali. Funcţia are 3 parametri: un întreg reprezentând numărul de componente şi doi pointeri la cei doi vectori şi intoarce ca rezultat 1/0.
Definiţi o funcţie care stabileste dacă două linii date ale unei matrici sunt sau nu ortogonale. Funcţia are doi parametri reprezentând numerele liniilor şi intoarce ca rezultat 1/0. Matricea este alocată static ca variabilă externă a funcţiilor.
Definiţi o funcţie `main()` care:
- citeşte un întreg n şi o matrice cu n linii şi coloane
- verifică în mod eficient dacă matricea este ortogonală pe linii, adică dacă oricare două linii sunt ortogonale şi afişează un mesaj corespunzător.

R4_10. Definiţi o funcţie care înmulţeşte două matrici pătrate având aceeaşi dimensiune. Funcţia va avea ca parametri dimensiunea matricei şi trei pointeri (pentru matricele deînmulţit, înmulţitor şi rezultat).
Definiţi o funcţie care iniţializează prin citire o matrice pătrată. Funcţia va avea ca parametri dimensiunea matricei iniţializate şi un pointer.
Definiţi o funcţie care împarte o matrice printr-un scalar. Funcţia va avea 3 parametri: dimensiunea matricei, scalarul şi un pointer.

R4_11. Definiţi o funcţie `main()` care:
- citeşte o matrice
- calculează şi afişează matricea: $B = I_n + \frac{A}{1!} + \frac{A^2}{2!} + \ldots \frac{A^n}{n!}$

unde n reprezintă dimensiunea matricei.

R4_12. Se consideră o matrice pătrată având n linii şi coloane ($n \le 10$) de elemente reale. Valoarea lui n şi cele $n \cdot n$ elemente sunt citite de la tastatură.
Să se stabilească poziţiile elementelor maxime în valoare absolută din fiecare linie, memorându-se într-un vector P. Astfel dacă elementul maxim din linia i se află în coloana j, atunci $P[i] = j$ (presupunem că nu avem două elemente egale într-o linie).
Să se verifice dacă în matricea A se pot face schimbări de linii astfel încât toate elementele maxime să fie situate pe diagonală principală.
Dacă acest lucru este posibil se vor face schimbările de linii în urma cărora elementele maxime se plasează pe diagonală.
Programul va afişa matricea iniţială şi matricea cu linii schimbate.
Matricea este alocată static, ca variabilă globală.
Indicaţie: Elementele maxime pot fi aduse pe diagonală prin schimbări de linii, dacă vectorul P are toate elementele distincte.

R4_13. Definiți o funcție care stabilește pozițiile componentelor egale din doi vectori cu componente reale (astfel, dacă $x[i]=y[i]$ și $x[k]=y[k]$, pozițiile vor fi i și k). Funcția are ca parametri: lungimea comună a celor doi vectori, pointeri la cei doi vectori și un pointer la un vector cu elemente 0/1 (1 dacă componentele din acea poziție sunt egale și 0 în caz contrar).
Definiți o funcție `main()` care:
- citește un întreg n,
- citește o matrice A cu n linii și coloane,
- determină și afișează toate perechile de linii având cele mai multe componente egale.

R4_14. 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.

R4_15. O matrice pătrată 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.

R4_16. Dintr-o matrice având n linii și n coloane să se afișeze liniile care reprezintă șiruri ordonate crescător și coloanele care reprezintă șiruri ordonate descrescător.

R4_17. 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 apare în B.

R4_18. Se consideră o matrice A cu p linii și q coloane, ce conține 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.

R4_19. 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.

R4_20. Definiți o funcție care alocă dinamic memorie pentru o matrice, cu n linii și coloane, păstrată linearizat[1] pe linii într-un vector.
Definiți o funcție care transpune pe loc o matrice pătrată, cu n linii și coloane, alocată dinamic și păstrată linearizat printr-un vector.
Definiți o funcție care înmulțește două matrici pătrate, alocate dinamic și păstrate linearizat prin vectori.
Definiți o funcție care verifică dacă o matrice, alocată dinamic este sau nu egală cu matricea unitate.
Scrieți o funcție `main()` care:
- Citește un întreg n și o matrice pătrată cu n linii și coloane.
- Verifică, folosind funcțiile definite mai sus, dacă matricea este ortogonală și afișează un mesaj corespunzător. (Într-o matrice ortogonală $A \cdot A^T = I_n$).

R4_21. Definiți o funcție care alocă dinamic memorie pentru o matrice pătrată, astfel încât elementele ei să poată fi referite cu 2 indici.
Definiți o funcție care citește o matrice pătrată într-o zonă alocată dinamic.
Definiți o funcție care înmulțește două matrici pătrate, alocate dinamic, adresând elementele numai prin intermediul pointerilor.
Scrieți o funcție `main()` care:
- Citește un întreg n și o matrice pătrată cu n linii și coloane.
- Calculează matricea obținută prin ridicarea matricii date la puterea n și o afișează.

[1] O matrice păstrată linearizat este memorată într-un vector

Capitolul 5
Şiruri de caractere

Breviar

Fişierul <ctype.h> conţine prototipurile funcţiilor:

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

Fişierul <string.h> conţine prototipurile următoarelor funcţii:

Semnătură	Efect
char* strcpy(char* d, const char* s)	Copiază şirul s în d, inclusiv '\0'. Întoarce d.
char* strncpy(char* d, const char* s, int n)	Copiază n caractere din şirul s în d, completând eventual cu '\0'. Întoarce d.
char* strcat(char* d, const char* s)	Concatenează şirul s la sfârşitul lui d. Întoarce d.
char* strncat(char* d,const char* s, int n)	Concatenează cel mult n caractere din şirul s la sfârşitul lui d, completând cu '\0'. Întoarce d.

Semnătură	Efect
int strcmp(const char* d, const char* s)	Compară şirurile d şi s, întoarce: -1 dacă d<s, 0 dacă d==s şi 1 dacă d>s
int stricmp(const char* d, const char* s)	Compară şirurile d şi s (ca şi strcmp()) fără a face distincţie între litere mari şi mici.
int strncmp(const char* d, const char* s, int n)	Similar cu strcmp(), cu deosebirea că se compară cel mult n caractere.
int strincmp(const char* d, const char* s, int n)	Similar cu strncmp(), cu deosebirea că nu se face distincţie între literele mari şi mici.
char* strchr(const char* d,char c)	Caută caracterul c în şirul d; întoarce un pointer la prima apariţie a lui c în d, sau NULL.
char* strrchr(const char* d,char c)	Întoarce un pointer la ultima apariţie a lui c în d, sau NULL.
char* strstr(const char* d, const char* s)	Întoarce un pointer la prima apariţie a subşirului s în d, sau NULL.
char* strpbrk(const char* d, const char* s)	Întoarce un pointer la prima apariţie a unui caracter din subşirul s în d, sau NULL.
int strspn(const char* d, const char* s)	Întoarce lungimea prefixului din d care conţine numai caractere din s.
int strcspn(const char* d, const char* s)	Întoarce lungimea prefixului din d care conţine numai caractere ce nu apar în s.
int strlen(const char* s)	Întoarce lungimea lui s ('\0' nu se numără).
char* strlwr(char* s)	Converteşte literele mari în litere mici în s.
char* strupr(char* s)	Converteşte literele mici în litere mari în s.
void* memcpy(void* d,const void* s,int n)	Copiază n octeţi din s în d, întoarce d.

Semnătură	Efect
`void* memmove(void* d,const void* s,int n)`	Ca şi memcpy, folosită daca s şi d se întrepătrund.
`void* memset(void* d,const int c,int n)`	Copiază caracterul c în primele n poziţii din d.
`int memcmp(const void* d,const void* s,int n)`	Compară zonele adresate de s şi d.
`char *strdup (const char *d)`	Copiază şirul d într-o nouă locaţie de memorie şi întoarce un pointer către acea locaţie.
`char* strtok(const char* d, const char* s)`	Caută în d subşirurile delimitate de caracterele din s. Primul apel întoarce un pointer la primul subşir din d care nu conţine caractere din s următoarele apeluri se fac cu primul argument NULL, întorcându-se de fiecare dată un pointer la următorul subşir din d ce nu conţine caractere din s. În momentul în care nu mai există subşiruri, funcţia întoarce NULL.

Probleme rezolvate

R5_1. Se citeşte de la tastatură un text format din linii şi terminat prin sfârşit de fişier (CTRL-Z). Stabiliţi numărul de litere, numărul de separatori, numărul de cuvinte, numărul de linii, numărul de propoziţii şi numărul de aliniate. Cuvintele sunt separate între ele prin spaţii-albe, punct, virgulă, două puncte, punct virgulă şi cratimă. La sfârşitul fiecărei propoziţii avem punct, iar un aliniat se termină cu punct urmat de sfârşit de linie.

Rezolvare: Caracterele sunt citite pe rând în aceeaşi variabilă *c* şi sunt clasificate astfel: literă – detectată cu funcţia `isapha()`, separator – folosim funcţia `isspace()` la care mai adăugăm teste pentru separatorii aleşi de noi, linie – la apariţia sfârşitului de linie, propoziţie – la apariţia punctului, aliniat – la apariţia sfârşitului de linie, dacă caracterul precedent a fost punct. Aceasta ne impune memorarea caracterului precedent, actualizat înaintea citirii unui nou caracter. Vom avea 6 contoare, incrementate corespunzător ce ţin evidenţa situaţiilor prezentate.

În ceea ce priveşte determinarea numărului de cuvinte, situaţia este puţin mai complicată. Vom folosi o variabilă de stare – `in_cuv` care ia valoarea TRUE sau FALSE după cum ne aflăm în interiorul sau în afara unui cuvânt. Iniţial `incuv=FALSE` (ne aflăm în afara unui cuvânt). La întâlnirea unei litere starea devine `incuv=TRUE`, iar la întâlnirea unui separator, dacă ne aflăm în cuvânt (`incuv=TRUE`) îl vom părăsi (`incuv=FALSE`) şi vom contoriza cuvântul.

5_1.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>

#define CTRL_Z     -1
#define LINIE_NOUA      10 // Cod ascii pentru linie noua

#define TRUE  1
#define FALSE 0

void main (void) {
  char c, c_precedent = '\0';
  int in_cuvant = FALSE;
  int n_lit = 0, n_sep = 0, n_cuv = 0;
  int n_lin = 0, n_prop = 0, n_alin = 0;

  while(1) {
      c = getchar();
      if (c == CTRL_Z)
        break; // Detectez sfarsitul textului
      if(isalpha(c)){
        n_lit++;
        in_cuvant = TRUE;
      }
      if(isspace(c) || c=='.' || c==',' || c==';' ||
         c==':' || c=='-'){
        n_sep++;
        if(in_cuvant){
            in_cuvant = FALSE;
            n_cuv++;
        }
```

```
        if(c == '.')
            n_prop++;
        if(c == LINIE_NOUA){
            n_lin++;
            if(c_precedent == '.')
              n_alin++;
        }
        c_precedent = c;
      }
  }
  printf("numar de litere = %d\n", n_lit);
  printf("numar de separatori = %d\n", n_sep);
  printf("numar de linii = %d\n", n_lin);
  printf("numar de cuvinte = %d\n", n_cuv);
  printf("numar de propozitii = %d\n", n_prop);
  printf("numar de aliniate = %d\n", n_alin);
  getch();
}
```

R5_2. Un şir de caractere reprezintă un număr real cu parte întreagă şi parte fracţionară. Primul caracter poate fi semnul numărului. Să se scrie o funcţie pentru conversia şirului în număr real. Nu se vor folosi funcţiile standard de conversie existente. Eventualele erori de reprezentare sunt sesizate prin valoarea -1 întoarsă de un parametru de eroare. În caz că nu sunt erori parametrul de eroare întoarce valoarea 0. Funcţia `main()` citeşte şiruri de caractere şi afişează conversia în real cu un număr specificat de zecimale. În caz de eroare se va afişa un mesaj corespunzător.

Rezolvare: Se testează mai întâi dacă numărul are semn, se memorează şi se trece peste el. Se converteşte apoi partea întreagă a numărului, până la întâlnirea punctului sau a sfârşitului de şir. Urmează apoi conversia părţii fracţionare (dacă există) şi formarea numărului, din partea întreagă, partea fracţionară şi semn.

5_2.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

// Functie ce converteste un sir de caractere in numar real
double atod(char *s, int &er){
  int semn = 1;
  double pint = 0; // Partea intreaga
```



```
  double pfrac= 0; // Partea fractionara
  double w = 1;    // Ponderea cifrei partii fractionare
  // Evaluare semn
  if(*s == '-') semn = -1;
  if(*s == '-' || *s == '+') s++;
  // Evaluare parte intreaga
  for( ; *s!='.' && *s!='\0'; s++)
      if ( isdigit(*s) )
        pint = 10*pint + *s - '0';
      else {
            er = -1;
            return 0;
      }
  // Evaluare parte fractionara
  if( *s == '.' ) {
      s++;                  // Se ignora punctul
      while( *s!='\0' ) {
        if ( isdigit(*s) ) {
            w *= 0.1;   // Actualizare pondere cifra fractionara
            pfrac += (*s - '0') * w;
        } else {
              er = -1;
              return 0;
        }
        s++; // Avansez la pozitia urmatoare din sir
      }
  }

  er = 0; // Conversie realizata cu succes
  return (pint + pfrac) * semn;
}
void main (void) {
  char s[20];
  int er;
  double nr;
  printf (" Introduceti numarul: ");
  while (gets(s)) {
      nr = atod(s, er);
      printf("%s ", s);
      if(er == 0)
        printf("= %8.4lf\n", nr);
      else
        printf( "eroare\n");
      printf (" Introduceti numarul: ");
  }
  getch();
}
```

R5_3. Un şir de caractere, introdus de la tastatură reprezintă definiţiile pe orizontală dintr-un careu de cuvinte încrucişate. După fiecare cuvânt (inclusiv ultimul) se pune '*'. Numărați cuvintele pe verticală şi afişaţi liniile verticale.

Rezolvare: Şirul de caractere reprezintă matricea (careul) vectorizată pe linii şi completată cu coloana $(n + 1)$ de asteriscuri.

Din observaţia că lungimea şirului de caractere $ls = n \cdot (n + 1)$ se deduce, prin rezolvarea ecuaţiei de gradul 2, valoarea lui n (dimensiunea careului).

Un cuvânt pe verticală are în şirul de caractere literele situate între ele la distanţa $(n + 1)$.

5_3.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <math.h>

void main (void) {
 char s[240];//sir de caractere ce contine careul de maxim 15*15
 int lc = 0; //lungimea unui cuvant
 int nc = 0; //numarul de cuvinte
 int n;      //dimensiunea careului;
 int ls;     //lungimea sirului de caractere
 printf (" Introduceti careul... \n");
 gets(s);
 ls = strlen(s);
 n = (sqrt(4*ls+1)-1)/2;
 int i, j, k;
 for (j = 0; j < n + 1; j++){
   for (i = 0; i < n; i++){
     k = j + i * (n + 1);
        if (s[k] != '*')
            lc++;
        else {
            if (lc > 1)
              nc++;
            lc = 0;
        }
        if (j != n)
            printf("%c", s[k]);
   }
   printf("\n");
 }
 printf(" Sunt %d cuvinte pe verticala\n", nc);
 getch();
}
```


R5_4. Scrieţi o funcţie care determină dacă un cuvânt (şir de caractere ce nu conţine separatori) reprezintă un palindrom. Scrieţi o funcţie `main()` care citeşte de la tastatură un text format din linii, alcătuite din cuvinte. Cuvintele sunt separate printr-unul din caracterele din şirul `" . , : ; - \t \n "`. La sfârşit se va afişa numărul cuvintelor palindroame din text şi care sunt acestea.

Rezolvare: Funcţia `palindrom()` converteşte mai întâi caracterele din şir la litere mici – folosind funcţia `strlwr()` şi apoi compară caracterele simetrice faţă de mijlocul şirului. La întâlnirea unei neconcordanţe întoarce `FALSE`.

În funcţia `main()` vom citi pe rând câte o linie (cu `gets()`), îi vom separa cuvintele, folosind funcţia `strtok()` şi vom testa fiecare cuvânt cu funcţia `este_palindrom()`. Palindroamele găsite sunt contorizate şi introduse într-un tablou de şiruri de caractere, de unde vor fi afişate la sfârşit.

5_4.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

#define NP 20
#define TRUE  1
#define FALSE 0

int este_palindrom(char* s) {
  s = strlwr(s); // Conversie in litere mici
  // Comparare caractere simetrice
  for(int i=0, j=strlen(s)-1; i < j; i++, j--)
      if( s[i] != s[j] ) return FALSE;
  return TRUE;
}

void main (void) {
  char sep[]=" .,;:-\t\n";
  char linie[80], * tpal[NP], *p, *s;
  int np = 0; //numarul de palindroame
  printf (" Introduceti o linie: ");
  while(gets(linie)) {
      for(p = strtok(linie, sep); p; p = strtok(0, sep))
        if(este_palindrom(p))
            tpal[np++] = strdup(p);
      printf (" Introduceti o linie: ");
      }
```

```
  // Afisare rezultate
  printf("S-au gasit %d palindroame\n", np);
  for(int j=0; j < np; j++)
    printf("%s\n", tpal[j]);
  getch();
}
```

R5_5. a) Să se scrie o funcție care șterge dintr-un șir de caractere un subșir specificat prin poziție și lungime. Funcția întoarce un pointer la șirul modificat.
b) Scrieți o funcție care inserează într-un șir, începând cu o poziție dată, un alt șir. Funcția întoarce un pointer la șirul nou creat, alocat dinamic.
c) Scrieți o funcție `main()` care citește două cuvinte și înlocuiește într-un text introdus de la tastatură, toate aparițiile primului cuvânt prin cel de-al doilea.

Rezolvare: O linie este citită (cu `gets()`) într-o zonă alocată static (`linie`), de unde este transferată într-o zonă alocată dinamic (pentru a putea fi ștearsă în cazul inserării unui șir). Înlocuirea cuvântului 1 prin cuvântul 2 presupune: găsirea cuvântului 1 în linie (folosind `strstr()`), ștergerea cuvântului găsit (cu `sterge()`) și inserarea în poziția veche a cuvântului 2 (cu `insert()`). Operația se repetă cât timp cuvântul există în linie.

Întrucât nu păstrăm tot textul, ci numai o linie din el, la terminarea înlocuirilor într-o linie, aceasta va fi afișată (cu `puts()`).

5_5.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

// Functia de stergere
char* sterge(char* s, int poz, int lg){
  strcpy(s+poz, s+poz+lg);
  return s;
}

// Funtia de inserare:
// se aloca spatiu pentru nou sir de caractere,
// se copiaza prima parte din sirul sursa, se concateneaza sirul
// ce trebuie inserat, iar apoi se concateneaza ultima parte din
// sirul sursa.
```



```
char* insert(char* s, int poz, char* si){
  char* p = (char*) malloc(strlen(s)+strlen(si)+1);
  strncpy(p, s, poz); p[poz]='\0'; // strncpy nu pune '\0' !
  strcat(p, si);
  strcat(p, s+poz);
  free(s);
  return p;
}

void main (void) {
  char linie[80], cuv1[10], cuv2[10];
  int l_cuv1;
  char *p, *l_dinamic;
  puts("Primul cuvant:"); gets(cuv1);
  l_cuv1=strlen(cuv1);
  puts("Al doilea cuvant cuvant:"); gets(cuv2);
  printf (" Introduceti linia: ");
  while(gets(linie)) {
      // Linia este transferata intr-o zona alocata dinamic
      l_dinamic = strdup(linie);
      p = strstr(l_dinamic, cuv1);
      while(p) {
        // Transformare pointer in indice
        int pozitie = p - l_dinamic;
        sterge(l_dinamic, pozitie, l_cuv1);
        l_dinamic = insert(l_dinamic, pozitie, cuv2);
        p = strstr(l_dinamic+pozitie, cuv1);
      }
      puts(l_dinamic);
      free(l_dinamic); // Eliberez zona de memorie
      printf (" Introduceti linia: ");
  }
  getch();
}
```

R5_6. Un text citit de la intrarea standard conține cuvinte separate prin spații albe (, . ; : \n). O anagramă a unui cuvânt se obține amestecându-i literele (de exemplu: briliant și labirint). Afișați la ieșirea standard toate anagramele, precedate de numărul lor.

Rezolvare: Se separă mai întâi cuvintele din text (cu `strtok()`). Se mențin două tablouri cu șiruri de caractere: unul pentru cuvintele citite și altul care conține cuvintele având literele sortate. Se ordonează cele două tablouri după cuvintele din `c_sortate`. Anagramele vor fi pe poziții consecutive în vectorul `cuvinte`.

5_6.cpp

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

#define TRUE  1
#define FALSE 0

// Sorteaza literele din sirul v folosind sortarea prin insertie
// Poate fi folosita orice metoda de sortare (R3_9,R3_10, R3_13)
void sort (char* v) {
  for (int k=1; k < strlen(v); k++) {
      char temp = v[k];
      int j = k;
      while (j > 0 && temp <= v[j-1]) {
        v[j] = v[j-1];
        j--;
      }
      v[j] = temp;
  }
}
// Inverseaza s1 cu s2 (parametrii sunt transmisi prin
adresa!)
inline void swap(char* &s1, char* &s2) {
      char *tmp;
      tmp = s1;
      s1 = s2;
      s2 = tmp;
}
void main(void) {
  char sep[]=" .,;:-\t\n"; // separatori de cuvinte
  char linie[80];          // o linie citita din text
  char *cuvinte[100];      // vector cu cuvintele introduse
  // vector cu cuvintele avand literele sortate
  char *c_sortate[100];
  int i,j,k;
  int flag;
  int nc = 0;       // numarul de cuvinte distincte

  // citesc textul si formez vectorii cuvinte si c_sortate
  printf (" Introduceti o linie: ");
  while (gets(linie))
      for( char *p = strtok(linie, sep); p; p=strtok(0, sep) ) {
        // Verific sa nu existe acelasi cuvant de 2 ori
        for (j=0; strcmp(cuvinte[j],p)!=0 && j<nc; j++) ;
        // Adaug cuvantul in lista de cuvinte
        if (j == nc) {
            cuvinte[nc]=strdup(p);
            c_sortate[nc]=strlwr(strdup(p));
            sort(c_sortate[nc]);
            nc++;
        }
        printf (" Introduceti o linie: ");
      }
```



```
  // Sortez vectorii cuvinte si c_sortate dupa vectorul
c_sortate
  flag = FALSE;
  while (!flag) {
      flag = TRUE;
      for (i=0;i<nc-1;i++)
        if (strcmp(c_sortate[i], c_sortate[i+1]) > 0) {
            swap(c_sortate[i],c_sortate[i+1]);
            swap(cuvinte[i],cuvinte[i+1]);
            flag=FALSE;
        }
  }
  // Afisez grupele de anagrame
  for (i=0;i<nc;) {
    // Determin cate anagrame am de afisat
    for (j=i+1;j<nc && strcmp(c_sortate[i],c_sortate[j])==0;j++)
                                  ;
    printf("%d Anagrame:\n",j-i);
    for (k=i;k<j;k++)
      printf("\t%s\n",cuvinte[k]);
    i=j;
  }
  getch();
}
```

R5_7. a) Definiţi o funcţie care determină poziţia primului caracter, dintr-un şir de caractere `txt`, care apare în alt şir de caractere `sep`. Căutarea în primul şir se face începând dintr-o poziţie specificată *p*. Dacă nici unul din caracterele din `txt` nu apare în `sep`, funcţia întoarce –1.
b) Definiţi o funcţie care determină poziţia primului caracter, dintr-un şir de caractere `txt`, care nu apare în alt şir de caractere `sep`. Căutarea în primul şir se face începând dintr-o poziţie specificată *p*. Dacă toate caracterele din `txt` apar în `sep`, funcţia întoarce –1.
c) Definiţi o funcţie `main()` care citeşte un întreg `n` şi un text format din mai multe linii. O linie conţine cuvinte, despărţite între ele prin separatorii " `,.;:-\n\t`". După citirea unei linii, aceasta este afişată trunchiind la *n* cuvintele care depăşesc această lungime (se vor folosi funcţiile definite la punctele anterioare).

Rezolvare: Prima funcţie compară câte un caracter din text, începând din poziţia specificată, pe rând, cu caracterele separatoare, până când găseşte o egalitate, întorcând poziţia caracterului din text. Dacă nici unul dintre separatori nu este egal cu caracterul din text, se încearcă următorul caracter din text. Dacă nu se găseşte nici un separator în text, se întoarce rezultatul –1.

A doua funcție întoarce poziția caracterului din text, dacă acesta nu este egal cu nici unul dintre separatori. În caz de egalitate cu un separator se încearcă următorul caracter din text.

Textul se citește progresiv linie cu linie. După citirea unei linii, se separă cuvintele (folosind cele două funcții de la punctele anterioare), se compară lungimea cuvântului cu lungimea maximă admisă și dacă este cazul se trunchiază. După ce au fost prelucrate toate cuvintele se afișează linia.

5_7.cpp

```
#include <stdio.h>
#include <string.h>

#define TRUE  1
#define FALSE 0

int find_first_of(char *txt, char *sep, int p) {
  for (char *i = txt+p; *i!='\0'; i++)
      for (char *j = sep; *j!='\0'; j++)
        if (*i == *j)
            return i-txt;
  return -1;
}

int find_first_not_of(char *txt, char *sep, int p) {
  // TRUE daca caracterul curent din txt se gaseste in sep
  int flag;
  for (char *i = txt+p; *i!='\0'; i++) {
      flag=FALSE;
      for (char *j = sep; *j!='\0'; j++)
        if (*i == *j)
            flag=TRUE;
      if (!flag) return i-txt;
  }
  return -1;
}

void main (void) {
  char sep[]=" .,:;-\n\t";
  char linie[80];
  int ic, sc, lc;  // Inceput, sfarsit si lungime cuvant
  int ll;          // Lungime linie
  int n;           // Lungimea la care se face trunchierea
  printf("N=");scanf("%d", &n);

  while(gets(linie)) {
    // Adaugam un separator la sfarsit de linie
    strcat(linie," ");
    ll = strlen(linie);
    sc = 0;
    while(1){
      ic = find_first_not_of(linie, sep, sc); //inceput cuvant
      sc = find_first_of(linie, sep, ic);     //sfarsit cuvant
      lc = sc - ic + 1;                       //lungime cuvant
      if(lc > n) {                            //scurtare cuvant
            strcpy(linie+ic+n, linie+sc);
        sc -= (lc-n);
        ll -= (lc-n);
       }
        if(sc >= ll || ic==-1 || sc==-1) break;
    }
    puts(linie);
  }
}
```


Probleme propuse

P5_1. Se citește un întreg n și n linii reprezentând orizontala unui careu de cuvinte încrucișate. Punctele negre sunt reprezentate prin '*'. Să se afișeze n linii reprezentând verticala careului. Să se numere cuvintele din careu, pe orizontală și verticală. Un cuvânt are cel puțin 2 litere

P5_2. Un șir de caractere reprezintă un întreg fără semn în baza 16. Să se scrie o funcție care întoarce valoarea numărului (în baza 10). Un parametru de eroare este setat la –1 în cazul detectării unei erori de reprezentare, și la 0 în caz contrar.

P5_3. Să se scrie o funcție, având ca parametru un număr real în precizie simplă, care obține reprezentarea acelui număr cu parte intreagă și parte fracționară (fără exponent) sub forma unui șir de caractere, transmis ca parametru. Nu se vor folosi eventualele funcții de conversie disponibile.

P5_4. Să se scrie o funcție având ca parametru o valoare `unsigned long`, care calculează reprezentarea numărului în baza 16 și depune rezultatul într-un șir de caractere dat ca parametru al funcției.

P5_5. Un text citit de la intrarea standard reprezintă un program C format din funcția `main()`. Să se determine nivelul maxim de adâncime pentru instrucțiunile compuse.
Indicație: Se folosește un contor, crescut la fiecare nivel (și scăzut după fiecare nivel). Valoarea maximă a contorului dă nivelul de adâncime.

P5_6. Pe baza unui text introdus de la tastatură și terminat prin Ctrl-Z să se creeze un tablou cu cuvintele distincte din text, însoțite de numărul lor de apariții. Cuvintele sunt separate prin spații albe `" .,;:'\n\t()-"`.

P5_7. Un text, citit de la tastatură, este format din cuvinte, separate prin spații albe. Textul se termină prin '.' .Să se scrie la ieșire cuvântul cel mai lung din acea linie (dacă există mai multe cuvinte de aceeași lungime maximă se va considera numai primul dintre ele).

P5_8. Să se scrie o funcție având ca parametri două șiruri de caractere, care șterge din primul șir toate aparițiile celui de-al doilea șir. Funcția întoarce un pointer la șirul modificat.
Scrieți o funcție `main()` care citește o linie care conține mai multe cuvinte și mai multe linii reprezentând un text. Acest text este memorat folosind un tablou de pointeri la șiruri de caractere alocate dinamic. Folosind funcția de mai sus se vor șterge din text toate cuvintele introduse la început și se va afișa textul rezultat.

P5_9. Prima linie, introdusă de la tastatură, conține un cuvânt, iar următoarele un text. După fiecare linie citită, să se afișeze numărul de apariții ale cuvântului în linie.

P5_10. Dintr-un text, introdus de la tastatură se cere să se separe constantele zecimale întregi și să se calculeze suma acestora. Se menționează că acestea pot avea lungimea până la 20 de cifre zecimale.

P5_11. Modificați funcția `strstr()`, astfel încât aceasta să întoarcă în locul adresei, un întreg reprezentând poziția (indicele) în șirul căutat unde începe prima apariție a subșirului sau –1 dacă subșirul nu este găsit.

P5_12. Modificați funcția `strstr()`, astfel încât aceasta să facă o căutare începând dintr-o poziție în șirul destinație, poziție specificată ca al treilea parametru. Funcția va întoarce un întreg reprezentând poziția (indicele) în șirul căutat unde se află prima apariție a subșirului sau –1 dacă subșirul nu este găsit.

P5_13. Modificați funcția `strstr()`, astfel încât aceasta să întoarcă poziția celei de a *n*-a apariții în șirul destinație a subșirului. Valoarea lui *n*, ca și poziția din care începe căutarea se dau ca parametri suplimentari ai funcției.

P5_14. Scrieți o funcție având același efect cu funcția `strtok()`.

P5_15. 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.

P5_16. Un text citit de pe mediul de intrare reprezintă un program C. Să se copieze pe mediul de ieșire, păstrând structura liniilor, dar înlocuind toate comentariile stil C prin comentarii stil C++. Menționăm că în C comentariile se pot întinde pe mai multe linii sau putem avea mai multe comentarii pe o linie. În stil C++ un comentariu ocupă o linie sau un sfârșit de linie.

P5_17. 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 întâi, iar celelalte trec în silaba următoare.

Nu se iau în considerare excepțiile de la aceste reguli.

P5_18. 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.

P5_19. 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.

P5_20. 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 $\frac{N}{2}$

În 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.

Capitolul 6
Structuri

Breviar

Declararea unei structuri (fără alocare de memorie):

```
struct nume_structură {
          declarații_câmpuri
};
```

Definire variabile de tip structură:

```
struct nume_structură listă variabile;
```

Declarare combinată cu definire:

```
struct nume_structură {declaratii campuri} lista_variabile;
```

Acces la un membru (câmp) al structurii:

```
variabilă_structură.nume_membru
```

Acces la un membru printr-un pointer la structură:

```
variabilă structură->nume_membru
```

Probleme rezolvate

R6_1. a) Să se definească tipurile `punct`, `dreptunghi`, `cerc` şi `nor` de puncte ca tipuri structuri. Tipul `punct` va avea drept câmpuri abscisa *x* şi ordonata *y* a punctului, tipul `cerc` va avea drept câmpuri `centrul` - un punct şi `raza` - o valoare reală, tipul `dreptunghi` - colţurile `stânga-jos` şi `dreapta-sus` pentru două vârfuri opuse ale dreptunghiului, care sunt puncte, iar tipul `nor` de puncte - numărul de puncte din nor şi coordonatele lor.
b) Să se definească funcţii având ca parametri un punct şi un dreptunghi (respectiv un cerc), care stabilesc dacă punctul este sau nu interior dreptunghiului (cercului).
c) Să se scrie un program care citeşte: un nor de puncte, un dreptunghi şi un cerc şi care stabileşte câte puncte din nor sunt interioare dreptunghiului şi câte interioare cercului şi afişează aceste informaţii.

Rezolvare: Pentru a stabili dacă un punct este interior unui dreptunghi, verificăm dacă valoarea abscisei sale este cuprinsă între valorile absciselor colţurilor dreptunghiului (şi procedăm la fel şi cu ordonatele).

În cazul cercului, trebuie ca distanţa de la centrul acestuia la punct să fie mai mică decât raza.

6_1.cpp

```
#include <stdio.h>
#include <alloc.h>
#include <assert.h>

typedef struct {
  double x, y;  // coordonatele punctului
} Punct;
typedef struct {
  Punct sj;  // coltul din stanga jos
  Punct ds;  // coltul din dreapta sus
} Dreptunghi;
typedef struct {
  Punct centru;  // centrul cercului
  double r;  // raza
} Cerc;
typedef struct {
  int nr;  // numarul de puncte
  Punct *puncte;
} Nor;
/*
 * functie care intoarce o valoare nenula daca punctul p este in
 * interiorul dreptunghiului d:
 */
int in_dreptunghi(Punct p, Dreptunghi d) {
  return (p.x > d.sj.x && p.x < d.ds.x && p.y > d.sj.y &&
          p.y < d.ds.y);
}
/*
 * functie care intoarce o valoare nenula daca punctul p este in
 * interiorul cercului c:
 */
int in_cerc(Punct p, Cerc c) {
      double d2;
      d2 =   (p.x - c.centru.x) * (p.x - c.centru.x) \
           + (p.y - c.centru.y) * (p.y - c.centru.y);
      return d2 < c.r * c.r;
}
```

```
void main(void) {
  Nor n;
  Dreptunghi d;
  Cerc c;
  int i;
  int cnt_c;  // numarul de puncte din interiorul cercului
  int cnt_d;  // numarul de puncte din interiorul
dreptunghiului
  printf("Introduceti datele dreptunghiului: \n");
  printf(" x(stanga jos) y(stanga jos): ");
  scanf("%lf %lf", &d.sj.x, &d.sj.y);
  printf(" x(dreapta sus) y(dreapta sus): ");
  scanf("%lf %lf", &d.ds.x, &d.ds.y);
  assert(d.sj.x < d.ds.x && d.sj.y < d.ds.y);
  printf("Introduceti datele cercului: \n");
  printf(" x(centru) y(centru): ");
  scanf("%lf %lf", &c.centru.x, &c.centru.y);
  printf(" raza: ");
  scanf("%lf", &c.r);
  assert(c.r > 0);
  cnt_c = cnt_d = 0;
  printf("Introduceti datele norului de puncte: \n");
  printf("Nr. de puncte: ");
  scanf("%d", &n.nr);
  n.puncte = (Punct*) malloc(n.nr * sizeof(Punct));
  printf("Introduceti coordonatele punctelor: \n");
  for (i = 0; i < n.nr; i++) {
    printf(" x%d y%d: ",i,i);
    scanf("%lf %lf", &n.puncte[i].x, &n.puncte[i].y);
    if (in_dreptunghi(n.puncte[i], d))
      cnt_d++;
    if (in_cerc(n.puncte[i], c))
      cnt_c++;
  }
  printf (" %d puncte sunt in interiorul dreptunghiului \n",
\
          cnt_d);
  printf (" %d puncte sunt in interiorul cercului \n",
cnt_c);
  free(n.puncte);
}
```


R6_2. Să se rezolve ecuația $ax^3 + bx^2 + cx + d = 0$, cu $a, b, c, d \in \mathbf{R}$, $a \neq 0$, folosind formulele lui Cardan.

Rezolvare: Pentru a rezolva ecuația cu ajutorul formulelor lui Cardan, parcurgem următorii pași:

1) Făcând substituția: $x = y - \frac{b}{3a}$ (1)

aducem ecuația la următoarea formă canonică: $y^3 + p \cdot y + q = 0$ (2)

în care: $p = \frac{c}{a} - \frac{b^2}{3a^2}$ $\quad q = \frac{2b^3}{27a^3} - \frac{bc}{3a^2} + \frac{d}{a}$ (3)

Discriminantul ecuației canonice este: $D = \frac{p^3}{3} + \frac{q^2}{2}$ (4)

2) Dacă notăm: $u = \sqrt[3]{-\frac{q}{2} + \sqrt{D}}$ $\quad v = \sqrt[3]{-\frac{q}{2} - \sqrt{D}}$ (5)

atunci rădăcinile ecuației canonice (2) sunt:

$$\begin{aligned} y_1 &= u + v \\ y_2 &= e_1 u + e_2 v \\ y_3 &= e_2 u + e_1 v \end{aligned} \quad (6)$$

unde e_1 și e_2 sunt rădăcinile complexe cubice ale unității. Pentru a evita discuția naturii rădăcinilor în funcție de semnul discriminantului, vom considera toate rădăcinile complexe.

3) Pentru a obține din rădăcinile (6) ale ecuației canonice rădăcinile ecuației inițiale, trebuie făcută din nou o substituție *în sens invers*, adică scăzând $\frac{b}{3a}$ din partea reală a rădăcinilor y_i ($i = 1, 2, 3$).

Vom defini o structură pentru numere complexe, iar operațiile cu aceste numere (adunarea, înmulțirea și extragerea rădăcinii de ordin 3) le implementăm ca funcții.

6_2.cpp

```
#include <stdio.h>
#include <math.h>

#define MIN_REAL 1.0e-6
#define PI 3.14159
#define SQR(x) ((x) * (x))

typedef struct {
  double re, im;
} Complex;
```

```
/* adunarea a 2 numere complexe: */
Complex aduna(Complex x, Complex y) {
  Complex z;
  z.re = x.re + y.re;
  z.im = x.im + y.im;
  return z;
}

/* inmultirea a 2 numere complexe: */
Complex mul(Complex x, Complex y) {
  Complex z;
  z.re = x.re * y.re - x.im * y.im;
  z.im = x.re * y.im + x.im * y.re;
  return z;
}

/* radacina de ordinul 3 dintr-un numar complex: */
Complex rad3(Complex x) {
  Complex y;
  double  r;  //modulul lui x
  double teta;  //argumentul lui x
  double rr;  //radacina de ordinul 3 din r

  r = sqrt(SQR(x.re) + SQR(x.im));
  if (r < MIN_REAL) { //modulul este foarte apropiat de 0
    rr = 0.0;
    teta = PI /2.0;
  }
  else
  { rr = exp(log(r) / 3.0);
    teta = atan(x.im / x.re);
  }
  y.re = rr * cos(teta / 3.0);
  y.im = rr * sin(teta / 3.0);
  return y;
}

void main(void) {
  Complex x[3];   // radacinile ecuatiei
  Complex y[3];   // radacinile ecuatiei canonice
  Complex e1, e2; // radacinile de ordinul 3 ale unitatii
  Complex u, v, s, t;
  double a, b, c, d;  //coeficientii ecuatiei
```



```
  double p, q, delta; //coeficienti si discriminant
                      // pentru ecuatia canonica
  int i;

  printf("Introduceti coeficientii ecuatiei \n");
  printf("a b c d: ");
  scanf("%lf %lf %lf %lf", &a, &b, &c, &d);
  /* coeficientii si discriminantul pentru ecuatia canonica: */
  p = c / a - SQR(b / a) / 3;
  q = 2 * (b/3/a) * SQR(b/3/a) - b * c / 3 / SQR(a) + d / a;
  delta = p/3 * SQR(p/3) + SQR(q/2);
  if (delta >= 0) {
    s.im = 0;  s.re = -q/2 + sqrt(delta);
    t.im = 0;  t.re = -q/2 - sqrt(delta);
  }
  else
  { s.im = sqrt(-delta);  s.re = -q/2;
    t.im = -s.im;  t.re = s.re;
  }
  u = rad3(s);
  v = rad3(t);
  e1.re = -0.5;  e1.im = 0.5 * sqrt(3);
  e2.re = e1.re;  e2.im = -e1.im;
  /* calculam radacinile ecuatiei canonice: */
  y[0] = aduna(u, v);
  s = mul(e1, u);
  t = mul(e2, v);
  y[1] = aduna(s, t);
  s = mul(e2, u);
  t = mul(e1, v);
  y[2] = aduna(s, t);
  /* calculam radacinile ecuatiei initiale (facem din nou
substitutie): */
  for (i = 0; i < 3; i++) {
    x[i].re = y[i].re - b / (3 * a);
    x[i].im = y[i].im;
  }
  for (i = 0; i < 3; i++) {
    printf ("\n x%i = %.3lf", i+1, x[i].re);
    if (x[i].im > 0) printf("+");
    if (x[i].im != 0.0) printf("%.3lf*i", x[i].im);
  }
}
```

R6_3. De pe mediul de intrare se citeşte o listă a studenţilor unei grupe şi a notelor obţinute de fiecare la un examen, informaţii cu care se creează un tablou cu componente înregistrări. La introducerea datelor, fiecărui nume i se rezervă primele 20 de caractere din linie, numele mai scurte fiind completate până la 20 cu spaţii libere. O linie se termină cu nota studentului. Lista este terminată printr-o linie vidă. Pot fi maximum 30 de studenţi. Să se scrie un program care:

a) Stabileşte numărul de studenţi prezenţi la examen.

b) Face statistica repartiţiei notelor pe studenţi sub forma:

Nota	Număr studenţi

c) Afişează listele cu studenţii care au obţinut aceeaşi notă.

Rezolvare: Pentru a realiza statistica cerută, creăm un tablou de 10 elemente, în care contorizăm, pe poziţia *i*, numărul studenţilor care au luat nota *i* + 1 (nu *i*, pentru că indexarea în tablou începe de la 0 şi nu de la 1, care este nota minimă posibilă. Am fi putut începe indexarea în tablou şi de la 1, pentru a evita acest decalaj).

6_3.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

typedef struct {
  char nume[21];
  int nota;
} Student;

void main(void) {
  Student grupa[30];
  char linie[25]; //linia curenta care se citeste de la intrare
  int i, j, n = 0;
  int nstud[10]; //vector de contoare pentru fiecare nota
  printf("\nIntroduceti lista cu studentii si notele ");
  printf("(terminata printr-o linie vida)\n");
  gets(linie);
  while (strlen(linie)) {
    /* extragem numele studentului: */
    strncpy(grupa[n].nume, linie, 20);
    grupa[n].nume[20] = 0;
    /* extragem nota si ne asiguram ca valoarea ei e corecta: */
    grupa[n].nota = atoi(linie + 20);
```



```
    assert(grupa[n].nota >= 1 && grupa[n].nota <= 10);
    n++;
    gets(linie);
  }
  printf("Numar studenti = %d \n", n);
  /* initializam contoarele: */
  for (i = 0; i < 10; nstud[i] = 0, i++);
  /* in nstud[i] retinem numarul studentilor cu nota i+1: */
  for (i = 0; i < n; i++)
    nstud[grupa[i].nota-1]++;
  printf("Repartitia studentilor pe note \n");
  printf("Nota  Numar studenti \n");
  for (i = 0; i < 10; i++)
    if (nstud[i] > 0)
      printf(" %2d \t %2d \n", i + 1, nstud[i]);
  /* afisam listele cu studentii care au obtinut aceeasi
nota: */
  for (i = 0; i < 10; i++)
    if (nstud[i] > 0) {
      printf("Lista studentilor avand nota %2d:\n", i + 1);
      for (j = 0; j < n; j++)
        if (grupa[j].nota-1 == i)
          printf("%s \n", grupa[j].nume);
      }
}
```

R6_4. a) Să se definească tipurile: număr `complex` şi `polinom` ca structuri (tipul polinom va avea câmpurile: `grad` – o valoare întreagă şi `coeficenţi` - un tablou cu componente numere complexe).

b) Să se definească funcţii pentru:

- adunarea a două numere complexe
- înmulţirea a două numere complexe
- calculul valorii unui polinom de variabilă complexă cu coeficenţi complecşi
- calculul polinomului derivat al unui polinom dat.

Să se scrie un program care:

- citeşte un polinom cu coeficenţi complecşi,
- citeşte p numere complexe ($p \leq 20$) şi
- stabileşte care dintre acestea sunt rădăcini ale polinomului şi ce multiplicităţi au ele.

Rezolvare:

6_4.cpp

```
#include <math.h>
#include <iostream.h>
#include <iomanip.h>
#define MIN_REAL 1.0e-6

typedef struct {
  double re, im;
} Complex;
typedef struct {
  int grad;
  Complex coef[20]; //coeficientii polinomului
} Polinom;

/* adunarea a 2 numere complexe: */
Complex aduna(Complex x, Complex y) {
  Complex z;
  z.re = x.re + y.re;
  z.im = x.im + y.im;
  return z;
}

/* inmultirea a 2 numere complexe: */
Complex mul(Complex x, Complex y) {
  Complex z;
  z.re = x.re * y.re - x.im * y.im;
  z.im = x.re * y.im + x.im * y.re;
  return z;
}

/* verificare de egalitate: */
int egale (Complex x, Complex y) {
  /* din cauza reprezentarii imprecise a numerelor reale, nu
punem conditia
   * ca valorile acestora sa conicida exact:
   */
  return (fabs(x.re - y.re) < MIN_REAL && fabs(x.im - y.im) <
MIN_REAL);
}

/* afisarea unui numar complex: */
void afiseaza (Complex x) {
  cout << "(" << setprecision(6) << x.re
     << ", " << setprecision(6) << x.im << ')';
}
```



```
/* valoarea polinomului p in punctul x: */
Complex calc_valoare(Polinom p, Complex x) {
  int i;
  Complex val;
  val.re = p.coef[p.grad].re;
  val.im = p.coef[p.grad].im;
  for (i = p.grad - 1; i >= 0; i--)
       val = aduna(mul(val, x), p.coef[i]);
  return val;
}

/* derivarea polinomului: */
Polinom deriveaza(Polinom p) {
  int i;
  Polinom pd;
  if (p.grad == 0) {
    pd.grad = 0;
    pd.coef[0].re = pd.coef[0].im = 0.0;
    return pd;
  }
  pd.grad = p.grad - 1;
  for (i = p.grad; i > 0; i--) {
       pd.coef[i-1].re = i * p.coef[i].re;
       pd.coef[i-1].im = i * p.coef[i].im;
      }
  return pd;
}

void main() {
  int i, p;
  char c;
  int m;  // multiplicitatea radacinii
  Polinom pol;  // polinomul initial
  Polinom pol_d;  // polinomul derivat
  Complex  numere[20], zero;
  /* citirea polinomului: */
  cout << "\n Gradul polinomului = ";
  cin >> pol.grad;
  cout << " Introduceti coeficientii polinomului:" << endl;
  for (i = pol.grad; i >= 0; i--) {
       cout << "a" << i << ".re " << "a" << i << ".im: ";
       cin >> pol.coef[i].re >> pol.coef[i].im;
  }
  /* citirea celor p numere complexe: */
  cout << " Cate numere complexe veti introduce? ";
  cin >> p;
  cout << " Introduceti numerele:" << endl;
```

```
  for (i = 0; i < p; i++) {
       cout << "n" << i << ".re " << "n" << i << ".im: ";
       cin >> numere[i].re >> numere[i].im;
  }
  zero.re = zero.im = 0.0;
  for (i = 0; i < p; i ++) {
    if (egale(calc_valoare(pol, numere[i]), zero)) {
      afiseaza(numere[i]); //numere[i] e radacina
      m = 1;
      pol_d = deriveaza(pol);
      /* calculam multiplicitatea: */
      while (egale(calc_valoare(pol_d, numere[i]), zero) \
                    && pol_d.grad > 0) {
        m++;
        pol_d = deriveaza(pol_d);
      }
      cout << " este radacina cu multiplicitatea " << m << endl;
    }
  }
}
```

R6_5. Un bilet pronosport conține numele jucătorului și 13 caractere 1, 2, *x* constituind reprezentarea codificată a rezultatelor unor meciuri de fotbal.
a) Să se descrie structura unui bilet pronosport .
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 afișează:
- lista jucătorilor, pentru fiecare indicându-se numărul de pronosticuri exacte
- lista jucătorilor, ordonată după numărul de pronosticuri exacte indicate.

Numele jucătorilor se citesc începând din coloana 1, iar pronosticurile din 2.

Rezolvare: Pentru a descrie un bilet, avem nevoie de următoarele câmpuri:
- `numele` jucătorului (șir de caractere)
- `rezultatele` celor 13 meciuri (șir de 13 caractere care pot fi '1', '2' sau 'x')

Numărul de pronosticuri corecte indicate de fiecare jucător se determină folosind funcția definită la punctul b), iar pentru ordonarea listei jucătorilor mai creăm două tablouri:
- `corecte`, în care reținem numărul de pronosticuri corecte date de jucători: `corecte[i]` = numărul de pronosticuri corecte pentru al *i*-lea jucător. Va trebui să sortăm (descrescător) acest tablou, ceea ce înseamnă că se va schimba ordinea

elementelor sale, deci după sortare în `corecte[i]` se va afla numărul de pronosticuri pentru jucătorul *j* (eventual cu $j \neq i$). Pentru a putea determina semnificația elementelor din tabloul `corecte` după sortare, folosim un alt tablou și anume:
- `index`, în care dacă `index[i] = j`, atunci `corecte[i]` = numărul de pronosticuri pentru jucătorul *j*. Inițial `index[i] = i` , cu $i = 0,1, \ldots n$, iar în timpul sortării, dacă interschimbăm două elemente din tabloul `corecte`, le vom interschimba și pe cele corespunzătoare din tabloul `index`.

6_5.cpp
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#define NMAX 14
typedef struct {
  char nume[21];  // numele jucatorului (max.20 caractere +
                  // terminator de sir)
  char rez[NMAX]; //rezultatele indicate
} Bilet;

/* afisarea unui mesaj de eroare si terminarea programului:
*/
void eroare (char* mesaj) {
  perror(mesaj);
  exit(-1);
}

/* determinarea numarului de componente care coincid pentru 2
   vectori:
*/
int nr_comp_egale(char v1[], char v2[]) {
  int i, cnt; // cnt - contor in care se retine nr.
componentelor identice
  if (strlen(v1) != strlen(v2)) return -1;
  cnt = 0;
  for (i = 0;  i < strlen(v1); i++)
       if (v1[i] == v2[i]) cnt++;
  return cnt;
}

/* interschimbarea elementelor cu indicii i si j dintr-un
vector: */
void interschimba(int i, int j, int v[]) {
  int aux;
  aux = v[i];
  v[i] = v[j];
  v[j] = aux;
}
```

```
void main(void) {
  int n, i, j;          // n - numarul de bilete
  int sortat;           // arata daca sortarea s-a terminat
  char rezultate[14];   // rezultatele inregistrate
  Bilet vb[20];
  int corecte[20];//nr. rezultate corecte pentru fiecare jucator
  int index[20];  // indicii elementelor din vb
  printf("Introduceti cele 13 rezultate corecte: ");
  scanf("%s", rezultate);
  printf("Introduceti numarul de jucatori: ");
  scanf("%d", &n);
  printf("Numele jucatorilor si pronosticurile lor:\n");
  for (i = 0; i < n; i++) {
    printf("Nume jucator %d: ", i + 1);
    scanf("%s", vb[i].nume);
    printf ("Pronostic jucator %d: ", i + 1);
    scanf("%s", vb[i].rez);
    /* initializam vectorul index: */
    index[i] = i;
  }
  /* afisam lista jucatorilor: */
  printf("Lista jucatori si numar de pronosticuri
corecte):\n");
  for (i = 0; i < n; i++) {
     corecte[i] = nr_comp_egale(vb[i].rez, rezultate);
     printf("%s  %d\n", vb[i].nume, corecte[i]);
  }
  /* sortam vectorul "corecte" prin metoda "Bubble sort": */
  sortat=0;
  for(i = 1;i < n && !sortat; i++) {
    sortat = 1;
    for(j = n - 1;j >= i; j--)
      if(corecte[j] > corecte[j-1]) {
        interschimba(j, j - 1, corecte);
        /* interschimbare si in vectorul index: */
        interschimba(j, j - 1, index);
       sortat=0;
      }
  }
  printf("Lista jucatorilor dupa rezultatele corecte):\n");
  for (i = 0; i < n; i++)
    printf("%s %d\n", vb[index[i]].nume, corecte[i]);
  getch();
}
```



```
    index[i] = i;
  }
  /* afisam lista jucatorilor: */
  printf("Lista jucatori si numar de pronosticuri
corecte):\n");
  for (i = 0; i < n; i++) {
     corecte[i] = nr_comp_egale(vb[i].rez, rezultate);
     printf("%s  %d\n", vb[i].nume, corecte[i]);
  }
  /* sortam vectorul "corecte" prin metoda "Bubble sort": */
  sortat=0;
  for(i = 1;i < n && !sortat; i++) {
    sortat = 1;
    for(j = n - 1;j >= i; j--)
      if(corecte[j] > corecte[j-1]) {
        interschimba(j, j - 1, corecte);
        /* interschimbare si in vectorul index: */
        interschimba(j, j - 1, index);
       sortat=0;
      }
  }
  printf("Lista jucatorilor dupa rezultatele corecte):\n");
  for (i = 0; i < n; i++)
    printf("%s %d\n", vb[index[i]].nume, corecte[i]);
  getch();
}
```

R6_6. a) Să se definească tipurile `punct` şi `triunghi` ca structuri.
b) Să se definească o funcţie având ca parametru un triunghi, şi care calculează aria acestuia.
c) Să se definească o funcţie având ca parametri un punct şi un triunghi, şi care stabileşte 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 citeşte 4 puncte şi determină folosind funcţia de la punctul c) dacă acestea pot forma un patrulater convex.

***Rezolvare*:** Tipul `punct` îl vom descrie prin cele 2 coordonate ale sale, iar tipul triunghi – prin cele 3 vârfuri, care sunt de tipul *punct*.

Aria triunghiului se calculează după formula: $A = \frac{1}{2} \cdot abs\left(\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}\right)$

unde (x_i, y_i) sunt coordonatele vârfurilor, iar cu `abs` am notat valoarea absolută.

```
/* verifica daca punctul p este interior triunghiului t: */
int este_interior(Punct p, Triunghi t) {
  Triunghi t1, t2, t3;
  t1.v1 = p;  t1.v2 = t.v1;  t1.v3 = t.v2;
  t2.v1 = p;  t2.v2 = t.v2;  t2.v3 = t.v3;
  t3.v1 = p;  t3.v2 = t.v3;  t3.v3 = t.v1;
  return (fabs(arie(t1) + arie(t2) + arie(t3) - arie(t)) <
MIN_REAL);
}

void main(void) {
  Punct p[4];
  Triunghi t;
  int i;
  int convex; // variabila care arata daca patrulaterul e convex
  cout << "Introduceti coordonatele punctelor:" << endl;
  for (i = 0; i < 4; i++) {
    cout << "p" << i << ".x = ";
    cin >> p[i].x;
    cout << "p" << i << ".y = ";
    cin >> p[i].y;
  }
  convex = TRUE;
  /*
   * verificam, pentru fiecare varf al patrulaterului, daca
este interior
   * triunghiului format de celelalte 3 varfuri:
   */
  for ( i = 0; i < 4; i++) {
    /* folosim permutari circulare: */
    t.v1 = p[i%4];  t.v2 = p[(i+1)%4];  t.v3 = p[(i+2)%4];
    if (este_interior(p[(i+3)%4], t)) {
      convex = FALSE;
      break;
    }
  }
  if (convex) cout << "Patrulaterul este convex" << endl;
    else cout << "Patrulaterul nu este convex" << endl;
}
```


R6_7. Să se actualizeze stocurile dintr-un depozit în urma satisfacerii a n comenzi ($n \leq 100$). O comandă este specificată prin:
- `cantitate` (o valoare întreagă) şi
- `tipul` produsului comandat (un tablou de 8 caractere).

Situaţia celor p produse din depozit ($p \leq 200$) este dată prin:
- `stoc` şi `stoc minim` - valori întregi şi
- `cod produs` - un tablou de 8 caractere.

O comandă va fi onorată dacă produsul comandat se află în depozit şi dacă prin satisfacerea comenzii stocul nu scade sub stocul minim de siguranţă. Programul va crea şi afişa:
- stocurile actualizate în urma satisfacerii comenzilor
- lista comenzilor neonorate datorită unor stocuri insuficiente
- lista comenzilor de produse inexistente în depozit.

Rezolvare:

6_7.cpp

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <assert.h>

typedef struct {
  char cod[8];  // codul produsului
  int cant;  // cantitatea
} Comanda;
typedef struct {
  char cod[8];
  int stoc, stoc_min;
} Produs;

void main(void) {
  Comanda comenzi[100];
  Comanda neonorate1[100];  // comenzi de produse inexistente
  Comanda neonorate2[100];  // comenzi neonorate (stocuri
                            // insuficiente)
  Produs depozit[200];
  int i, j;
  int p;  // numarul de produse din depozit
  int nc; // numarul de comenzi
  int cnt1, cnt2;  //numar de elemente din vectorii neonorate1,2
  printf("Numarul de produse din depozit : ");
  scanf("%d", &p);
  for (i = 0; i < p; i++) {
```

```
    printf("Produs %d - cod : ", i);
    scanf("%s", depozit[i].cod);
    printf("Produs %d - stoc : ", i);
    scanf("%d", &depozit[i].stoc);
    printf("Produs %d - stoc minim : ", i);
    scanf("%d", &depozit[i].stoc_min);
    assert(depozit[i].stoc > depozit[i].stoc_min);
  }
  printf("Numarul de comenzi : ");
  scanf("%d", &nc);
  for (i = 0; i < nc; i++) {
    printf("Comanda %d - cod : ", i);
    scanf("%s", comenzi[i].cod);
    printf("Comanda %d - cantitate : ", i);
    scanf("%d", &comenzi[i].cant);
    assert(comenzi[i].cant > 0);
  }
  cnt1 = cnt2 = 0;
  for (i = 0; i < nc; i++) {
    /* cautam produsul in depozit: */
    for (j = 0; j < p - 1 &&
         strcmp(depozit[j].cod, comenzi[i].cod) != 0; j++)
      ;
    /*
     * depozit[j] este fie produsul cautat, fie ultimul produs
     * din deposit (deoarece ciclul for "merge" doar pana la
     * penultimul element)
     */
    if (!strcmp(depozit[j].cod, comenzi[i].cod)) {
      if (comenzi[i].cant <= depozit[j].stoc -
                             depozit[j].stoc_min)
      // comanda poate fi onorata, actualizam valoarea stocului:
        depozit[j].stoc -= comenzi[i].cant;
      else  //stoc insuficient
            neonorate2[cnt2++] = comenzi[i];
    }
    else  //produsul nu a fost gasit
      neonorate1[cnt1++] = comenzi[i];
  }
  printf("Stocurile actualizate: \n");
  printf("Cod \t\t Stoc  \t Stoc minim \n");
  for (i = 0; i < p; i++)
    printf("%8s \t %4d \t %4d\n", depozit[i].cod, \
           depozit[i].stoc, depozit[i].stoc_min);
  if (cnt1 > 0) {
```



```
    printf("Comenzile de produse inexistente: \n");
    printf("Cod \t\t Cantitate \n");
    for (i = 0; i < cnt1; i++)
      printf("%8s \t %4d\n",neonorate1[i].cod, \
             neonorate1[i].cant);
  }
  else
    printf("Nu au fost comenzi de produse inexistente\n");
  if (cnt2 > 0) {
    printf("Comenzi neonorate datorita stocuri insuficiente:");
    printf("\nCod \t\t Cantitate \n");
    for (i = 0; i < cnt2; i++)
      printf("%8s \t %4d\n", neonorate2[i].cod, \
             neonorate2[i].cant);
  }
  else
    printf("Stocuri suficiente pentru toate produsele\n");
  getch();
}
```

R6_8. 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 două dată o precede pe prima

Funcția va întoarce una din valorile −1, 0, 1.

Scrieți o funcție `main()` care citește două date calendaristice, le validează și stabilește succesiunea lor cronologică.

Rezolvare:

6_8.cpp

```
#include <stdio.h>

typedef struct {
  unsigned int an;
  unsigned int luna;
  unsigned int zi;
} Data;

/* vector in care retinem ultima zi pentru fiecare luna: */
int zile[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

```
/* verificare daca anul este bisect (daca da, se intoarce o
   valoare nenula): */
int bisect(int an) {
  return ((an %4 == 0 && an % 100 != 0) || an % 400 == 0);
}

/* verifica daca o data este valida (daca da, se intoarce o
   valoare nenula):*/
int e_valida(Data d) {
  if (d.luna > 12 || d.luna < 1 || d.zi < 1) return 0;
  /* daca anul este bisect si luna este februarie: */
  if (bisect(d.an) && d.luna == 2) return (d.zi <= 29);
  /* anul nu este bisect sau luna nu este februarie: */
  return (d.zi <= zile[d.luna - 1]);
}

/*
 * Functie care compara 2 date calendaristice
 * Rezultat intors: -1 daca d1 precede d2
 *                   0 daca datele sunt egale
 *                   1 daca d2 precede d1
 */
int compara_date(Data d1, Data d2) {
   if (d1.an < d2.an) return -1;
   if (d1.an > d2.an) return 1;
   /* d1.an = d2.an : */
   if (d1.luna < d2.luna) return -1;
   if (d1.luna > d2.luna) return 1;
   /* d1.an = d2.an si d1.luna = d2.luna : */
   if (d1.zi < d2.zi) return -1;
   if (d1.zi > d2.zi) return 1;
   /* d1.an = d2.an si d1.luna = d2.luna  si d1.zi = d2.zi : */
   return 0;
}

void main(void) {
  Data d1, d2;
  do {
    printf("Introduceti data d1 (zi luna an)\n");
    scanf("%d %d %d", &d1.zi, &d1.luna, &d1.an);
    if (!e_valida(d1))
      printf ("Data nu este valida, introduceti una corecta\n");
  } while (!e_valida(d1));
  do {
    printf("Introduceti data d2 (zi luna an)\n");
```



```
    scanf("%d %d %d", &d2.zi, &d2.luna, &d2.an);
    if (!e_valida(d2))
      printf ("Data nu este valida, introduceti una \
               corecta!\n");
  } while (!e_valida(d2));
  switch (compara_date(d1, d2)) {
    case -1:
      printf("d1 precede d2\n");
      break;
    case  0:
      printf("Datele sunt egale\n");
      break;
    case  1:
      printf("d1 urmeaza dupa d2\n");
  }
}
```

R6_9. Definiți tipul mulțime de numere reale ca o structură cu următoarele câmpuri:
- numărul de elemente (o valoare întreagă)
- valorile elementelor (un tablou de numere reale, 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, ș.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.

Rezolvare: După cum sugerează indicația, pentru stabilirea multiplicităților vom proceda astfel: reținem rădăcinile într-o mulțime (notată în program cu `rad1`) și derivăm polinomul în mod repetat, la fiecare pas eliminând din mulțimea respectivă rădăcinile cărora le-am putut stabili multiplicitatea. Algoritmul continuă până când am epuizat toate rădăcinile (mulțimea devine vidă).

6_9.cpp

```
#include <stdio.h>
#include <assert.h>
#define NMAX 100

typedef struct {
  int card;  //cardinalul multimii
  double elem[NMAX];  //elementele
} Multime;
typedef struct {
  int grad;
  double coef[NMAX];
} Polinom;

/* verifica daca numarul x apartine multimii m: */
int apartine(double x, Multime m) {
  int i;
  for (i = 0; i < m.card; i++)
    if (x == m.elem[i]) return 1;
  return 0;
}

/* diferenta a 2 multimi: */
Multime diferenta(Multime m1, Multime m2) {
  int i;
  Multime dif;

  dif.card = 0;
  for (i = 0; i < m1.card; i++) {
    if (!apartine(m1.elem[i], m2))
      dif.elem[dif.card++] = m1.elem[i];
  }
  return dif;
}

/* adauga un element la o multime: */
void adauga(double x, Multime& m) {
  assert(m.card < NMAX);
  m.elem[m.card] = x;
  m.card++;
}

/* afiseaza elementele unei multimi: */
void afiseaza(Multime m) {
  int i;
  for (i = 0; i < m.card; i++)
    printf ("%.2lf  ", m.elem[i]);
  printf("\n");
}
```



```
/* valoarea polinomului p in punctul x: */
double calc_valoare(Polinom p, double x) {
  int i;
  double val = 0.0;
  for (i = p.grad; i >= 0; i--)
    val = val * x + p.coef[i];
  return val;
}

/* deriveaza polinomului p (rezultatul se pune in pd): */
void deriveaza(Polinom p, Polinom& pd) {
  int i;
  if (p.grad == 0) {
    pd.grad = 0;
    pd.coef[0] = 0.0;
  }
  else
  { pd.grad = p.grad - 1;
    for (i = p.grad; i > 0; i--)
      pd.coef[i-1] = i * p.coef[i];
  }
}

void main(void) {
  Polinom p;         // polinomul initial
  Polinom pd1, pd2; // folosite pentru derivare
  Multime radacini; //radacinile polinomului
  Multime rad1, rad2, dif;
  int i, k;
  int mult;//in pasul curent,radacinile au multiplicitate>= mult
  printf(" Gradul polinomului = ");
  scanf("%d", &p.grad);
  assert(p.grad < NMAX);
  printf(" Introduceti coeficientii polinomului:\n");
  for (i = 0; i <= p.grad; i++) {
      printf("a%d = ", i);
      scanf("%lf", &p.coef[i]);
  }
  printf(" Numarul de radacini = ");
  scanf("%d", &radacini.card);
  assert(radacini.card < NMAX);
```

```
  printf("Introduceti radacinile:\n");
  k = -1;          // numara radacinile reale ale polinomului
  for (i = 0; i < radacini.card; i++) {
      k++;
      printf("x%d = ", k);
      scanf("%lf", &radacini.elem[k]);
      if (calc_valoare(p, radacini.elem[k]) != 0) {
        printf ("\n\t %lf NU ESTE radacina a polinomului. \n",\
                    radacini.elem[k]);
        k--;
        }
  }
  radacini.card = k + 1;
  assert (radacini.card > 0);
  /*copiem p in pd1 si radacini in rad1, pentru a nu pierde
    valoarea initiala:*/
  rad1 = radacini;
  pd1 = p;
  /*in primul pas, radacinile din multimea rad1 au
    multiplicitatea cel putin 1 */
  mult = 1;
  while (rad1.card > 0) {
      rad2.card = 0;  //multimea rad2 e initial vida
      deriveaza(pd1, pd2);
      /* punem in multimea rad2 radacinile polinomului derivat:
      */
      for (i = 0; i < rad1.card; i++) {
        if (calc_valoare(pd2, rad1.elem[i]) == 0.0)
            adauga(rad1.elem[i], rad2);
      }
      dif = diferenta(rad1, rad2);
      if (dif.card) {
        printf("Radacinile cu multiplicitatea %d:\n", mult);
        afiseaza(dif);
      }
      mult++;
      pd1 = pd2;
      rad1 = rad2;
  }
}
```


R6_10. 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, făcând vectorizarea matricii pe linii.
Să se definească funcții pentru adunarea și înmulțirea a două matrici rare, precum și funcții pentru crearea structurii matrice rară 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.

Rezolvare:

1) *Crearea structurii matrice rară:*

Vom crea pentru memorarea unei matrici rare o structură cu următoarele câmpuri:

- `nlin, ncol` – numărul de linii/coloane
- `nelem` – numărul de elemente nenule
- `elem` – tablou cu elementele nenule
- `poziții` – tablou cu pozițiile liniarizate ale elementelor nenule

Dacă al *k*-lea element nenul dintr-o matrice rară se află pe linia *l* și coloana *c*, atunci poziția liniarizată se calculează astfel:

```
poziții[k] = l * ncol + c
```

Numerotarea liniilor și coloanelor începe de la 0.

2) *Afișarea unei matrici rare:*

Pentru a calcula linia și coloana pe care se află elementul de pe poziția liniarizată poz, folosim relațiile:

```
Linie = poz / ncol
Coloană = poz % ncol
```

Deoarece matricea rară poate avea dimensiuni foarte mari, când facem afișarea nu vom reține în memorie toate liniile acesteia, ci doar linia curentă (care se afișează în momentul respectiv). Pentru fiecare linie a matricii funcția realizează următoarele operații:

- inițializează linia curentă cu 0
- parcurge tabloul cu elemente nenule, calculând linia și coloana acestora și le pune pe poziția corespunzătoare în linia curentă. La primul element care nu mai este situat pe linia curentă se oprește, afișează linia și continuă algoritmul cu linia următoare, tabloul cu elemente nenule parcurgându-se în continuare din poziția unde se rămăsese înainte.

3) *Adunarea a două matrici rare:*

Vectorii pozițiilor liniarizate ai matricilor rare fiind ordonați strict crescător, adunarea matricilor se face interclasând cei doi vectori. Elementele aflate pe aceleași poziții în cele două matrici se adună, iar celelalte se copiază în matricea sumă.

4) *Înmulțirea a două matrici rare:*

De această dată nu mai putem face interclasare, ci procedăm astfel:
calculăm fiecare element $c[i][j]$ al produsului de matrici $a \cdot b$ după formula:

$$c[i][j] = \sum_{i=0}^{n-1} a[i][k] \cdot b[k][j]$$

Deci, pentru fiecare k, vom determina dacă în matricile a și b există elemente nenule pe pozițiile $[i][k]$ și $[k][j]$ (acest lucru este realizat de funcția `caută_poz_matrice()`). Dacă există, adăugăm produsul lor la suma de mai sus.

6_10.cpp

```
#include <stdio.h>
#include <conio.h>
#define NMAX 100  //numarul maxim de elemente nenule din matrice

typedef struct {
  int nlin, ncol;  // numarul de linii, coloane
  int nelem;  // numarul de elemente nenule
  float elem[NMAX];  // elementele nenule
  int pozitii[NMAX];  // pozitiile in care se afla elementele
nenule
} Mat_rara;

/* determinarea liniei l si coloanei c a unui element din
 * matrice, cunoscand pozitia liniarizata poz (si numarul
 * de coloane din matrice - col)
 */
void determina_indici(int poz, int ncol, int& l, int& c) {
  c = poz % ncol;
  l = poz / ncol;
}

/* verificare daca in matrice exista element nenul in pozitia
 * (liniarizata) poz; daca exista se intoarce indicele
acestuia
 * in vectorul ce retine elementele nenule (vect_poz); daca nu,
 * se intoarce -1
 */
int cauta_poz_liniara(int poz, int vect_poz[], int nelem) {
  int gasit = 0;
  int i = 0;
  while (i < nelem && !gasit) {
    if (vect_poz[i] == poz)
      gasit = 1;
```



```
    if (!gasit) i++;
  }
  if (!gasit) return -1;
  return i;
}

/* asemanator cu functia de mai sus, dar se cauta dupa linie
si coloana: */
int cauta_poz_matrice(int l, int c, Mat_rara mat) {
  int poz = l * mat.ncol + c;
  return cauta_poz_liniara(poz, mat.pozitii, mat.nelem);
}

/* citirea si crearea unei matrici rare: */
Mat_rara creeaza_mat_rara() {
  Mat_rara mat;
  int l, c, i;
  printf("Nr. de linii: ");
  scanf("%d", &mat.nlin);
  printf("Nr. de coloane: ");
  scanf("%d", &mat.ncol);
  printf("Nr. de elemente nenule: ");
  scanf("%d", &mat.nelem);
  printf("Introduceti elementele nenule: \n");
  for (i = 0; i < mat.nelem; i++) {
    printf("Elementul %d : ", i + 1);
    scanf("%f", &mat.elem[i]);
    printf("Linia pt. elementul %d : ", i + 1);
    scanf("%d", &l);
    printf("Coloana pt.elementul %d : ", i + 1);
    scanf("%d", &c);
    /* calculul pozitiei liniarizate: */
    mat.pozitii[i] = (l - 1) * mat.ncol + (c - 1);
  }
  return mat;
}

/* afisarea unei matrici rare: */
void afiseaza_mat_rara(Mat_rara mat) {
  int l, i, j, poz;
  float lin_crt[NMAX]; //linia curenta afisata a matricii
  poz = 0; // indice in vectorul cu elemente nenule
  /* afisam matricea linie cu linie: */
  for (l = 0; l < mat.nlin; l++) {
    for (i = 0; i < mat.ncol; i++)
      lin_crt[i] = 0.0;
```

```
    while (poz < mat.nelem) {
      /* calculam linia si coloana elementului curent: */
      determina_indici(mat.pozitii[poz], mat.ncol, i, j);
      if (i > 1) break; // s-au terminat elementele nenule de pe
                        // linia curenta
      lin_crt[j] = mat.elem[poz];
      poz++;
    }
    printf("\n");
    /* afisam linia curenta: */
    for (i = 0; i < mat.ncol; i++)
      printf("% 6.2f", lin_crt[i]);
  }
}

/* adunarea a doua matrici rare (c <- a + b): */
int aduna_mat_rare(Mat_rara a, Mat_rara b, Mat_rara& c) {
  int poz_a, poz_b, poz_c;  // indici in vectorii cu elemente
                            // nenule
  /* dimensiunile matricelor trebuie sa coincida: */
  if (a.nlin != b.nlin || a.ncol != b.ncol) {
    printf ("\n Matricele nu se pot aduna");
    return -1;
  }
  /* initializam matricea suma: */
  c.nlin = a.nlin;
  c.ncol = a.ncol;
  c.nelem = 0;
  poz_a = poz_b = poz_c = 0;
  /* parcurgem vectorii cu elemete nenule ai celor doua matrici:
  */
  while (poz_a < a.nelem && poz_b < b.nelem)    {
    if (a.pozitii[poz_a] == b.pozitii[poz_b]) {
      /* am gasit 2 elemente cu aceeasi pozitie, le adunam: */
      c.elem[poz_c] = a.elem[poz_a++] + b.elem[poz_b++];
      c.pozitii[poz_c++] = a.pozitii[poz_a - 1];
      c.nelem++;
      continue;
    }
    if (a.pozitii[poz_a] > b.pozitii[poz_b]) {
       /* in matricea b ne aflam la o pozitie "mai mica" decat
        * pozitia din a; copiem elementul in c si inaintam:
        */
```



```
      c.elem[poz_c] = b.elem[poz_b];
      c.pozitii[poz_c++] = b.pozitii[poz_b++];
      c.nelem++;
      continue;
    }
    if (a.pozitii[poz_a] < b.pozitii[poz_b]) {
      c.elem[poz_c] = a.elem[poz_a];
      c.pozitii[poz_c++] = a.pozitii[poz_a++];
      c.nelem++;
    }
  }

  /* una dintre matrici a fost parcursa in întregime, copiem in
   * c restul elementelor din cealalta:
   */
  if (poz_a < a.nelem)
    for (; poz_a < a.nelem;
         c.elem[poz_c++] = a.elem[poz_a++], c.nelem++)
      c.pozitii[poz_c] = a.pozitii[poz_a];
  else
    for (; poz_b < b.nelem;
         c.elem[poz_c++] = b.elem[poz_a++], c.nelem++)
      c.pozitii[poz_c] = b.pozitii[poz_b];
  /*
   * in cele doua "for"-uri de mai sus se poate observa ordinea
   * in care se executa instructiunile(mai intai se face
   *  atribuirea din corpul ciclului si apoi celelalte,
   * in care se incrementeaza si indicii)
   */
  return 1;
}

/* inmultirea a 2 matrici rare (c <- a * b): */
int inmulteste_mat_rare(Mat_rara a, Mat_rara b, Mat_rara& c)
{
  int i, j, k;
  int poz_a, poz_b, poz_c; //indici in vectorii cu elemente
                           //nenule
  float cij; // valoarea elementului c[i][j]
  if (a.ncol != b.nlin) {
    printf("\n Matricile nu se pot inmulti");
    return -1;
  }
```

```
  c.nlin = a.nlin;
  c.ncol = b.ncol;
  c.nelem = 0;
  poz_c = 0;
  for (i = 0; i < c.nlin; i++)
    for (j = 0; j < c.ncol; j++) {
      /* calculam elementul [i,j] din matricea produs: */
      cij = 0.0;
      for (k = 0; k < a.ncol; k++) {
        /* cautam elementele din pozitiile [i,k] si [k,j] in a,
           respectiv b: */
        poz_a = cauta_poz_matrice(i, k, a);
        poz_b = cauta_poz_matrice(k, j, b);
        if (poz_a >= 0 && poz_b >= 0)
          cij += a.elem[poz_a] * b.elem[poz_b];
      }
      if (cij != 0.0) {
        c.elem[poz_c] = cij;
        c.pozitii[poz_c++] = i * c.ncol + j;
        c.nelem++;
      }
    }
  return 1;
}

void main(void)  {
  Mat_rara a, b, s, p;
  printf("Matricea a \n" );
  printf(" (numerele de linii/coloane incep de la 1) \n");
  a = creeaza_mat_rara();
  printf("Matricea b(numerotare indici de la 1) \n");
  b = creeaza_mat_rara();
  printf(" Matricea suma este: \n");
  aduna_mat_rare(a, b, s);
  afiseaza_mat_rara(s);
  printf("\n Matricea produs este: \n");
  inmulteste_mat_rare(a, b, p);
  afiseaza_mat_rara(p);
  getch();
}
```


Probleme propuse

P6_1. Un experiment fizic este precizat prin numărul de determinări şi valorile măsurate.

a) Să se definească structura experiment.

b) Să se definească o funcţie având ca parametru un experiment, care calculează media aritmetică a măsurătorilor.

c) Să se scrie un program care citeşte numărul de determinări şi valorile lor şi creează cu acestea o înregistrare şi calculează, folosind funcţia de mai sus, abaterea standard : $\sqrt{\frac{\sum_{i=1}^{n} x_i^2 - n\left(\sum_{i=1}^{n} \frac{x_i}{n}\right)^2}{n-1}}$

P6_2. a) Să se definească tipul `punct` ca o structură.

b) Să se definească o funcţie, având ca parametri trei puncte, care stabileşte dacă acestea sunt sau nu coliniare.

c) Să se scrie un program care citeşte un întreg n ($n \le 50$) şi n puncte şi afişează numerele tripletelor de puncte coliniare.

P6_3. Pentru aprovizionarea unui magazin se lansează n comenzi ($n \le 100$). O comandă este precizată prin două elemente:

- tipul produsului comandat (un tablou de 8 caractere) şi
- 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. Dacă două comenzi diferite i şi j cu $i < j$ se referă la un acelaşi produs vom comasa în comanda i cantitatea comandată în j şi vom anula comanda j.

O comandă va fi caracterizată deci prin:

- tipul produsului comandat,
- cantitatea comandată şi
- faptul că este o comandă în vigoare sau a fost anulată. În procesul de centralizare a comenzilor se vor face toate comparaţiile posibile între comenzi diferite, care n-au fost anulate.

P6_4. La o disciplină cu verificare pe parcurs, fiecărui student i s-au acordat trei note la trei lucrări de control şi un calificativ pentru activitatea la seminar (insuficient, suficient, bine şi foarte bine). Pe baza acestor informaţii se acordă o notă finală în felul următor: se face media celor 3 note la care se adaugă 0, 0.25, 0.5 sau 0.75 conform calificativului, iar rezultatul se trunchiază.

a) Să se definească tipul `situaţie` ca un tip structură având drept câmpuri:
   - `numele` - un tablou de 20 de caractere,
   - `notele` - trei valori întregi între 1 şi 10 şi
   - `calificativul` - un caracter (I, S, B, F).

b) Să se definească o funcţie având ca parametru o situaţie, care calculează nota finală.

c) Să se scrie un program care citeşte pentru cei *n* studenţi
   - `numele` - din primele 20 de poziţii
   - `calificativul` - un caracter in poziţia 21
   - `cele 3 note` - valori întregi, separate prin spaţii

   şi afişează lista studenţilor promovaţi şi lista studenţilor care au obţinut note de 9 şi 10.

P6_5. O dată calendaristică este exprimată prin trei valori întregi: anul, luna şi ziua. O persoană este precizată prin: nume şi prenume (maxim 30 de caractere) şi data naşterii - o dată calendaristică.

a) Să se descrie tipurile `dată` şi `persoană` ca structuri.

b) Să se definească o funcţie având ca parametru o persoană, funcţie care calculează vârsta persoanei în ani împliniţi.

c) Să se scrie un program care citeşte o listă de *n* persoane (*n* este citit înaintea listei) şi datele lor de naştere şi folosind funcţia definită mai sus afişează lista persoanelor majore.

P6_6. a) Să se descrie tipurile `punct` şi `dreaptă` ca tipuri structură.

b) Să se definească o funcţie având ca parametri două drepte, un punct şi o variabilă întreagă, funcţie care stabileşte dacă cele două drepte se intersectează, caz în care calculează coordonatele punctului de intersecţie, sau dacă sunt paralele; parametrul boolean separă situaţia drepte paralele / concurente.

c) Să se scrie un program care citeşte *n* drepte ($n \leq 100$) şi afişează perechile de drepte paralele, iar pentru fiecare pereche de drepte neparalele - coordonatele punctului de intersecţie. 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)
```


P6_7. Definiţi o structură "`divmul`" având ca membri: un întreg şi două tablouri cu componente întregi.
Definiţi o funcţie care determină primul divizor al unui număr (întreg lung fără semn) şi multiplicitatea acestuia. Funcţia are trei parametri: numărul, divizorul şi multiplicitatea (ultimii doi parametri reprezintă rezultate calculate de funcţie).
Definiţi o funcţie care primind un număr întreg determină: numărul divizorilor primi distincţi, tabloul divizorilor primi şi tabloul multiplicităţilor divizorilor primi. Funcţia are un singur parametru - numărul întreg, iar rezultatele sunt întoarse printr-o structură `divmul`.
Definiţi o funcţie care primeşte doi parametri numere întregi şi întoarce ca rezultat o structură `divmul` (conţinând număr de divizori, tabloul divizorilor şi tabloul multiplicităţilor divizorilor) din care se poate calcula c.m.m.d.c. al numerelor. Se ştie că c.m.m.d.c. conţine divizorii comuni ai celor două numere cu multiplicităţile cele mai mici.
Definiţi o funcţie având ca parametru o structură `divmul`, care întoarce ca rezultat c.m.m.d.c.
Definiţi o funcţie `main()` care citeşte *n* numere întregi şi calculează c.m.m.d.c. al lor.

P6_8. Definiţi o structură "`rational`" având ca membri doi întregi pozitivi.
Definiţi o funcţie având ca parametri doi întregi, funcţie care întoarce ca rezultat o structură `rational`.
Definiţi o funcţie având ca parametru o structură rational, care întoarce ca rezultat numărătorul fracţiei raţionale (acelaşi lucru şi pentru numitor).
Definiţi o funcţie care simplifică o fracţie raţională. Funcţia are ca parametru o structură - fracţia raţională şi întoarce ca rezultat tot o structură - fracţia simplificată.
Definiţi o funcţie care adună două fracţii raţionale. Funcţia are ca parametri două structuri, fracţiile de adunat, şi întoarce ca rezultat fracţia raţională sumă.
Definiţi o funcţie `main()` care:
- citeşte un întreg $n \leq 20$ şi *n* fracţii raţionale
- calculează suma celor *n* fracţii raţionale şi afişează rezultatul fracţie raţională, folosind funcţiile definite mai sus.

P6_9. Definiţi structură "`matrice`" având ca membri doi întregi reprezentând numărul de linii, respectiv coloane ale matricii un tablou cu 2 dimensiuni (cu limitele 10 şi 10 în care se păstrează elementele matricei).
Definiţi o funcţie care înmulţeşte două matrici. Funcţia va avea 3 parametri, cele două structuri matrici care se înmulţesc şi produsul şi va întoarce ca rezultat 1/0 după cum înmulţirea matricilor este sau nu posibilă.
Definiţi o funcţie care compară o matrice cu matricea unitate. Funcţia are un parametru structura matrice de comparat şi întoarce un rezultat întreg 0/1.

Definiți o funcție care inițializează prin citire o structură matrice. Funcția verifică dacă numărul de linii și coloane sunt mai mici sau egale cu 10, repetând citirea în caz contrar. Funcția nu are parametri și întoarce ca rezultat structura inițializată prin citire.
Definiți o funcție `main()` care:
- citește două matrici
- verifică dacă una este inversa celeilalte, afișând un mesaj corespunzător.

P6_10. Definiți structurile "`complex`" (câmpuri `parte reala` și `parte imaginara`) și "`polinom`" rar cu coeficienți complecși (câmpuri `numar termeni`, tablou `coeficienti complecși` și tablou `întreg exponenti`).
Definiți o funcție care adună două numere complexe. Funcția are doi parametri - structuri "`complex`" și întoarce ca rezultat o structură "`complex`" (suma).
Definiți o funcție care adună două polinoame rare cu coeficienți complecși. Funcția are 3 parametri: două structuri "`polinom`" și un pointer la structura polinom rezultat.
Definiți o funcție care inițializează prin citire de la tastatură o structură polinom. Funcția nu are parametri și întoarce o structură "`polinom`".
Definiți o funcție care afișează la terminal o structură polinom. Funcția are un parametru structură și nu întoarce nimic.
Scrieți o funcție `main()` care citește două polinoame rare cu coeficienți complecși, calculează polinomul sumă și îl afișează.
Indicație: În polinomul sumă se copiază coeficienții și exponenții celor două polinoame și se adună numărul de termeni nenuli. Se reduce apoi numărul de termeni, comasând termenii care au același exponent într-un singur termen.

Capitolul 7
Fișiere

Breviar

Fișierul `<stdio.h>` conține prototipurile următoarelor funcții:

Semnătură	Efect
`FILE* fopen(char* nume, char* mod);`	Deschide fișierul. Întoarce un pointer la fișierul deschis sau NULL dacă operația eșuează.
`int fclose(FILE* pf);`	Închide fișierul.
`int fgetc(FILE* pf);`	Citește un caracter din fișier și întoarce caracterul citit sau EOF.
`int fputc(char c, FILE* pf);`	Scrie caracterul primit ca parametru în fișier.
`char* fgets(char* s, int n, FILE* pf);`	Citește din fișier în `s` cel mult `n-1` caractere, sau până la întâlnirea lui `'\n'`, în locul căruia pune `'\0'`. Întoarce `s` sau NULL, dacă s-a citit EOF.
`int fputs(char* s, FILE* pf);`	Copiază șirul în fișierul de ieșire. Înlocuiește terminatorul `'\0'` cu `'\n'`.
`int fscanf(FILE* pf, char* format, lista_adrese);`	Citește din fișier sub controlul formatului. Întoarce numărul de câmpuri citite sau EOF, în caz de eroare sau sfârșit de fișier. Se folosește cu fișiere text.
`int fprintf(FILE* pf, char* format, lista_expresii);`	Scrie în fișier sub controlul formatului. Întoarce numărul de caractere scrise sau valoare negativă în caz de eroare. Se folosește cu fișiere text.
`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 citite efectiv. Se folosește cu fișiere binare.
`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 scrise efectiv. Se folosește cu fișiere binare.
`int fseek(FILE* pf, long depl, int orig);`	Poziționează cursorul în fișier la `depl` octeți față de început, poziția curentă sau sfârșitul fișierului.
`void rewind(FILE* pf);`	Poziționează 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.

Fişiere binare - Probleme rezolvate

R7_1. Să se scrie un program pentru crearea unui fişier binar, având articole structuri cu următoarele câmpuri:

- Nume depunător – şir de maxim 30 de caractere
- Data depunerii – o structură având câmpurile întregi: zi, lună, an.
- Suma depusă – o valoare reală.

Articolele sunt grupate pe zile în ordine cronologică. Datele se introduc de la consolă, fiecare pe trei linii. Să se afişeze apoi conţinutul fişierului.
(*Probleme înrudite:* **P7_1**)

Rezolvare: Vom introduce mai întâi datele într-o ordine aleatoare, apoi le vom sorta după dată. În final, vectorul de structuri va fi scris în fişierul `output.dat`.

7_1.cpp

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

void main (void) {
  // Cream un vector de articole pentru a le
  // sorta ulterior dupa data depunerii.
  // Articolul suplimentar este necesar la sortare.
  // In plus, vom citi ceea ce am scris in fisier
  // intr-un vector separat de articole.
  struct {
    char nume[30];
    struct {
      int zi, luna, an;
    } data;
    double suma;
  } articole[20], articol, citite[20];
  FILE *f;
  int n;                        // Numarul de articole
  int found;                    // Folosit la sortare
  // Citire date de intrare
  printf(" Introduceti numarul de articole: ");
  scanf ("%d", &n);
  for (int i = 0; i < n; i++) {
    printf (" Introduceti numele depunatorului: ");
    scanf ("%s", articole[i].nume);
    printf (" Introduceti data depunerii <ZZ/LL/AAAA>: ");
    scanf ("%d/%d/%d", &articole[i].data.zi, \
           &articole[i].data.luna, &articole[i].data.an);
```



```
    printf (" Introduceti suma depusa: ");
    scanf ("%lf", &articole[i].suma);
  }
  // Sortam dupa data depunerii
  do {
    found = 0;
    for (i = 0; i < n-1; i++)
      // Testam cazurile in care data articolului curent este
      // 'mai mare' decat data articolului urmator
      if ( (articole[i].data.an > articole[i+1].data.an) || \
           (articole[i].data.an == articole[i+1].data.an && \
            articole[i].data.luna > articole[i+1].data.luna)|| \
          (articole[i].data.an == articole[i+1].data.an && \
           articole[i].data.luna == articole[i+1].data.luna && \
           articole[i].data.zi > articole[i+1].data.zi) ) {
        articol = articole[i];
        articole[i] = articole[i+1];
        articole[i+1] = articol;
        found = 1;
      }
  } while (found);
  // Deschidere fisier
  if ( (f = fopen ("output.dat","wb")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
  // Scriem in fisier si il inchidem
  // Scriem intai numarul de articole
  fwrite (&n, sizeof(int), 1, f);
  fwrite (&articole, sizeof(articol), n, f);
  fclose (f);
  if ( (f = fopen ("output.dat","rb")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
  fread (&n, sizeof(int), 1, f);
  fread (&citite, sizeof(articol), n, f);
  for (i = 0; i < n; i++)
    printf (" Articolul %d: %s, %d/%d/%d, %lf\n", i+1, \
            citite[i].nume, citite[i].data.zi, \
            citite[i].data.luna, citite[i].data.an, \
            citite[i].suma);
  fclose (f);
  getch();
}
```

R7_2. Să se scrie un program pentru actualizarea fişierului creat în problema anterioară, pe baza unor foi de restituire, introduse de la tastatură, conţinând numele şi suma solicitată. Programul semnalează la consolă următoarele situaţii:
- Solicitant inexistent
- Suma solicitată depăşeşte soldul.

Fişierul actualizat este listat la consolă.

Rezolvare: Pentru simplificare am introdus ca primă informaţie din fişier numărul de articole. Astfel, la fiecare foaie de restituire ne vom poziţiona la începutul datelor relevante din fişier (adică după primul câmp ce conţine numărul de articole) şi vom face o căutare în fişier după numele solicitantului. Dacă acesta nu este găsit, înseamnă că nu se află în baza de date. Dacă este găsit şi cere o sumă mai mare decât cea pe care o are depusă, se va întoarce de asemenea un mesaj de eroare. Altfel, se actualizează baza de date.

7_2.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

void main (void) {
  // Vom citi ceea ce am scris in fisier
  // intr-un vector de articole (pentru a testa
  // ca actualizarea s-a facut corect.
  struct {
    char nume[30];
    struct {
      int zi, luna, an;
    } data;
    double suma;
  } articol, citite[20];

  FILE *f;
  double suma;
  char nume[30];
  char cont = 'D';            // Pentru continuare
  int n;                      // Numarul de articole

  // Deschidere fisier
  if ( (f = fopen ("output.dat","r+b")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
```



```
  // Actualizare
  while (toupper(cont) == 'D') {
    printf (" Foaie de restituire. \n");
    printf (" Introduceti numele: "); scanf ("%s", nume);
    printf (" Introduceti suma: "); scanf ("%lf", &suma);
    // Ne pozitionam...luam in calcul pe n = nr de articole
    fseek (f, sizeof(int), 0);
    articol.suma = -1.0;
    while (!feof(f) && strcmp(articol.nume, nume) != 0)
      fread (&articol, sizeof(articol), 1, f);
    if (!feof(f))
      if (suma > articol.suma)
      printf (" Suma este prea mare! \n");
      else
      {
      articol.suma -= suma;
      fseek (f, ftell(f)-sizeof(articol), 0);
      fwrite (&articol, sizeof(articol), 1, f);
      }
    flushall();
    printf (" Doriti sa continuati? <D/N> ");
    scanf ("%c", &cont);
  }
  fseek (f, 0, 0);
  fread (&n, sizeof(int), 1, f);
  fread (&citite, sizeof(articol), n, f);
  for (int i = 0; i < n; i++)
    printf (" Articolul %d: %s, %d/%d/%d, %lf\n", i+1, \
              citite[i].nume, citite[i].data.zi, \
              citite[i].data.luna, citite[i].data.an, \
              citite[i].suma);
  fclose (f);
  getch();
}
```

R7_3. Pentru a sorta elementele unui tablou V, fără a le deplasa, se crează un nou tablou P, în care un element P_i reprezintă poziţia pe care ar avea-o elementul corespunzător din V în tabloul sortat, adică numărul de elemente care ar trebui să se găsească înaintea fiecărui element din tabloul sortat: $P_i = \text{număr}(V_j \le V_i)$, $0 \le j \le n-1;\ j \ne i$

De exemplu:

I	0	1	2	3	4
V	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 V. Cel mai mic element se află în V în poziția k, astfel încât $P_k = 0$, următorul – în poziția corespunzătoare lui $P_k = 1$, ultimul element corespunde poziției k pentru care $P_k = n - 1$.

Dacă tabloul V nu are toate elementele distincte, pentru crearea tabloului P se face modificarea: $P_i = \text{num}\underset{j<i}{\text{ă}}\text{rul}(V_j \le V_i) + \text{num}\underset{j>i}{\text{ă}}\text{rul}(V_j < V_i)$

De exemplu:

	0	1	2	3	4	5	6	7	8
V	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 V avem un fișier cu tip. Sortarea fișierului în raport cu o cheie (unul din câmpurile articolelor fișierului) revine la crearea 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, crează 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.

Rezolvare: Vom genera vectorii ind (P din exemplul de mai sus) și x (ca în exemplul de mai sus). Apoi imediat, soluțiile sunt date de $v[x[i]]$.

Fișierul binar ce conține vectorul v va fi dat ca parametru în linia de comandă.

7_3.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <alloc.h>

// Scrie fisierul index pe baza fisierului de intrare
FILE *index (FILE *f_in) {
  int n, *v, *ind;
  FILE *f_index;
  int lt;       //  Numarul elementelor din vectorul initial care
                //  sunt mai mici decat elementul curent.
  int i, j;     // Indecsi
```



```
  // Citire fisier de intrare
  fread (&n, sizeof(int), 1, f_in); // Numarul de elemente din v
  v = (int *) malloc (n * sizeof(int));
  ind = (int *) malloc (n * sizeof(int));
  fread (v, sizeof(int), n, f_in);
  rewind (f_in);
  // Calculam vectorul index
  for (i = 0; i < n; i++) {
    lt = 0;
    for (j = 0; j < n; j++)
      if (j == i);
      else {
        if (j < i && v[i] >= v[j])
          lt += 1;
        else
          if (v[i] > v[j])
            lt += 1;
      }
    ind[i] = lt;
  }
  // Scriem fisierul index
  if ( (f_index = fopen ("index.dat","w+b")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (2);
  }
  fwrite (&n, sizeof(int), 1, f_index);
  fwrite (ind, sizeof(int), n, f_index);
  rewind (f_index);
  return f_index;
}

// Sorteaza vectorul din fisierul f_in folosind fisierul f_index
void sort (FILE *f_in, FILE *f_index) {
  int *v, *ind, *x;
  int n, i, j;
  int aux;
  // Citire fisiere de intrare
  fread (&n, sizeof(int), 1, f_in);
  v = (int *) malloc (n * sizeof(int));
  fread (v, sizeof(int), n, f_in);
  fread (&n, sizeof(int), 1, f_index);
  ind = (int *) malloc (n * sizeof(int));
  fread (ind, sizeof(int), n, f_index);
  // Cream vectorul x
  x = (int *) malloc (n * sizeof(int));
```

```
  for (i = 0; i < n; i++) {
    aux = 0;
    while (ind[aux] != i) aux++;
    x[i] = aux;
  }
  // Afisam vectorul sortat
  printf (" Vectorul sortat este: \n");
  for (i = 0; i < n; i++)
    printf (" %d", v[x[i]]);
}

void main (int argc, char *argv[]) {
  FILE *f_in, *f_index;
  int n, k;
  // Deschidere fisiere
  if (argc != 2) {
    printf (" Utilizare: 7_3.exe <fisier_de_intrare> \n");
    exit (1);
  }
  if ( (f_in = fopen (argv[1],"rb")) == NULL )    {
    printf (" Eroare la deschiderea fisierului! ");
    exit (2);
  }
  // Prelucrarea datelor
  f_index = index (f_in);
  sort (f_in, f_index);
  fclose (f_in);
  fclose (f_index);
  getch();
}
```

R7_4. Se consideră dat fişierul de întregi `numere.dat`. Să se creeze fişierul `fibopr.dat`, conţinând numai acele elemente din fişierul iniţial care sunt numere din şirul lui Fibonacci, numere prime.

Rezolvare: Vom folosi trei funcţii: `fib` (calculează termenul *n* din şirul lui Fibonacci), `test_fib` (verifică dacă numărul *n* aparţine şirului lui Fibonacci) şi `prime` (verifică dacă numărul *n* este prim).

Ideea este simplă: se citeşte câte un număr din fişierul `numere.dat` şi se verifică dacă este prim şi dacă aparţine şirului lui Fibonacci. În caz afirmativ, se scrie numărul în fişierul `fibopr.dat`.

7_4.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

// Intoarce termenul n din sirul lui Fibonacci
int fib (int n) {
  if (n == 0 || n == 1) return 1;
  return (fib(n-1) + fib(n-2));
}

// Verifica daca numarul n este din sirul lui Fibonacci
int test_fib (int n) {
  int i = 0;
  while (fib(i++) < n);
  return (fib(i-1) == n);
}

// Verifica daca numarul n este prim
int prime (int n) {
  for (int i = 2; i < n/2; i++)
    if (n % i == 0) return 0;
  return 1;
}

void main (void) {
  FILE *f_in, *f_out;
  int n, k;
  // Deschidere fisiere
  if ( (f_in = fopen ("numere.dat","rb")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
  if ( (f_out = fopen ("fibopr.dat","wb")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
  while (!feof(f_in)) {
    fread (&k, sizeof(int), 1, f_in);
    if (prime(k) && test_fib(k))
      fwrite (&k, sizeof(int), 1, f_out);
  }
  fclose (f_in);
  fclose (f_out);
}
```

R7_5. Pentru stabilirea câştigătorilor la concursul: *PRO ştii şi câştigi la telefon* se folosesc două fişiere: `concurenti` (conţinând articole cu câmpurile `nume` – şir de caractere şi `telefon` – şir de caractere) şi *premii* (câmpuri `telefon` – şir de caractere şi `premiu` – valoare reală).
Scrieţi un program care afişează numele câştigătorilor şi valoarea premiilor. Se vor afişa de asemenea: valoarea totală a câştigurilor, premiul maxim şi numele celui care l-a câştigat.

Rezolvare: Nu trebuie decât să căutăm numerele de telefon din fişierul `conc.dat` în fişierul `premii.dat` şi să luăm în considerare câştigul (suma câştigată şi numele câştigătorului). Vom presupune că aceste fişiere sunt deja scrise. De asemenea, vom reţine în variabila `max_prize` valoarea premiului maxim. Aceasta va fi iniţializată cu o valoare negativă oarecare (spre exemplu –1).

7_5.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

typedef struct contestant {
    char name[30], phone[30];
  } contestant;
typedef struct prize {
    char phone[30];
    double value;
  } prize;

void main (void) {
  FILE *f_conc, *f_prizes;
  double sum = 0.0;           // Suma premiilor
  double max_prize = - 1.0;   // Premiul maxim
  char winner[30];            // Castigatorul premiului maxim
  contestant cont;
  prize prz;
  // Presupunem ca nu au doua persoane acelasi telefon
  int found;
  clrscr();
  // Deschidere fisiere
  if ( (f_conc = fopen ("conc.dat","rb")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
  if ( (f_prizes = fopen ("premii.dat","rb")) == NULL ) {
    printf (" Eroare la deschiderea fisierului! ");
    exit (1);
  }
  while (!feof(f_prizes)) {
    fread (&prz, sizeof(prize), 1, f_prizes);
    rewind (f_conc);
    found = 0;
    while (!feof(f_conc) && !found) {
      fread (&cont, sizeof(contestant), 1, f_conc);
      if (strcmp(cont.phone, prz.phone) == 0) {
        found = 1;
        printf (" %s a castigat %lf\n", cont.name, prz.value);
        sum += prz.value;
        if (max_prize < prz.value) {
          max_prize = prz.value;
          strcpy (winner, cont.name);
        }
      }
    }
  }
  printf (" %s a castigat %lf, adica cel mai mult\n", winner,
           max_prize);
  printf (" Suma premiilor este: %lf\n", sum);
  fclose (f_conc);
  fclose (f_prizes);
}
```


R7_6. Abonaţii unei companii de telefoane sunt reţinuţi în fişierul `abonati.dat` ce conţine înregistrări de forma:
- nume abonat (şir de 25 de caractere)
- număr de telefon (şir de 25 de caractere)

Există de asemenea un fişier binar `plati.dat` ce conţine câte un articol pentru fiecare chitanţă de achitare a taxei telefonice sub forma:
- număr de telefon (şir de 10 caractere)
- suma plătită (real).

Să se scrie în C:
- o funcţie pentru căutarea unui abonat în fişierul `plati.dat`, care întoarce rezultatul 1 (există) / 0 (nu există)
- o funcţie `main()` pentru tipărirea la imprimantă a numelor abonaţilor care nu şi-au achitat taxa telefonică.

Rezolvare: Se citesc informațiile din fișierul "abonati.dat" una câte una și se verifică dacă există o înregistrare corespunzătoare în fișierul de plăți, adică dacă abonat.tel=plata.tel.

Imprimanta este privită în C ca un fișier de tip text (stdprn). Tipărirea la imprimantă se realizează folosind funcția: fprintf(stderr, ...);

7_6.cpp

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define FISIER_ABONATI "abonati.dat"
#define FISIER_PLATI   "plati.dat"

struct PLATA {
  char tel[10];
  float suma;
} ;
struct ABONAT {
  char nume[25];
  char tel[10];
} ;

// Cauta in fisierul de plati dupa campul tel
int cauta(char *tel, char *fisier) {
  FILE *f;
  struct PLATA articol;
  if (!(f=fopen(FISIER_ABONATI,"rb"))) {
    printf("Nu pot deschide fisierul de abonati!");
    exit(-1);
  }
  while (fread(&articol,sizeof(articol),1,f)>0)
    if (strcmp(articol.tel,tel)==0) {
      fclose(f);
      return 1;
    }
  fclose(f);
  return 0;
}

int main(void) {
  struct ABONAT abonat;
  FILE *f;
  // Deschid fisierul de abonati ("rb" = READ BINARY)
  if (!(f=fopen(FISIER_ABONATI,"rb"))) {
    printf("Nu pot deschide fisierul de plati!");
    exit(-1);
  }
```



```
  // Citesc inregistrarile referitoare la abonati una cate una
  // si caut in fisierul de plati
  while (fread(&abonat,sizeof(abonat),1,f)>0)
    if (!cauta(abonat.tel,FISIER_PLATI))
      fprintf(stdout,"%s\n",abonat.nume);
  fclose(f);
  return 1;
}
```

R7_7. Un fișier binar nrlinii.dat conține întregi reprezentând numerele unor linii dintr-un fișier text text.txt. Știind că acești întregi sunt ordonați crescător, scrieți un program care afișează la consolă liniile din fișierul text a căror numere apar în fișierul binar.

Rezolvare:

7_7.cpp

```
#include <stdio.h>
#include <conio.h>

int main(void) {
  clrscr();
  FILE *b, *t;
  int linie_afisata,linie_curenta=0;
  char linie[1000];
  // Deschid fisierele
  t=fopen("text.txt","rt");
  b=fopen("nrlinii.dat","rb");
  if (t==NULL || b==NULL) {
    printf("Nu pot deschide unul din fisiere.");
    return -1;
  }
  while (1) {
    // Citesc o noua informatie din fisierul binar
    // totodata testez daca citirea s-a facut bine
    // si daca nu am ajuns la sfarsitul fisierului
    if (fread(&linie_afisata,sizeof(int),1,b)<=0)
      break;
    // citesc linii din fisierul text pana cand ma pozitionez
pe
    // linia ca trebuie sa o afisez
    while (linie_afisata>linie_curenta) {
      linie_curenta++;
      fgets(linie,1000,t);
    }
    fputs(linie,stdout);
  }
  fclose(t);
  fclose(b);
  return 0;
}
```

R7_8. Să se scrie un program care primeşte date numere pare şi afişează descompunerile distincte ale fiecărui număr par ca o sumă de două numere prime. Datele pot fi introduse în 3 moduri:

a) interactiv, caz în care linia de comandă nu are parametri, iar numerele se introduc de la tastatură, câte un număr pe o linie. Numerele se termină prin marcajul de sfârşit de fişier.
b) numerele se dau ca parametri ai liniei de comandă
c) numerele se citesc dintr-un fişier binar, caz în care comanda are 2 parametri: `-f` şi numele fişierului (Ex. "`-fINPUT.DAT`").

Rezolvare: Citirea numerelor în cele 3 variante se face astfel:

- Dacă numărul de argumente din linia de comandă (`argc`) este egal cu 1, numerele se citesc de la tastatură. Un program C/C++ primeşte întotdeauna cel puţin un argument (numele programului executabil).
- Dacă `argc = 2` suntem în una din situaţiile:
  a) Citirea se face din fişier. În acest caz argumentul este de forma: "`fNume Fisier`", deci al doilea caracter al argumentului este '`f`'
  b) Am primit ca argument un singur număr
- Dacă `argc > 2`, se citesc argumentele liniei de comandă şi se convertesc la tipul întreg (folosind funcţia `atoi`).

Pentru a scrie un număr ca sumă a 2 numere prime se caută toate descompunerile distincte de forma $nr = i + (nr - i)$ şi se verifică dacă i, respectiv $nr - i$ sunt numere prime.

7_8.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define BOOL  unsigned char
#define FALSE 0
#define TRUE  1

// Determina daca numarul n este prim sau nu
BOOL prim(int n) {
  for (int i=2;i<floor(sqrt(n))+1;i++)
    if (n % i == 0)
      return FALSE;
  return TRUE;
}

// Afiseaza descompunerea numarului nr ca suma de 2 numere prime
```



```
void descompunere(int nr) {
  for (int i=2; i<nr; i++)
    if (prim(i) && prim (nr-i)) {
      printf("%d=%d+%d\n",nr,i,nr-i);
      return;
    }

}

// argc - numarul de argumente din linia de comanda
// argv - vector de siruri de caractere ce contine argumentele
int main(int argc,char **argv) {
  FILE *f;
  char nume[100]; // Numele fisierului de intrare
  int nr;
  switch (argc) {
    case 1: // linia de comanda nu contine argumente
      while (scanf("%d",&nr)!=EOF)
        descompunere(nr);
      break;
    case 2: // linia de comanda contine un singur argument
      if (argv[1][1]=='f') { // numerele sunt citite din fisier
        sscanf(argv[1],"-f%s",nume);
        if (!(f=fopen(nume,"rb"))) {
          printf("Nu pot deschide fiserul %s\n",nume);
          return -1;
        }

        while (fread(&nr,sizeof(nr),1,f)>0)
          descompunere(nr);

        fclose(f);
        break;
      }
    default: // numerele sunt citite din linia de comanda
      for (int i=1;i<argc;i++) {
        nr=atoi(argv[i]);
        descompunere(nr);
      }
  }
  return 0;
}
```

R7_9. Să se scrie un program care primeşte date numere naturale şi găseşte, pentru fiecare număr, numerele prime cele mai apropiate de acesta. Datele se citesc de pe mediul de intrare şi sunt scrise într-un fişier binar ce conţine înregistrări de forma:

```
int numar; // Numărul citit
int a;     // Cel mai apropiat număr prim mai mic decât numărul citit
int b;     // Cel mai apropiat număr prim mai mare decât numărul citit
```

În cazul în care numărul citit este prim, înregistrarea corespunzătoare din fişier va avea `numar = a = b`.

Rezolvare:

7_9.cpp

```
#include <stdio.h>
#include <math.h>

#define BOOL  unsigned char
#define FALSE 0
#define TRUE  1

struct INREG {
  int numar;
  int a;
  int b;
};

// Determina daca numarul n este prim sau nu
BOOL prim(int n) {
  for (int i=2;i<floor(sqrt(n))+1;i++)
      if (n % i == 0)
        return FALSE;
  return TRUE;
}

/* Gaseste cele mai apropiate numere prime si returneaza o
 * structura de tipul INREG.
 */
struct INREG gaseste(int nr) {
  struct INREG inregistrare;
  int i;
  inregistrare.numar=nr;
  if (prim(nr)) { // Verific daca numarul este prim
      inregistrare.a=nr;
      inregistrare.b=nr;
      return inregistrare;
  }
```



```
  // Determin cel mai apropiat numar prim mai mic decat nr
  for (i=nr-1;;i--)
      if (prim(i)) {
        inregistrare.a=i;
        break;
      }
  // Determin cel mai apropiat numar prim mai mare decat nr
  for (i=nr+1;;i++)
      if (prim(i)) {
        inregistrare.b=i;
        break;
      }
  return inregistrare;
}

void main(void) {
  FILE *out;
  int nr;
  struct INREG inregistrare;
  if (!(out=fopen("OUTPUT.BIN","wb"))) {
      printf("Nu pot crea fisierul de iesire\n");
      return;
  }
  while (scanf("%d",&nr)!=EOF) {
      inregistrare=gaseste(nr);
      fwrite(&inregistrare,sizeof(INREG),1,out);
  }
  fclose(out);
}
```

R7_10. Se dă un fişier binar `numere.bin` care conţine numere naturale. Să se scrie un program care înlocuieşte numerele din acest fişier cu răsturnatele lor.

Exemplu: Dacă fişierul conţine numerele: 102, 5, 0, 123, 100, la terminarea programului fişierul va avea următorul conţinut: 201, 5, 0, 321, 1.

Rezolvare: Există 2 variante pentru a modifica înregistrările dintr-un fişier:

a) Cu fişier temporar:

- se creează un nou fişier pe baza fişierului dat (cu modificările necesare)
- se şterge fişierul original (folosind funcţia `remove`)
- se redenumeşte fişierul temporar cu numele fişierului original (folosind funcţia `rename`)

b) Se deschide fişierul cu "r+b" (citire/scriere în mod binar) şi se suprascriu înregistrările care se doresc modificate.

Problema este rezolvată folosind cea de-a doua variantă.

7_10.cpp

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

// Calculeaza rasturnatul unui numar
int rasturnat(int nr) {
  int rasturnat=0;
  while (nr>0) {
    rasturnat+=nr%10;
    rasturnat*=10;
    nr/=10;
  }
  return rasturnat/10;
}
void main(void) {
  FILE *f;
  int nr,i=0;
  // Deschid fisierul pentru operatii READ/WRITE in mod binar
  if (!(f=fopen("NUMERE.BIN","r+b"))) {
      printf("Nu pot deschide fisierul \n");
      return ;
  }
  while (fread(&nr,sizeof(int),1,f)>0) {
      nr=rasturnat(nr);
      // Ma pozitionez pe inregistrarea anterioara
      // si scriu noa valoare
      fseek(f,i*sizeof(int),SEEK_SET);
      fwrite(&nr,sizeof(int),1,f);
      // Ma pozitionez pentru a citi o noua valoare
      i++;
      fseek(f,i*sizeof(int),SEEK_SET);
  }
  fclose(f);
}
```


R7_11. Un fişier conţine articole cu structura: `cod` (şir de caractere), `nume` (şir de caractere) şi `cantitate` (real).
Să se scrie un program care permite:
- creare
- consultare
- ştergere.

Opţiunea se introduce ca argument al liniei de comandă, prin numele complet al operaţiei:
- **Creare Nume_fisier** – Se creează fişierul `Nume_fisier` prin citirea de la tastatură a articolelor. După fiecare linie introdusă se interoghează dacă se continuă.
- **Consultare Nume_fisier** – Conţinutul fişierului `Nume_fisier` este listat pe ecran, câte 20 de linii. Pentru continuarea afişării se apasă ENTER.
- **Ştergere Nume_fisier cod_articol** – Este căutat articolul în fişier şi se marchează punându-i codul `xxx`

Rezolvare:

7_11.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#define FALSE 0
#define TRUE  1
struct ARTICOL {
  char cod[10];
  char nume[30];
  float cantitate;
};

// Executa functia de creare fisier
void creare(char *fisier) {
  FILE *f;
  struct ARTICOL a;
  // Deschid fisierul pentru scriere in mod binar
  if ((f=fopen(fisier,"wb")) == NULL)
  {
      printf (" Nu pot deschide fisierul. \n");
      exit (1);
  }
```

```
  do {
      //Articolele citite unul cate unul si scrise in fisier
      printf (" Introduceti codul, numele si cantitatea: ");
      scanf("%s %s %f",&a.cod,&a.nume,&a.cantitate);
      fwrite(&a,sizeof(ARTICOL),1,f);
      printf("Doriti sa continuati ? (d/n)\n");
  } while (getch()=='d');
  fclose(f);
}

// Executa functia de consultare fisier
void consultare(char *fisier) {
  FILE *f;
  struct ARTICOL a;
  int i=1;
  // Deschid fisierul pentru citire in mod binar
  if ((f=fopen(fisier,"rb")) == NULL)
  {
      printf (" Nu pot deschide fisierul. \n");
      exit (1);
  }
  while (fread(&a,sizeof(ARTICOL),1,f)>0)
    // Am grija sa nu afisez articolele care au fost sterse
    if (strcmp(a.cod,"xxx")!=0) {
      printf("%10s %30s %10.2f\n",a.cod,a.nume,a.cantitate);
      // Daca am afisat 20 de articole, ma opresc si intreb
      if (i++%20==0) {
        printf("Doriti sa continuati ? (d/n)\n");
        if (getch()=='n')
          break;
      }
    }
  fclose(f);
}

// Executa functia de stergere
void stergere(char *fisier,char *cod) {
  FILE *f;
  struct ARTICOL a;
  int gasit=FALSE;
  // Fisierul este deschis in pentru citire/scriere in mod binar
  f=fopen(fisier,"r+b");
  while (fread(&a,sizeof(ARTICOL),1,f)>0)
    if (strcmp(a.cod,cod)==0) {
      // Marchez articolul ca fiind sters
      strcpy(a.cod,"xxx");
```



```
      // Ma pozitionez cu o inregistrare inainte pentru a
      // suprascrie articolul
      fseek(f,ftell(f)-sizeof(ARTICOL),SEEK_SET);
      fwrite(&a,sizeof(ARTICOL),1,f);
      gasit=TRUE;
      break;
    }
  if (!gasit)
    printf("Eroare: Articolul nu a fost gasit\n");
  fclose(f);
}

void main(int argc,char **argv) {
  if (strcmp(argv[1],"Creare")==0)
    creare(argv[2]);
  if (strcmp(argv[1],"Consultare")==0)
    consultare(argv[2]);
  if (strcmp(argv[1],"Stergere")==0)
    stergere(argv[2],argv[3]);
}
```

Fişiere text - Probleme rezolvate

R7_12. Dintr-un fişier text `text.txt` se separă toate cuvintele, plasându-le într-un fişier binar `cuv.dat`, având ca articole şiruri de 10 caractere (cuvintele mai scurte se completează cu spaţii, iar cele mai lungi se trunchiază la primele 10 caractere).

Rezolvare:

7_12.cpp

```
#include <stdio.h>
#include <string.h>
#define MAX 1000 // Numarul maxim de caractere de pe o linie
void main(void) {
  FILE *in,*out;
  char linie[MAX],cuv[12],*p;
  char desp[]=" ;,.\n\t\0"; // despartitorii de cuvinte
  int l; // lungimea cuvantului
  // Deschid cele 2 fisiere
  in=fopen("TEXT.TXT","rt");
  out=fopen("CUV.DAT","wb");
  if (!in || !out) {
```

```
    printf("Nu pot deschide unul din fisiere\n");
    return ;
  }
  while (!feof(in)) {
    // Citesc fisierul de intrare linie cu linie folosind fgets
    fgets(linie,MAX,in);
    p=linie;
    do {
      // Formez cuvintele
      l=0;
      for(;strchr(desp,*p)==NULL;p++)
        if (l<10)
          cuv[l++]=*p;
      // Completez cu spatii pana la 10 caractere
      memset(cuv+l,' ',10-l);
      // Pun terminatorul de sir de caractere
      cuv[10]=0;
      // Verific daca este cuvamt format numai din caractere
      // despartitoare (acest lucru se intampla daca exista 2
      // caractere despartitoare consecutive) apoi scriu in
      // fisierul binar
      if (strchr(desp,cuv[0])==NULL)
        fwrite(cuv,11,1,out);
      p++;
    } while (strlen(linie)>p-linie);
  }
  fclose(out);
  fclose(in);
}
```

R7_13. Se dă un fişier text.

a) Să se determine numărul de linii din fişier.

b) Să se creeze un nou fişier cu liniile din primul, apărând în ordine inversă.

c) Pe baza fişierului iniţial, să se creeze un fişier de caractere, în care nu mai apar caracterele de sfârşit de linie, iar fiecare linie este precedată de lungimea ei (un octet).

d) Se dă un fişier de întregi reprezentând numere de linii din fişierul iniţial. Să se afişeze la imprimantă liniile din fişier în ordinea precizată de fişierul de întregi.

Rezolvare: Vom scrie o funcţie (`calc_pozitii_linii()`) care, pentru fisierul dat ca parametru, întoarce un vector în care se reţin poziţiile de început în cadrul fişierului ale tuturor liniilor acestuia. Deoarece fişierul poate fi foarte mic sau foarte mare, memoria pentru vector se alocă dinamic, incremental. Pentru a afişa liniile în ordine inversă, ne folosim de poziţiile lor de început, memorate în vector.

Numărul de linii este determinat tot în cadrul funcţiei de mai sus (este reţinut ca parametru transmis prin referinţă).

7_13.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <assert.h>

#define INC 10   // increment pentru alocarea memoriei
#define LMAX 81  // lungimea maxima a unei linii + 1

/* afisarea unui mesaj de eroare si iesirea din program: */
void eroare(char *s) {
  perror(s);
  exit(-1);
}

/* calcularea pozitiilor de inceput ale liniilor din fisierul f:
 * n - numarul de linii, care este de asemenea calculate  */
int* calc_pozitii_linii(FILE *f, int& n) {
  int i;
  int *vpoz;  // vector in care se retin pozitiile de inceput
  char s[81];  // linia curenta
  vpoz = (int*) malloc(INC * sizeof(int));
  fseek(f, 0, SEEK_SET);
  i = 0;
  vpoz[0] = 0;
  while (fgets(s, LMAX, f)) {
    if (i > 0 && i % INC == 0)
    /* memoria se realoca incremental: */
      realloc(vpoz, (i+INC) * sizeof(int));
    vpoz[++i] = ftell(f);
  }
  n = i;
  return vpoz;
}
```

```
void main(void) {
  FILE *f, *fn, *fres1, *fres2;
  char nume[20];  // nume pentru fisiere
  char s[LMAX];  // linia curenta din fisier
  int n;  // numarul de linii din fisierul de intrare
  int *vpoz;  // vectorul pozitiilor de inceput ale liniilor
  int i;
  char lung;  // lungimea unei linii
  printf("Numele fisierului de intrare: ");
  scanf("%s", nume);
  f  = fopen(nume, "rt");
  if (f == NULL) eroare("fopen");
  fres1  = fopen("rez1.txt", "wt"); // fisierul cu linii in
ordine inversa
  if (fres1 == NULL) eroare("fopen");
  fres2  = fopen("rez2.txt", "wt"); // fisierul de caractere
  if (fres2 == NULL) eroare("fopen");
  /* (a) NUMARUL DE LINII DIN FISIER: */
  vpoz = calc_pozitii_linii(f, n);
  printf("Fisierul are %d linii \n", n);
  /* (b) FISIERUL CU LINII IN ORDINE INVERSA: */
  for (i = n - 1; i >= 0; i--) {
    fseek(f, vpoz[i], SEEK_SET);
    fgets(s, LMAX, f);
    fputs(s, fres1);
    /* daca dupa ultima linie nu exista '\n', se adauga in
       fisierul rezultat:
     */
    if (i == n - 1 && s[strlen(s) - 1] != '\n')
      fputs("\n", fres1);
  }
  fclose(fres1);
  rewind(f);
  /* (c) FISIERUL DE CARACTERE: */
  while (fgets(s, LMAX, f)) {
    lung = strlen(s) - 1; // ignoram caracterul '\n'
    fputc(lung + '0', fres2);
    for (i = 0; i < strlen(s) - 1; i++)
      fputc(s[i], fres2);
  }
  fclose(fres2);
  /* (d) LISTAREA LA IMPRIMANTA: */
  printf("Numele fisierului cu numere de linii : ");
  scanf("%s", nume);
  fn  = fopen(nume, "rt");
  if (fn == NULL) eroare("fopen");
  while (fscanf(fn,"%d ",&i) != EOF) {
```



```
    assert(i > 0 && i <= n);
    // in fisier, numerele liniilor incep de la 1 => folosim
    // vpoz[i - 1]:
    fseek(f, vpoz[i - 1], SEEK_SET);
    fgets(s, LMAX, f);
    fprintf(stdprn, "%s", s);
  }
  fclose(fn);
  fclose(f);
  getch();
}
```

R7_14. Fişierul text `prog.c` reprezintă un program sursă C. Să se copieze acest fişier la ieşirea standard suprimând toate comentariile.

Rezolvare: Vom citi fişierul de intrare caracter cu caracter şi, când întâlnim un posibil început de comentariu (caracterul '/'), analizăm şi caracterul următor:

- Dacă acesta este '*', aveam într-adevăr un început de comentariu şi nu vom mai copia cele două caractere la ieşire.
- Altfel, copiem la ieşire '/' –ul şi "punem înapoi" în fluxul de intrare cel de-al doilea caracter citit, pentru a se relua analiza începând de la el (acest lucru este necesar deoarece putem avea o secvenţă de genul '//*', în care caracterul de după primul '/' chiar este început de comentariu şi trebuie luat în considerare; "punerea înapoi" se realizează cu funcţia `putback()`).

Sfârşitul de comentariu se detectează într-un mod asemănător celui de mai sus.

7_14.cpp

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void) {
  fstream f;
  char nume[20];  // numele fisierului
  char c1, c2;  // caracterele care se citesc din fisier
  cout << "Numele fisierului :";
  cin >> nume;
  f.open(nume, ios::in);
  if (!f) {
    cout << "Fisierul nu s-a putut deschide";
    exit(-1);
  }
  while (1) {
```

```
    /* citim din fisier pana intalnim '/' sau pana se termina
       fisierul: */
    while (f.get(c1))
      if (c1 != '/') cout << c1;
      else break;
    if (f.eof()) break;  // fisierul s-a terminat
    /* am intalnit '/', analizam caracterul urmator: */
    f.get(c2);
    if (c2 == '*') {  // inceput de comentariu
      while (1) {
        while (f.get(c1))
          if (c1 == '*') break; //am ajuns la un posibil
                                //terminator de comentariu
        if (f.eof()) break;
        /* am intalnit '*', analizam caracterul urmator: */
        f.get(c2);
        if (c2 == '/') // comentariul s-a terminat
          break;
        else  // nu fusese inceput de comentariu
          f.putback(c2);
      }
    }
    else { // nu era inceput de comentariu, afisam c1
      cout << c1;
      f.putback(c2);
    }
  }
  f.close();
}
```

R7_15. Un fişier text, cu numele introdus de la tastatură are următoarea structură:

```
...
linii
#R nume_fişier
linii
...
```

Să se creeze un nou fişier, din fişierul dat, care în locul liniei specificate va avea inserat fişierul numit.

Rezolvare: Parcurgem fişierul de intrare linie cu linie şi, când întâlnim o linie care începe cu `"#R"`, determinăm numele fişierului care trebuie inserat (presupunem că acesta începe exact în a patra poziţie a liniei; dacă nu am fi făcut această presupunere, am fi putut folosi `sscanf` sau `strtok` - şi atunci numele ar fi putut avea oricâte spaţii înainte). Inserarea fişierului se face tot "linie cu linie".

7_15.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define LMAX 81  // lungimea maxima a unei linii + 1

/* afisarea unui mesaj de eroare si iesirea din program: */
void eroare(char *s) {
  perror(s);
  exit(-1);
}

void main(void) {
  FILE *f, *fres; // fisierele de intrare si de iesire
  FILE *fins;  // fisierul de inserat
  char nume[20];  // nume pentru fisiere
  char s[LMAX], s2[LMAX];  // liniile curente din fisiere
  printf("Numele fisierului de intrare: ");
  scanf("%s", nume);
  f  = fopen(nume, "rt");
  if (f == NULL) eroare("fopen");
  printf("Numele fisierului de iesire: ");
  scanf("%s", nume);
  fres  = fopen(nume, "wt");
  if (fres == NULL) eroare("fopen");
  /* parcurgem fisierul de intrare linie cu linie: */
  while (fgets(s, LMAX, f)) {
    if (s[0] == '#' && s[1] == 'R') {
      strcpy(nume, s + 3);  // nume fisier incepe de la s[3]
      nume[strlen(nume) - 1] = 0; // "stergem" caracterul '\n'
                                  // de la sfarsit
      fins  = fopen(nume, "rt");
      if (fins == NULL) eroare("fopen");
      /* inseram fisierul: */
      while (fgets(s2, LMAX, fins))
        fputs(s2, fres);
    }
    else  // copiem linia in fisierul de iesire
      fputs(s, fres);
  }
  fclose(f);
  fclose(fres);
  fclose(fins);
}
```

R7_16. Un fişier text conţine blocuri delimitate de caracterele #B şi #K. Să se creeze din aceste blocuri fişiere cu acelaşi nume cu fişierul iniţial, având extensiile 001, 002, ...

Rezolvare: Detectarea blocurilor se face asemănător cu cea a comentariilor din programele sursă C (vezi problema de mai sus).

Pentru a genera numele fişierelor de ieşire procedăm astfel: copiem în şirul pe care trebuie să îl creăm caracterele din numele fişierului de intrare, până la '.', fără acesta (deoarece numele fişierului de intrare poate să nu aibă extensie) ; apoi adăugăm un '.' şi extensia corespunzătoare (vom folosi un contor pentru a reţine numărul fişierelor generate).

7_16.cpp

```
#include <stdio.h>
#include <fstream.h>
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

/* afisarea unui mesaj de eroare si iesirea din program: */
void eroare(char *s) {
  perror(s);
  exit(-1);
}

/* generarea numelui pentru al "nr" - lea fisier de iesire:
 * n_vechi = numele fisierului de intrare
 */
char* creeaza_nume_nou(char* n_vechi, int nr) {
  char extensie[3];
  char *n_nou;  // numele pe care il vom genera
  int i;
  n_nou = (char*) malloc((strlen(n_vechi) + 4) * sizeof(char));
  /* cream extensia: */
  sprintf(extensie, "%03d", nr);
  /* copiem in nume nou inceputul numelui vechi (fara extensie):
  */
  for (i = 0; i < strlen(n_vechi) && n_vechi[i] != '.'; i++)
    n_nou[i] = n_vechi[i];
  n_nou[i] = '.';
  n_nou[++i] = 0; // trebuie sa adaugam terminatorul de sir
  strcat(n_nou, extensie);
  return n_nou;
}
```



```
void main(void) {
  fstream f, fr; // fisierele de intrare si de iesire
  char nume[20];  // numele fisierului de intrare
  char *nume_res; // nume pentru fisierele de iesire
  int cnt = 0; // contor care numara fisierele de iesire
  char c1, c2;  // caracterele care se citesc din fisier
  cout << "Numele fisierului de intrare:";
  cin >> nume;
  f.open(nume, ios::in);
  if (!f) eroare("open");
  while (1) {
    /* citim din fisier pana intalnim '#' sau pana se termina
       fisierul: */
    while (f.get(c1))
      if (c1 == '#') break;
    if (f.eof()) break;  // fisierul s-a terminat
    /* am intalnit '#', analizam caracterul urmator: */
    f.get(c2);
    if (c2 == 'B') { // inceput de bloc
      /* initializam un nou fisier de iesire: */
      cnt++;
      nume_res = creeaza_nume_nou(nume, cnt);
      fr.open(nume_res, ios::out);
      if (!fr) eroare("open");
      /* copiem caracterele din bloc in fisierul de iesire: */
      while (1) {
        while (f.get(c1))
        if (c1 != '#')
          fr.put(c1);
          else break; //un posibil terminator de bloc
        if (f.eof()) break;
        /* am intalnit '#', analizam caracterul urmator: */
        f.get(c2);
        if (c2 == 'K') // blocul s-a terminat
        break;
        else {
          fr.put(c1);
          f.putback(c2);
        }
      }
      fr.close();
    }
    else  // nu era inceput de bloc
      f.putback(c2);
  }
  f.close();
}
```

R7_17. Ştergeţi toate apariţiile unui cuvânt dintr-un fişier text. Cuvântul care se şterge, ca şi numele fişierului se citesc de la tastatură. Programul va crea un nou fişier, care în final va primi numele fişierului iniţial, iar acesta va fi şters.

Rezolvare: Vom citi fişierul de intrare linie cu linie. Pentru a separa cuvintele dintr-o linie putem folosi funcţii ca `strtok()` sau `sscanf()` , dar acestea elimină delimitatorii (adică spaţiile albe, tab-urile etc.) şi noi dorim să le copiem şi pe acestea în fişierul de ieşire. Deci aplicăm altă metodă: căutăm apariţii ale şirului de şters în linia curentă şi apoi verificăm dacă sunt *cuvinte separate* (delimitate de spaţii) sau nu.
Exemplu: dacă *ab* este cuvântul de şters, nu trebuie să-l ştergem din linia:
abcd xyz (nu este cuvânt separat)

Verificarea, dacă un şir este cuvânt sau nu, este făcută în funcţia "`e_cuvânt()`", care întoarce un rezultat pozitiv dacă:

- şirul este la începutul liniei şi este urmat de spaţiu, sau:
- şirul este la sfârşitul liniei şi este precedat de spaţiu, sau:
- şirul este în interiorul liniei şi înainte şi după el sunt spaţii.

(*Observaţie*: verificarea, dacă un caracter este spaţiu, se face cu funcţia `isspace()` , pentru generalitate).

După ce am generat fişierul de ieşire, îl ştergem pe cel iniţial şi îl redenumim pe cel nou (dându-i numele fişierului iniţial).

7_17.cpp

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>

#define LMAX 81  // lungimea maxima a unei linii + 1
#define nume_temp "res.tmp" // numele temporar al fisierului
de iesire

/* afisarea unui mesaj de eroare si iesirea din program: */
void eroare(char *s) {
  perror(s);
  exit(-1);
}

/* intoarce 1 daca sirul "cuv" e cuvant separat si 0 altfel:
 * linie = linia din care face parte cuvantul
 * poz = pointer catre inceputul cuvantului in linie
 */
```



```
int e_cuvant(char *linie, char *poz, char *cuv) {
  /* daca sirul este la inceput de linie: */
  if (poz == linie) {
    if (strlen(linie) == strlen(cuv)) /* sir singur pe linie */
      return 1;
    if (isspace(*(poz + strlen(cuv)))) // dupa sir urmeaza un
                                       // spatiu
      return 1;
    return 0; /* dupa sir nu urmeaza spatiu */
  }
  /* daca sirul este la sfarsit de linie: */
  if (poz + strlen(cuv) == 0) {
    if (isspace(*(poz - 1))) /* inainte de cuvant este spatiu */
      return 1;
    return 0;
  }
  /* sirul nu este nici la inceput, nici la sfarsit de linie: */
  if (isspace(*(poz - 1)) && isspace(*(poz + strlen(cuv))))
    return 1;
  return 0;
}

void main(void) {
  FILE *fin, *fout; // fisierele de intrare si de iesire
  char numefis[20]; // nume pentru fisiere
  char cuv[20];  // cuvantul de sters
  char *linie;   // linia curenta din fisier
  char *rez;     // linia care se copiaza in fisierul de iesire
  char * poz;// pointer la inceputul cuvantului in linia curenta
  char *linie2;  // pointer pentru deplasarea in  cadrul liniei
  printf("Numele fisierului de intrare: ");
  scanf("%s", numefis);
  fin  = fopen(numefis, "rt");
  if (fin == NULL) eroare("fopen");
  printf("Cuvantul de sters: ");
  scanf("%s", cuv);
  fout  = fopen(nume_temp, "wt");
```

```
  if (fout == NULL) eroare("fopen");
  linie = new char [LMAX];
  linie2 = new char [LMAX];
  rez = new char [LMAX];
  poz = new char [LMAX];
  // parcurgem fisierul de intrare linie cu linie:
  while (fgets(linie, LMAX, fin)) {
    // initializam linia care se va copia in fisierul de iesire:
    rez[0] = 0;
    linie2 = linie;
    // cautam cuvantul:
    while ((poz = strstr(linie2, cuv)) != NULL) {
      if (e_cuvant(linie, poz, cuv))
        // copiem partea din linie dinaintea cuvantului de
        // sters:
        strncat(rez, linie2, (poz - linie2));
      else
        // nu e cuvant separat, nu se sterge:
        strncat(rez, linie2, (poz - linie2) + strlen(cuv));
      // modificam pozitia curenta
      linie2 = poz + strlen(cuv);
    }
    // nu am mai gasit cuvantul, copiem restul liniei:
    strcat(rez, linie2);
    fputs(rez, fout);
  }
  fclose(fin);
  fclose(fout);
  // stergem fisierul de intrare:
  if (remove(numefis) != 0)
   eroare("remove");
  // redenumim fisierul de iesire (cu numele celui de intrare):
  if (rename(nume_temp, numefis) == 0)
    printf ("Fisierul a fost modificat");
  else
    eroare("rename");
  getch();
  delete[] linie;
  delete[] linie2;
  delete[] rez;
  delete[] poz;
}
```


Fişiere binare - Probleme propuse

P7_1. Să se scrie un program, care, folosind fişierul creat în problema 1 rezolvată, calculează şi afişează:
- Suma maximă depusă, împreună cu data şi numele depunătorului.
- Numărul depunerilor din fiecare zi şi suma totală depusă în fiecare zi, ţinând cont că tranzacţiile dintr-o zi sunt contigue în fişier.

P7_2. Să se scrie un program, care, folosind fişierul creat în problema 1 rezolvată, actualizează acest fişier 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 funcţie care determină numărul de zile între data depunerii şi data curentă, pentru a calcula dobânda cuvenită.

P7_3. Se consideră dat fişierul de întregi `numere.dat` . Să se creeze fişierul `prime.dat` , conţinând numai acele elemente din fişierul iniţial care sunt numere prime.

P7_4. Se consideră fişierul abonaţi.dat cu articole structuri având câmpurile:
- `Nume` – un şir de 20 de caractere
- `Adresă` – un şir 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 fişierul de abonaţi, ştergând pe aceia al căror abonament a expirat la data curentă. Actualizarea se face creând un nou fişier în care se trec numai abonaţii al căror abonament nu a expirat şi care la sfârşit va primi numele fişierului iniţial.
Se va defini şi folosi o funcţie care compară două date (d_1 şi d_2) şi întoarce 1, dacă d_1 este înaintea lui d_2, şi 0 în caz contrar.

P7_5. Un fişier binar conţine valori întregi ordonate crescător. Să se creeze un nou fişier cu valorile de mai sus ordonate strict crescător.

P7_6. O bază de date este compusă din `na` articole, iar un articol este format din `nc` câmpuri. De exemplu:

Nume	Grupa	Nota1	Nota2	Nota3
Popescu Ion	313CA	6	8	4
...	...			
Ionescu Tudor	311CA	3	5	6

În acest exemplu un articol are 5 câmpuri. Descrierea câmpurilor (capul de tabel) se face prin:

- `număr` de câmpuri din articol (`nc` – întreg)
- `nume` câmp (11 caractere)
- `tip` câmp (1 caracter: *C* = şir de caractere, sau *N* = numeric)
- `lungime` câmp (un întreg)

ultimele 3 descrieri apar pentru fiecare dintre cele `nc` câmpuri.
Descrierea câmpurilor ocupă prima porţiune din fişier. În exemplul dat, descrierea câmpurilor este:

```
5Nume    C20Grupa   C 5Nota1   N 2Nota2   N 2Nota3   N 2
```

Urmează apoi articolele din fişier:

```
Popescu Ion          313CA 6 8 4
```

Fişierul bază de date are numele dat ca parametru al comenzii. Tot ca parametri se mai dau două şiruri de caractere, reprezentând un nume de câmp şi o valoare de câmp (De exemplu: `Grupa` şi `313CA`).
Se cere:

- să se determine lungimea unui articol şi numărul de articole
- să se afişeze toate articolele având în numele de câmp specificat valoarea specificată (în exemplul nostru, vor fi afişaţi toţi studenţii din grupa 313CA).

Indicaţie: Din fişierul binar bază de date se citeşte mai întâi numărul de câmpuri din articol, şi apoi se citesc descrierile câmpurilor.
Se adună lungimile câmpurilor, obţinându-se lungimea articolului.
Numărul de articole se obţine împărţind spaţiul cuprins după descrierea câmpurilor, până la sfârşitul fişierului, la lungimea articolului.
Se caută, în continuare, numele de câmp în descrierea câmpurilor. Dacă nu este găsit, se dă un mesaj de eroare şi se termină programul. În caz contrar, se reţine poziţia câmpului în articol, se citeşte valoarea câmpului din fişier şi se compară cu valoarea căutată: în caz de egalitate se afişează tot articolul.
Se repetă operaţia pentru fiecare articol, poziţionându-ne pe câmpul localizat.

P7_7. Se consideră fişierele binare: `carti` cu articole (`autor, titlu`), `ani` cu articole (`titlu, an`), `edituri` cu articole (`editura, titlu`).
Pe baza acestora se cere să se creeze un fişier binar cu articole (`autor, an, număr_edituri`) în care nu există două articole cu câmpurile `autor` şi `an` identice.
Fişierul creat are numele dat ca parametru al comenzii.

P7_8. Un client lansează o precomandă la o firmă specificând:
- `nume client` (30 caractere)
- `produs comandat` (30 caractere)
- `cantitatea comandata` (real).

Pentru a onora precomanda, firma caută produsul comandat în depozit şi trimite clientului care a lansat comanda un răspuns de forma:
- `adresa client`
- mesaj de forma: `Produsul xxx există şi costă XXXX` sau `Produsul xxx nu există`

Pentru a trimite aceste răspunsuri, firma foloseşte o agendă conţinând perechi:
- `nume client`
- `adresa client`

Scrieţi un program care foloseşte fişierele binare: `comenzi`, `depozit` şi `agenda` şi creează fişierul text `raspunsuri`.

P7_9. O agenţie matrimonială dispune de o bază de date conţinând următoarele informaţii despre candidaţi::
- sex (1 caracter)
- vârsta (întreg)
- înălţime (real)
- profesie (20 caractere)
- venit (real)
- nume (30 caractere)
- adresă (30 caractere)

Solicitanţii de servicii matrimoniale completează un formular conţinând:
- numele solicitantului

precum şi primele 5 câmpuri din cele de mai sus.
Criteriile de selecţie pentru aceste câmpuri sunt: ==, <=, >=, ==, >= .
Scrieţi un program, care, folosind fişierele binare "`agentie`" şi "`criterii`", selectează dintre candidaţii care satisfac aceste criterii pe cel cu venitul cel mai mare şi creează un fişier text cu răspunsuri de forma:
`<Nume_solicitant> perechea potrivita este <Nume_candidat> <Adresa_candidat>`
sau:
`<Nume_solicitant> perechea potrivita nu exista încă.`

P7_10. Se da un fişier binar cu nume preluat ca parametru al comenzii, având articole structuri de forma:
- nume depunător (30 caractere)
- data depunerii (3 întregi)
- suma depusă.

Scrieţi un program care actualizează acest fişier, adăugând la suma depusă dobândă la data curentă, introdusă de la tastatură. Dobânda anuală este 45%.

Indicaţie: $\text{suma} <- \text{suma} \cdot \left(1 + 0{,}45 \frac{\text{nrzile}}{365}\right)$.

P7_11. Mai mulți clienți lansează unei firme mai multe comenzi de forma:
- nume client (30 caractere)
- nume produs (20 caractere)
- cantitate comandată (real)

Un client poate apare de mai multe ori în comenzi diferite, eventual cu produse diferite.
Firma face o centralizare a comenzilor pe clienți, astfel încât să existe o singură comandă valorică pentru un client de forma:
- nume client (30 caractere)
- valoare totală produse comandate(real)

În acest scop firma își consultă catalogul de produse (un fișier text) care conține linii cu numele produsului și prețul unitar, separate prin spații libere.
Scrieți un program, care, folosind fișierul binar de comenzi și fișierul text catalog de produse, creeaza fișierul binar de comenzi valorice centralizate pe clienți (astfel că un client apare într-o singură comandă).
Comenzile de produse inexistente în catalog sunt listate la ieșirea standard de eroare. Numele celor 3 fișiere se preiau ca parametri ai comenzii.
Indicație: O comandă este căutată după nume produs în catalog; dacă este găsită se calculează valoarea și este căutată după client în fișierul de comenzi centralizate - dacă este găsită, se modifică valoarea, în caz contrar se creează un nou articol care se adaugă la sfârșit.

P7_12. Mai mulți clienți lansează unei firme mai multe comenzi de forma:
- nume client (30 caractere)
- nume produs (20 caractere)
- cantitate comandată (real)

Un același produs poate apare de mai multe ori în comenzile mai multor clienți.
Firma își centralizează comenzile pe produse, astfel încât să existe o singură comandă valorică la un produs (cerut de mai mulți clienți).
În acest scop firma își consultă catalogul de produse (un fișier text) care conține linii cu numele produsului și prețul unitar, separate prin spații libere.
Scrieți un program, care, folosind fișierul binar de comenzi și fișierul text catalog de produse, creează fișierul binar de comenzi valorice centralizate pe produse, conținând perechi:
- nume produs (20 caractere)
- valoare totală comandată (real)

Comenzile de produse inexistente în catalog sunt listate la ieșirea standard de eroare.
Numele celor 3 fișiere se preiau ca parametri ai comenzii.

P7_13. Scrieți un program care inserează într-un fișier text, începând cu o linie cu număr dat, linii din alt fișier.
Numele fișierului în care se face inserarea, numărul liniei din acest fișier unde începe inserarea, numele fișierului inserat, numărul liniei de unde se inserează și numărul de linii inserate se preiau ca parametri ai comenzii. Fișierul modificat primește în final numele fișierului inițial.
Exemplu: `inser f1.txt 10 f2.txt 50 25`

Fișiere text - Probleme propuse

P7_14. Scrieți o funcție care afișează cuvintele palindroame dintr-un șir de caractere dat ca parametru. Cuvintele din șir sunt separate prin: spațiu, punct, virgulă, două puncte și punct-virgulă.
Scrieți o funcție `main()` care citește linii din fișierul text `sursa.txt` și afișează numărul liniei și cuvintele palindroame din linia respectivă.

P7_15. Un fișier text conține un bloc delimitat de șirurile de caractere #KB și #KK . Să se creeze din acest bloc un nou fișier având același nume și extensia `.blk`. Numele fișierului sursă este citit de la tastatură.

P7_16. Un fișier text conține un bloc delimitat de șirurile #B și #K . Să se creeze un nou fișier din fișierul dat ștergând acest bloc. Noul fișier are același nume cu fișierul inițial dar nu va avea nici o extensie.

P7_17. Un fișier text conține blocuri delimitate de caracterele #B și #K, formate dintr-un număr întreg de linii. Să se afișeze din acest fișier, la imprimantă, numai blocurile marcate.

P7_18. Un fișier text conține blocuri delimitate de caracterele #B și #K, formate dintr-un număr întreg de linii. Să se creeze un nou fișier, prin concatenarea blocurilor din fișierul dat.

P7_19. Pentru obținerea unei gradații, o persoană trebuie să susțină cel puțin `np` probe la care să obțină o medie de cel puțin `med`.
O linie dintr-un fișier text de intrare conține numele candidatului și calificativele obținute la probele susținute. Numele este separat de note prin spații libere, iar notele sunt separate între ele prin virgule și spații.

Creați un fișier binar cu persoanele care au obținut gradația. Acesta va avea articole cu câmpurile: `nume` (șir de 30 de caractere), `număr_de_examene` prezentate (întreg) și `media` obținută (real).
Numele celor două fișiere, numărul minim de examene și media de promovare sunt date ca parametri ai comenzii.

P7_20. Pornind de la un fișier text, să se creeze un alt fișier cu același nume dar cu extensia `.pal`, format din liniile din fișierul inițial care reprezintă palindroame. Numele fișierului inițial este dat ca parametru al comenzii.
Exemplu de linie palindrom: `'A man, a plan, a canal - Panama!`

P7_21. Un fișier text, reprezentând un program sursă, conține comentarii de tip C (`/*. . .*/`), în care pe o linie se pot afla mai multe comentarii, dar și un comentariu se poate întinde pe mai multe linii.
Scrieți un program care modifică acest text, astfel încât comentariile să fie în stilul C++, adică `//...` un comentariu terminându-se cu un sfârșit de linie.
Numele fișierului de intrare este dat ca parametru al comenzii. Fișierul de ieșire va avea același nume și extensia `.cpp`.

P7_22. Scrieți un program care extrage dintr-un fișier text toate cuvintele reprezentând constante întregi zecimale și le plasează într-un fișier binar de întregi lungi. Separatorii între cuvinte pot fi toate spațiile albe, virgulă și punct virgulă. Numele fișierului text este preluat din linia de comandă. Fișierul binar are același nume și extensia .BIN.

P7_23. Scrieți un program care șterge dintr-un fișier text, începând cu o linie specificată un număr de linii și creeaza un nou fișier.
Numele fișierului, linia și numărul de linii sunt date ca parametri ai comenzii. Fișierul modificat va avea același nume cu fișierul inițial.
Exemplu: `sterge date.txt 20 5`

P7_24. Scrieți un program care citește un fișier text și crează un nou fișier, făcând toate liniile din fișier de aceeași lungime *n*.
Numele fișierului de intrare și valoarea lui *n* se dau ca parametri ai comenzii. Fișierul de ieșire are același nume, dar extensia `.out`
Indicație: Linia scrisă în fișierul de ieșire se formează concatenând la restul liniei precedente linia curentă. În momentul în care linia de ieșire depășește (>=) *n* caractere se scriu câte *n* caractere din ea în fișierul de ieșire. Caracterele rămase trec în linia precedentă.

P7_25. Un fișier text conține linii cu următoarea structură: un nume (de student) urmat de mai multe valori întregi, separate între ele prin spații libere, numere reprezentând notele obținute de student la examenele la care s-a prezentat.
Știind că în total s-au susținut *n* examene, să se creeze un nou fișier text, în care fiecare linie conține numele studentului, media notelor la examenele susținute și numărul de examene rămase a fi susținute.
Numele fișierului de intrare și numărul total de examene (*n*) se preiau din linia de comandă.

P7_26. Scrieți un program care înlocuiește într-un fișier text toate aparițiile unui cuvânt prin alt cuvânt. Numele fișierului și cele două cuvinte sunt preluate ca parametri ai liniei de comandă. Se presupune că tabloul în care se memorează linia este suficient de mare pentru a conține linia modificată. În final, fișierul modificat va avea același nume cu fișierul inițial.

Capitolul 8
Clase

Breviar

Declararea unei clase:

```
class nume_clasa{
tip_acces:
  declarare date membri;
  prototipuri funcţii membri;

  friend tip nume_functie_nemembru(lista_parametri);
}
```

Definirea funcţiilor clasei şi a funcţiilor nemembri (prieteni):

```
tip nume_clasa:: nume_functie_membru(lista_parametri){
                    corp_functie;
}
tip nume_functie_nemembru(lista_parametri){
                    corp_functie;
}
```

Declararea unei clase derivate:

```
class nume_clasa_derivata : tip_acces nume_clasa_parinte{
  declarare date si functii membri;
}
```

Probleme rezolvate

R8_1. Proiectaţi şi implementaţi clasa `Raţional` care să permită lucrul cu numere raţionale.
Constructorul clasei va avea două argumente: numărătorul, respectiv numitorul numărului raţional (constructorul poate avea şi un singur argument, caz în care al doilea se ia implicit 1,sau nici un argument, caz în care se iau valorile implicite 0 şi 1).
Se va asigura un constructor de copiere.
Se vor prevedea funcţii membri pentru:
- accesul la numărătorul, respectiv numitorul numărului raţional

- redefinirea operatorilor `+=`, `-=`, `*=`, `/=` pentru adunarea, scăderea, înmulţirea şi împărţirea numărului raţional cu un alt număr raţional dat ca argument

Se va redefini operatorul >> ca funcţie prieten pentru citirea unui număr raţional de la intrarea standard
Se vor asigura funcţii nemembru pentru:
- testul de egalitate a două numere raţionale (redefinirea operatorului ==)
- scrierea unui număr raţional la ieşirea standard (redefinirea operatorului <<)
- redefinirea operatorilor +, -, *, / pentru a permite operaţii cu două argumente numere raţionale.
- redefinirea operatorului de atribuire.

Scrieţi o funcţie `main()` care citeşte n fracţii raţionale şi calculează suma lor.

Rezolvare:

8_1.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <stdlib.h>

class Rational{
  int ns, nj; // numarator si numitor
public:
  int cmmdc(int a, int b);
  Rational(int n1=0, int n2=1){
    ns = n1; nj = n2;
  }
  Rational(Rational& r){
    ns = r.sus();
    nj = r.jos();
  };
  int sus() const { return ns; }
  int jos() const { return nj; }
  void setsus(int x) { ns = x; }
  void setjos(int y) { nj = y; }
  Rational& operator+=(const Rational& r);
  Rational& operator-=(const Rational& r);
  Rational& operator*=(const Rational& r);
  Rational& operator/=(const Rational& r);
  friend istream& operator>>(istream& is, Rational& r);
  friend ostream& operator<<(ostream& os, const Rational& r);
  friend int operator==(const Rational& r1, const Rational& r2);
```

```
  friend Rational operator+(const Rational& r1,
                            const Rational& r2);
  friend Rational operator-(const Rational& r1,
                            const Rational& r2);
  friend Rational operator*(const Rational& r1,
                            const Rational& r2);
  friend Rational operator/(const Rational& r1,
                            const Rational& r2);
  friend void simplifica(Rational& r);
};

Rational& Rational::operator+=(const Rational& r){
  int t = ns * r.jos() + nj * r.sus();
  nj = nj * r.jos();
  ns = t;
  return *this;
};

Rational& Rational::operator-=(const Rational& r){
  int t = ns * r.jos() - nj * r.sus();
  nj = nj * r.jos();
  ns = t;
  return *this;
};

Rational& Rational::operator*=(const Rational& r){
  ns *= r.sus();
  nj *= r.jos();
  return *this;
};

Rational& Rational::operator/=(const Rational& r){
  ns *= r.jos();
  nj *= r.sus();
  return *this;
};

int Rational::cmmdc(int a, int b){
  int r;
  do {
    r = a % b;
    a = b;
    b = r;
  } while(r);
  return a;
};
```



```
istream& operator>>(istream& is, Rational& r){
  int x, y;
  is >> x >> y;
  r.setsus(x);
  r.setjos(y);
  return is;
};

ostream& operator<<(ostream& os, const Rational& r){
  os << r.sus() << " / " << r.jos();
  return os;
};

int operator==(Rational& r1, Rational& r2){
  simplifica(r1);
  simplifica(r2);
  return r1.sus()==r2.sus() && r1.jos()==r2.jos();
};

Rational operator+(const Rational& r1, const Rational& r2){
  Rational r(r1);
  r += r2;
  return r;
};

Rational operator-(const Rational& r1, const Rational& r2){
  Rational r(r1);
  r -= r2;
  return r;
};

Rational operator*(const Rational& r1, const Rational& r2){
  Rational r(r1);
  r *= r2;
  return r;
};

Rational operator/(const Rational& r1, const Rational& r2){
  Rational r(r1);
  r /= r2;
  return r;
};

void simplifica(Rational& r){
  int s = r.sus();
```

```
  int j = r.jos();
  int c = r.cmmdc(s, j);
  s /= c;
  j /= c;
  r.setsus(s);
  r.setjos(j);
}

void main(void){
  clrscr();
  Rational suma, t;
  int n;  //numar de termeni
  cout << "Numar de termeni=";
  cin >> n;
  cout << "Introduceti fractiile, cate una pe linie\n";
  for (int i = 0; i < n; i++){
    cin >> t;
    simplifica(t);
    suma += t;
    simplifica(suma);
  }
  cout << suma << endl;
  getch();
}
```

R8_2. Proiectați și implementați clasa `Complex` care să permită lucrul cu numere complexe.

Constructorul clasei va avea ca argumente partea reală, respectiv imaginară a numărului complex (în mod implicit aceste valori se iau 0).

Se va asigura un constructor de copiere.

Se vor prevedea funcții membri pentru:

- accesul la partea reală, respectiv imaginară a numărului complex
- redefinirea operatorilor `+=`, `-=`, `*=`, `/=` pentru adunarea, scăderea, înmulțirea și împărțirea numărului complex cu un alt număr complex dat ca argument
- modulul numărului complex
- argumentul numărului complex

Se va redefini operatorul `>>` ca funcție prieten pentru citirea unui număr complex de la intrarea standard

Se vor asigura funcții nemembru pentru:

- testul de egalitate a două numere complexe (redefinirea operatorului `==`)
- scrierea unui număr complex la ieșirea standard (redefinirea operatorului `<<`)
- redefinirea operatorilor `+, -, *, /` pentru a permite operații cu două argumente numere complexe
- redefinirea operatorului de atribuire

Rezolvare:

8_2.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
#include <conio.h>

class Cplx{
  double re, im;
public:
  //constructori
  Cplx(double x=0, double y=0);
  Cplx(const Cplx& z);
  //acces la membri
  double real() const { return re; }
  double imag() const { return im; }
  //setare membri
  void setreal(double x) { re = x; }
  void setimag(double y) { im = y; }
  //atribuire simpla
  Cplx& operator=(const Cplx& z);
  //atribuiri compuse
  Cplx& operator+=(const Cplx& z);
  Cplx& operator-=(const Cplx& z);
  Cplx& operator*=(const Cplx& z);
  Cplx& operator/=(const Cplx& z);
  //modul si argument
  double mod(const Cplx& z);
  double arg(const Cplx& z);
  //intrari/iesiri
  friend istream& operator>>(istream& is, Cplx& z);
  friend ostream& operator<<(ostream& is, const Cplx& z);
  //operatori binari
  friend int operator==(const Cplx& s, const Cplx& d);
  friend int operator!=(const Cplx& s, const Cplx& d);
  friend Cplx operator+(const Cplx& s, const Cplx& d);
  friend Cplx operator-(const Cplx& s, const Cplx& d);
  friend Cplx operator*(const Cplx& s, const Cplx& d);
  friend Cplx operator/(const Cplx& s, const Cplx& d);
  //operatori unari
  friend Cplx operator-(const Cplx& z);
  friend Cplx operator!(const Cplx& z); //conjugat
  friend Cplx operator++(Cplx& z);      //prefix
  friend Cplx operator++(Cplx& z,int); //postfix
};

// implementare clasa
```

```
Cplx::Cplx(double x, double y){
  re = x;
  im = y;
}

Cplx::Cplx(const Cplx& z){
  re = z.real();
  im = z.imag();
}

Cplx& Cplx::operator=(const Cplx& z){
  if(&z != this){
    re = z.real();
    im = z.imag();
  }
  return *this;
}

Cplx& Cplx::operator+=(const Cplx& z){
  re += z.real();
  im += z.imag();
  return *this;
}

Cplx& Cplx::operator-=(const Cplx& z){
  re -= z.real();
  im -= z.imag();
  return *this;
}

Cplx& Cplx::operator*=(const Cplx& z){
  re = re * z.real() - im * z.imag();
  im = re * z.imag() + im * z.real();
  return *this;
}

Cplx& Cplx::operator/=(const Cplx& z){
  double t=z.real()*z.real()+z.imag()*z.imag();
  re = (re * z.real() + im * z.imag()) / t;
  im = (im * z.real() - re * z.imag()) / t;
  return *this;
}

double Cplx::mod(const Cplx& z) {
  double x = z.real();
  double y = z.imag();
  return sqrt(x*x + y * y);
}
```



```
double Cplx::arg(const Cplx& z) {
  double x = z.real();
  double y = z.imag();
  return atan2(x, y);
}

istream& operator>>(istream& is, Cplx& z){
  double x, y;
  is >> x >> y;
  z.setreal(x);
  z.setimag(y);
  return is;
}

ostream& operator<<(ostream& os, const Cplx& z){
  os << "(" << z.real() << "," << z.imag() << ")" << endl;
  return os;
}

int operator==(const Cplx& s, const Cplx& d){
  return s.real()==d.real() && s.imag()==d.imag();
}

int operator!=(const Cplx& s, const Cplx& d){
    return s.real()!=d.real() || s.imag()!=d.imag();
}

Cplx operator+(const Cplx& s, const Cplx& d){
    return Cplx(s.real()+d.real(), s.imag()+d.imag());
}
/* Se mai poate scrie
 * Cplx operator+(const Cplx& s, const Cplx& d){
 *   Cplx z(s);
 *   z += d;
 *   return z;
 * }
 */

Cplx operator-(const Cplx& s, const Cplx& d){
    return Cplx(s.real()-d.real(), s.imag()-d.imag());
}
```

```
Cplx operator*(const Cplx& s, const Cplx& d){
    return Cplx(s.real()*d.real()-s.imag()*d.imag(),
           s.real()*d.imag()-d.real()*s.imag());
}

Cplx operator/(const Cplx& s, const Cplx& d){
    double t=d.real()*d.real()+d.imag()*d.imag();
    return Cplx((s.real()*d.real()+s.imag()*d.imag())/t ,
           (s.real()*d.imag()-s.real()*d.imag())/t);
}

Cplx operator-(const Cplx& z){
    return Cplx(-z.real(), -z.imag());
}

Cplx operator!(const Cplx& z){
    return Cplx(z.real(), -z.imag());
}

Cplx operator++(Cplx& z){      //prefix
     z.setreal(z.real()+1.0);
     return z;
}

Cplx operator++(Cplx& z,int){  //postfix
    Cplx t(z);
    t.setreal(t.real()+1.0);
    return t;
}

void main(void){
  clrscr();
  Cplx c1, c2, c3;
  cin >> c1;
  c2 = c1;
  cout << c2;
  c3=c2;
  cout << c3;
  c1 = c2 + c3;
  cout << c1;
  ++c1;
  cout << c1;
  c2 = c1++;
  cout << c1 << c2;
  getch();
}
```


R8_3. Clasa `Matrice` cu elemente reale este definită astfel:

```
class Matrice{
  float **a;
  int l, c;
public:
  Matrice(int l1=1, int c1=1, float vi=0);
  Matrice(Matrice& x);
  ~Matrice();
  Matrice& operator=(Matrice& x);
  float& val(int l1, int c1);
  int linii() const { return l; }
  int coloane() const { return c; }
  void setl(int l1) { l = l1; }
  void setc(int c1) { c = c1; }
  friend Matrice operator+(const Matrice& x, const Matrice& y);
  friend Matrice operator*(const Matrice& x, const Matrice& y);
  friend int operator==(const Matrice& x, const Matrice& y);
  friend ostream& operator<<(ostream& os, const Matrice& y);
  friend istream& operator>>(istream& is, Matrice& y);
};
```

Implementați această clasă.

Rezolvare:

8_3.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <assert.h>
#include <conio.h>

Matrice::Matrice(int l1, int c1, float vi){
    l = l1;
    c = c1;
    a = new float* [l];
    for (int i = 0; i < l; i++){
      a[i] = new float[c];
      for (int j = 0; j < c; j++)
        a[i][j] = vi;
    }
};

Matrice::Matrice(Matrice& x){
    l = x.linii();
    c = x.coloane();
    a = new float* [l];
```

```
    for (int i = 0; i < l; i++){
      a[i] = new float[c];
      for (int j = 0; j < c; j++)
        a[i][j] = x.val(i, j);
    }
};

Matrice::~Matrice(){
    for(int i = 0; i < l; i++)
      delete [] a[i];
    delete [] a;
};

Matrice& Matrice::operator=(Matrice& x){
    if(this == &x) return *this;
    l = x.linii();
    c = x.coloane();
    a = new float* [l];
    for (int i = 0; i < l; i++){
      a[i] = new float[c];
      for (int j = 0; j < c; j++)
        a[i][j] = x.val(i, j);
    }
    return *this;
};

float& Matrice::val(int l1, int c1){
    assert(l1>=0 && l1<l && c1>=0 && c1 < c);
    return a[l1][c1];
};

Matrice operator+(const Matrice& x, const Matrice& y){
  assert(x.linii()==y.linii() && x.coloane()==y.coloane());
  int lin = x.linii();
  int col = x.coloane();
  Matrice z(lin, col);
  for(int i = 0; i < lin; i++)
    for(int j=0; j < col; j++)
      z.val(i,j) = x.val(i,j) + y.val(i,j);
  return z;
};

Matrice operator*(const Matrice& x, const Matrice& y){
  assert(x.coloane()==y.linii());
  int lz = x.linii();
  int cz = y.coloane();
  int col= x.coloane();
```



```
  Matrice z(lz, cz);
  for(int i = 0; i < lz; i++)
    for(int j=0; j < cz; j++){
      float s =0.;
      for(int k = 0; k < col; k++)
      s+= x.val(i,k) * y.val(k,j);
      z.val(i,j) = s;
    };
  return z;
};

int operator==(const Matrice& x, const Matrice& y){
  if(x.linii()!=y.linii() || x.coloane() != y.coloane())
    return 0;
  for(int i=0; i < x.linii(); i++)
    for(int j=0; j < x.coloane(); j++)
      if(x.val(i,j) != y.val(i,j)) return 0;
  return 1;
};

ostream& operator<<(ostream& os, const Matrice& y){
  os << endl;
  for(int i = 0; i < y.linii(); i++){
    for(int j = 0; j < y.coloane(); j++)
      os << y.val(i, j) << " ";
    os << endl;
  }
  return os;
};

istream& operator>>(istream& is, Matrice& y){
  int l, c;
  cout << "Numar linii: ";
  is >> l;
  y.setl(l);
  cout << "Numar coloane: ";
  is >> c;
  y.setc(c);
  cout << "Elementele pe linii\n";
  for(int i = 0; i < y.linii(); i++)
    for(int j = 0; j < y.coloane(); j++)
      is >> y.val(i, j);
  return is;
};

void main(void){
  clrscr();
```

```
    Matrice a, b, c;
    cin >> a;
    cout << a;
    cin >> b;
    cout << b;
    c = a + b;
    cout << c;
    c = a * b;
    cout << c;
    getch();
}
```

R8_4. Clasa `Polinom`, care permite realizarea operațiilor aritmetice cu polinoame cu coeficienți reali, este definită astfel:

```
class Polinom {
protected:
   int n;
   float *data;
public:
   Polinom(int n1, float *data1);
   Polinom(int n1=0);
   Polinom(Polinom &p);
   ~Polinom(){if(data) delete [] data;};
   void setn(int m){ n = m; }
   Polinom & operator=(const Polinom &p);
   int size()const { return n; }
   void resize(int n1);
   void normalize();
   float & operator[](int i)const;
   Polinom & operator+=(Polinom &p);
   friend int operator>(const Polinom &p, float eps);
   friend Polinom operator+(Polinom &p1, Polinom &p2);
   friend Polinom operator/(const Polinom &p1, const Polinom
&p2);
   friend Polinom operator%(const Polinom &p1, const Polinom
&p2);
   friend Polinom cmmdc(const Polinom &p1,  const Polinom &p2);
   friend ostream & operator<<(ostream &os, const Polinom &p);
   friend istream & operator>>(ostream &is, Polinom &p);
};
```

Implementați aceste funcții.

Scrieți o funcție `main()` care citește două polinoame și calculează cmmdc al lor.

Rezolvare: *Indicație*: Pentru calculul câtului și restului împărțirii a două polinoame se simulează împărțirea manuală:

$$\frac{a_n \cdot x^n + \cdots + a_0}{b_m \cdot x^m + \cdots + b_0} = c_{n-m} \cdot x^{n-m} + \cdots c_0 + \frac{r_{m-1} \cdot x^{m-1} + \cdots + r_0}{b_m \cdot x^m + \cdots + b_0}$$

Dacă se ia $k=n-m : 0$ și $j=m : 0$, atunci:

$c_k = a_{k+m} / b_m$ și $a_{k+j} -= c_k \cdot b_j$ și $r_j = a_j \qquad j = m-1:0$

8_4.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <assert.h>
#define EPS 1.E-3

Polinom::Polinom(int n1){
  n = n1;
  data = new float[n + 1];
  for(int i=0; i<=n; i++)
    data[i]=0.;
}

Polinom::Polinom(int n1, float *data1){
  n = n1;
  data = new float[n + 1];
  memcpy(data, data1, (n+1) * sizeof(float));
}

Polinom::Polinom(Polinom &p){
  int j;
  if(data) delete [] data;
  n = p.size();
  data = new float[n + 1 ];
  for(j=0; j<=n; j++)
    data[j] = p[j];
}

Polinom& Polinom:: operator=(const Polinom &p){
  if( this == &p) return *this;
  delete [] data;
```

```
  n = p.size();
  data = new float[n + 1];
  for(int j=0; j<=n; j++)
    data[j] = p[j];
  return *this;
};

void Polinom::resize(int n1){
  if(n1 <= n){
    for(int j=n1+1; j <= n; j++)
      data[j] = 0.;
    n = n1;
  } else{
          float *data1 = new float[n1+1];
          memcpy(data1, data, (n + 1) * sizeof(float));
          for(int i = n + 1; i <= n1; i++)
          data1[i] = 0.;
          n = n1;
          if(data) delete [] data;
          data = data1;
        }
};

void Polinom::normalize(){
  int i;
  i = n;
  while(i >= 0 && fabs(data[i]) < EPS)
    i--;
  n = i;
  resize(n);
};

float & Polinom:: operator[](int i)const {
  assert(i >=0 && i <= size());
  return data[i];
};

int operator>(const Polinom &p, float eps){
  int gr;
  gr = p.size();
  for(int i=gr; i>=0; i--)
    if(p.data[i] > eps) return 1;
  return 0;
}

Polinom & Polinom::operator+=(Polinom &p){
  int max;
```



```
  max = (n > p.size())? n : p.size();
  if(max > n)
    resize(max);
  else
    p.resize(max);
  for(int i=0; i<=max; i++)
    data[i] += p[i];
  normalize();
  return *this;
};

Polinom operator+(Polinom &p1, Polinom &p2){
  Polinom p(p1);
  p += p2;
  return p;
};

Polinom operator/(const Polinom &p1, const Polinom &p2){
  int n1, n2, q;
  n1 = p1.size();
  n2 = p2.size();
  q = n1 - n2;
  Polinom a, b;
  a = p1;
  b = p2;
  Polinom c(q);
  int i, j;
  for(i = q; i >= 0; i--){
    c[i] = a[i+n2] / b[n2];
    for(j = n2; j >= 0; j--)
      a[i+j] -= c[i] * b[j];
  }
  a.normalize();
  c.normalize();
  return c;
};

Polinom operator%(const Polinom &p1, const Polinom &p2){
  int n1, n2, q;
  n1 = p1.size();
  n2 = p2.size();
  q = n1 - n2;
  Polinom a, b;
  a = p1;
  b = p2;
  Polinom c(q);
  int i, j;
```

```
  for(i = q; i >= 0; i--){

    c[i] = a[i+n2] / b[n2];
    for(j = n2; j >= 0; j--)
      a[i+j] = a[i+j] - c[i] * b[j];
  }
  a.normalize();
  return a;
};

Polinom cmmdc(const Polinom &p1, const Polinom &p2){
  Polinom a, b;
  a = p1;
  b = p2;
  int m;
  m = p2.size()-1;
  Polinom r(m);
  do {
      r = a % b;
      r.normalize();
      a = b;
      a.normalize();
      b = r;
  } while (r.size() >= 0 && r > EPS);
  return a;
};

ostream & operator<<(ostream &os, const Polinom &p){
  int m = p.size();
  for(int j = m; j >= 0;j--){
    if(p[j] > 0 && j < m)
      os << "+";
    os << p[j] << "*x^" << j << " ";
  };
  os << endl;
  return os;
};

istream & operator>>(istream &is, Polinom &p){
  int m;
  is >> m;
  p.setn(m);
  //cout << "coeficienti in ordine crescatoare puteri:\n";
  for(int j = 0; j <= m; j++)
    is >> p[j];
  return is;
}
```



```
void main(){
  clrscr();
  Polinom aa(3);
  ifstream f("DATE.IN");
  f >> aa; cout << "a=" << aa;
  Polinom bb(2);
  f >> bb; cout << "b=" << bb;
  f.close();
  Polinom c(3);
  c = cmmdc(aa, bb);
  cout << "c=" << c;
  getch();
}
```

R8_5. Fie definiția clasei String:

```
class String {
protected:
  int n; // Cata memorie este alocata
  char* data;
public:
  String();
  String(char*);
  String(String&);
  ~String();
  // Modifica lungimea sirului de caractere
  void resize(int new_size);
  // Lungimea sirului de caractere
  int size();
  // Verifica daca sirul este vid
  int is_empty();
  // Cauta s in obiect incepand din p
  int find(String& s, int p);
  // Cauta aparitia unui character din s in obiect incepand din p
  int find_first_of(String& s, int p);
  // Cauta primul character din obiect ce nu este in s incepand din p
  int find_first_not_of(String& s, int p);
  // Creaza subsir, incepand din p, de lungime l
  String substr(int p, int l);
  // Insereaza in obiect incepand din p, sirul s
  void insert(int p, String& s);
  // Sterge din sirul obiect l caractere, incepand din p
  void remove(int p, int l);
  // Inlocuieste in obiect incepand din p, l caractere cu sirul s
```

```
  void replace(int p, int l, String& s);
  // Supraincarcare operator indexare
  char& operator[](int);
  // Supraincarcare operator atribuire
  void operator=(String&);
  // Supraincarcare operator concatenare
  void operator+=(String&);
  // Supraincarcare operator verificare egalitate
  friend operator==(String&, String&);
  // Supraincarcare operator <<
  friend ostream& operator<<(ostream& , String&);
} ;
```

Implementați această clasă.

Rezolvare: Funcția `resize()` realocă spațiu de memorie pentru obiect. Dacă zona realocată este mai mare, se face o nouă alocare de memorie, iar apoi se copiază datele vechi și se completează la dreapta cu spații. Dacă zona realocată este mai mică decât cea actuală, nu se face o altă alocare, ci se pune terminatorul de șir după noul șir redimensionat.

8_5.cpp

```
#include <string.h>
#include <assert.h>
#include <iostream.h>

#define NOT_FOUND -1

void String::resize(int lnou) {
  if (data == 0) n=0;
  if (lnou < n) data[lnou] = '\0';
  else {
       int i;
       char* data_nou = new char[lnou+1];
       assert(data_nou);
       for (i=0; i<n && data[i]!='\0'; i++)
         data_nou[i] = data[i];
       for ( ; i < lnou; i++)
         data_nou[i] = ' ';
       data_nou[i] = '\0';
       if(data != NULL) delete [] data;
       data = data_nou;
       n = lnou;
  }
```



```
}
String::String() {
  data = 0;
  resize(0);
}

String::String(char* p) {
  data = 0;
  int l;
  for(l = 0 ; *(p+l); l++) ;
  resize(l);
  strcpy(data, p);
}

String::String(String& str) {
  data = 0;
  resize(str.size());
  strcpy(data, str.data);
}

void String::operator=(String& str){
  if (this != &str) {
       resize(str.size());
       strcpy(data, str.data);
  }
}

String::~String() {
  delete [] data;
}

int String::size() {
  for (int i = 0; i < n; i++)
       if ( data[i] == '\0' )
         return i;
  return n;
}

int String::is_empty() {
  return data[0]=='\0';
}

char& String::operator[](int i) {
  assert (i <= n);
  return data[i];
}
```

```
String String::substr(int poz, int lg) {
  assert(poz+lg <= size());
  String s;
  s.resize(lg);
  for(int i = 0; i<lg; i++)
      s[i] = data[poz+i];
  return s;
}

void String::remove(int poz, int lg) {
  int stop = poz + lg;
  while(stop < n && data[stop] != '\0')
      data[poz++] = data[stop++];
  data[poz] = '\0';
}

void String::insert(int poz, String& s) {
  int lg = size();
  int ls = s.size();
  int ln = lg + ls;
  int i;
  resize(ln);
  for(i = lg; i >= poz; i--)
      data[i+ls] = data[i];
  for(i = 0; i < ls; i++)
      data[poz+i] = s[i];
}

void String::replace(int poz, int lg, String& s) {
  remove(poz, lg);
  insert(poz, s);
}

void String::operator+=(String& s) {
  insert(size(), s);
}

String operator+(String& s1, String& s2) {
  String copie(s1);
  copie += s2;
  return copie;
}

int operator==(String& s1, String& s2) {
  return strcmp(s1.data, s2.data) == 0;
}
```



```
int String::find(String& s, int poz) {
  int l = s.size();
  int stop = size() - l;
  for (int i = poz; i <= stop; i++)
      if (substr(i, l) == s)
        return i;
  return NOT_FOUND;
}

int String::find_first_of(String& s, int poz) {
  int l = size();
  for(int i = poz; i < l; i++)
      if ( strchr(s.data, data[i]) )
        return i;
  return NOT_FOUND;
}

int String::find_first_not_of(String& s, int poz) {
  int l = size();
  for (int i = poz; i < l; i++)
      if ( !strchr(s.data, data[i]) )
        return i;
  return NOT_FOUND;
}

ostream& operator<<(ostream& out, String& s) {
      out << s.data;
      return out;
}
```

R8_6. Pentru a converti numere întregi de orice lungime între baze diferite, definim clasa NLB (Număr Lung în baza B):

```
class NLB{
protected:
  int n;       //lungime
  int b;       //baza
  char* data; //sir de caractere ce pastreaza numarul si terminatorul
  int cifra(char);
public:
  NLB(int b1=10, char* data1=""); //constructor de initializare
  NLB(NLB &x);                    //constructor de copiere
  ~NLB();                         //destructor
  NLB & operator=(NLB& x);        //operator de atribuire
  void resize(int n1);
```

```
  int size()const { return n; };
  int base()const { return b; };
  char& operator[](int i);
  NLB operator+(int p); //aduna intregul p la numarul lung
  NLB operator*(int p); //inmulteste numarul lung cu intregul p
  NLB operator/(int p); //imparte numarul lung cu intregul p
  int operator%(int p); //restul impartirii numarului lung la p
  NLB cv10B(int p);     //conversie din baza 10 in baza p
  NLB cvB10();          //conversie din baza 10 in baza b
  NLB operator>(int p); //conversie din baza b in baza p

  friend ostream& operator<<(ostream& os, NLB& x);
};
```

Implementați această clasă, pentru ca programul de mai jos să fie executabil.

```
void main(){
  NLB x(2,"1101001"), y(8,"253647"), z;
  z=x > 3;
  cout << z;
}
```

Rezolvare: Numărul lung este păstrat ca un şir de caractere, alocat dinamic, începând cu cifra cea mai semnificativă în poziția 0 şi incluzând terminatorul de şir.

Funcția privată `int cifra(char);` ne permite să aflăm valoarea unui caracter cifră, conversia inversă făcându-se prin intermediul tabloului `hexa[]`.

Funcția `resize()` realocă spațiul de memorie alocat numărului lung. Dacă se alocă mai puțină memorie, nu se face realocare ci se trunchiază numărul în pozițiile cele mai semnificative. Dacă se alocă mai multă memorie, în spațiul nou alocat, numărul este aliniat la dreapta şi completat la stânga cu caractere `'0'`.

8_6.cpp

```
#include <iostream.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#include <conio.h>

char hexa[] = "0123456789ABCDEF";

NLB::NLB(int b1, char* data1){
  int n1 = strlen(data1);
  data = new char[n1+1];
  strcpy(data, data1);
  n = n1;
  b = b1;
};
```



```
NLB::NLB(NLB &x){
  int n1 = x.size();
  data = new char[n1+1];
  n = n1;
  b = x.base();
  for(int i=0; i<=n; i++)
    data[i] = x[i];
};

NLB::~NLB(){
  n = 0;
  delete [] data;
};

NLB & NLB::operator=(NLB& x){
  if(this == &x) return *this; //evita autoatribuirea
  int n1 = x.size();
  delete [] data;
  data = new char[n1+1];
  n = n1;
  b = x.base();
  for(int i=0; i<=n; i++)
    data[i] = x[i];
  return *this;
};

char & NLB::operator[](int i){
  assert(i >= 0 && i <= n);
  return data[i];
};

void NLB::resize(int n1){
  if(n1 <=n)
    memmove(data, data+n-n1,n1+1); //trunchiaza partea cms
  else {
    char* data1 = new char[n1 + 1];
    memset(data1, '0', n1 - n);
    strncpy(data1 + n1 - n, data, n);
    data1[n1] = '\0';
    delete [] data;
    data = data1;
  }
  n = n1;
};
```

```
int NLB::cifra(char c){
  assert(isdigit(c) || isxdigit(c));
  for(int i = 0; i < 16 && c != hexa[i]; i++)
                ;
  return i;
}

NLB NLB::operator+(int p){
  NLB z;
  int lz = size();
  int sum = p, tr = 0;  //suma pe rang si transport
  z.resize(lz);
  for(int j = lz-1; j >= 0; j--){
    sum += tr + cifra(data[j]);
    tr = sum / b;
    sum %= b;
    z[j] = hexa[sum];
    sum = 0;
  };
  if(tr){
    z.resize(++lz);
    z[0] = hexa[tr];
  }
  return z;
};

NLB NLB::operator*(int p){
  NLB z;
  int lz = size();
  int prod, tr = 0;  //produs partial si transport
  z.resize(lz);
  for(int j = lz-1; j >= 0; j--){
    prod = tr + cifra(data[j]) * p;
    tr = prod / b;
    prod %= b;
    z[j] = hexa[prod];
  };
  if(tr){
    z.resize(++lz);
    z[0] = hexa[tr];
  }
  return z;
};

NLB NLB::operator/(int p){
  NLB z;
  int lz = size();
```



```
  int rp = 0;  //rest partial
  z.resize(lz);
  for(int j = 0; j < lz; j++){
    rp = b * rp + cifra(data[j]); //noul rest partial
    z[j] = hexa[rp / p];
    rp %= p;
  }
  while(z[0] == '0' && z.size() > 0)
    z.resize(--lz);
  return z;
};

int NLB::operator%(int p){
  NLB z;
  int lz = size();
  int rp = 0;  //rest partial
  z.resize(lz);
  for(int j = 0; j < lz; j++){
    rp = b * rp + cifra(data[j]); //noul rest partial
    z[j] = hexa[rp / p];
    rp %= p;
  }
  while(z[0] == '0' && z.size() > 0)
    z.resize(--lz);
  return rp;
};

NLB NLB::cvB10(){
  if(b==10) return *this;
  NLB z(10), y;
  int lz = size();
  z.resize(2*lz);  //convertind din baza 16 avem mai multe cifre
  for(int j=0; j < lz; j++){
    y = z * b;
    z = y + cifra(data[j]);
  }
  while(z[0] == '0' && z.size() > 0)
    z.resize(--lz);
  return z;
};

NLB NLB::cv10B(int p){
  if(p==10) return *this;
  NLB z, w(p,"0");
  int lz = size();
  int j, k, i;
  z.resize(lz);              //numarul in baza 10
```

```
  w.resize(4 * lz);          //numarul in baza p
  for(j = 0; j < lz; j++)   //copiaza obiectul in z
    z[j] = data [j];
  for(j = 0; j < 4 * lz; j++){ //cifrele se obtin inversate
    w[j] = hexa[z % p];
    z = z / p;
  }
  for(k = 0, i = 4*lz-1; k < i; k++, i--){
      char c = w[k];
      w[k] = w[i];
      w[i] = c;
  };
  j = 0;
  while(w[j] == '0')
    j++;
  w.resize(4*lz - j);
  return w;
};

NLB NLB::operator>(int p){
  if(b == p) return *this;
  NLB z;
  z = cvB10();
  return z.cv10B(p);
};

ostream& operator<<(ostream& os, NLB& x){
  for(int j = 0; j < x.size(); j++)
    os << x[j];
  os << endl;
  return os;
};

void main(void){
  clrscr();
  NLB x(2,"1101001"), y(8, "253647"), z;
  cout << x;
  cout << y;
  z = x.cvB10();
  cout << z;
  z = x > 3;
  cout << z;
  getch();
}
```


R8_7. Considerăm definiția clasei `Natural`, care permite adunarea unor întregi fără semn de orice lungime în baza 10.

```
class Natural{
protected:
  int n;
  char* data;
public:
  Natural(char*);     //constructor de initializare
  Natural(Natural&); //constructor de copiere
  Natural& operator=(Natural&);
  ~Natural();
  int length()const{return n;}
  void resize(int nn);
  friend Natural operator+(Natural&, Natural&);
} ;
```

Fie clasa derivată `Intreg`, care permite adunarea numerelor întregi cu semn de orice lungime: `Natural` ← `Intreg`.

Clasa `Intreg` are un membru suplimentar – `semn` – un întreg cu valoarea 0, dacă numărul este pozitiv și 1, dacă este negativ.

Dați definiția clasei `Intreg` și implementați cele două clase. Implementați în clasa `Intreg` și operatorul << supraîncărcat.

Rezolvare: Cifrele unui număr sunt reținute într-un vector începând cu cea mai puțin semnificativă (`data[0]` este cea mai puțin semnificativă cifră). A fost aleasă această reprezentare deoarece este potrivită pentru operațiile de adunare și scădere.

Funcția `operator+()` simulează adunarea manuală, calculând pentru fiecare rang suma și transportul în rangul următor. Se pornește de la cea mai puțin semnificativă cifră și se continuă până când transportul este zero și am ajuns la ultima cifră din ambele numere.

Exemplu: 123 + 89 = 212

Vom înmulți mai întâi 123 cu cifra cea mai puțin semnificativă (9):

123	3	2	1	+
89	9	8		
Transport	0	1	1	
Rezultat	9+3=2 Transport=1	2+8+1=1 Transport=1	1+1=2 Transport=0	

Pentru a aduna numere întregi a fost definită funcția `Natural::operator-()` care scade două numere cu condiția ca rezultatul să fie număr natural. Adunarea a două numere întregi revine la a face suma/diferența modulelor acestor numere.

8_7.cpp

```
#include <iostream.h>
#include <string.h>
#include <assert.h>

#define MAX(a,b) ((a>b) ? a:b)
#define MIN(a,b) ((a<b) ? a:b)

#define TRUE  1
#define FALSE 0

class Natural {
protected:
  int n;
  char *data;
public:
  Natural() { n=0; data=NULL; }
  Natural(char *);
  Natural(Natural&);
  Natural& operator=(Natural&);
  ~Natural() { if (data) delete[] data; }
  int length() const {return n;}
  void resize(int);
  char& operator[](int i) { return data[i]; }
  friend Natural operator+(Natural&, Natural&);
  friend Natural operator-(Natural&, Natural&);
  friend int operator<(Natural&, Natural&);
};

Natural::Natural(char *sir) {
  n=strlen(sir);
  data=new char[n];
  for (int i=0;i<n;i++)
    data[i]=sir[n-i-1]-'0';
}

Natural::Natural(Natural& x) {
  n=0; data=NULL;
  resize(x.length());
  memcpy(data,x.data,n);
}
```



```
Natural& Natural::operator=(Natural& x){
  if (this != &x) {
    resize(x.length());
    memcpy(data,x.data,n);
  }
  return *this;
}

void Natural::resize(int new_size) {
  char *new_data=new char[new_size];
  memcpy(new_data,data,MIN(new_size,n));
  if (data) delete[] data;
  data=new_data;
  n=new_size;
}

Natural operator+(Natural& a, Natural& b) {
  Natural c;
  c.resize(MAX(a.length(),b.length())+1);
  int transport=0;
  int digit,i;
  for (c.n=0;c.n<a.length() || c.n<b.length() ||
       transport>0;c.n++) {
    digit=transport;
    if (c.n<a.length()) digit+=a[c.n];
    if (c.n<b.length()) digit+=b[c.n];
    c[c.n]=digit%10;
    transport=digit/10;
  }
  c.resize(c.n);
  return c;
}

Natural operator-(Natural& a, Natural& b) {
  Natural c;
  c.resize(a.length());
  // Se presupune a>=b (rezultatul este un numar natural)
  assert(a.length() >= b.length());
  int imprumut=0;
  int digit,i;
  for (i=0; i<a.length() ; i++) {
    digit=a[i]-imprumut;
    imprumut=0;
    if ( i < b.length() ) digit-=b[i];
    if (digit>=0)
      c[i]=digit;
    else {
      imprumut=1;
      c[i]=digit+10;
    }
  }
```

```
  // Se elimina posibilele zerouri din fata numarului
  while (c[c.length()-1]==0)
    c.resize(c.length()-1);
  return c;
}

int operator<(Natural& a, Natural& b) {
  int i;
  if (a.length() != b.length()) return a.length()<b.length();
  for (i=a.length();i>=0 && a[i]==b[i];i--) ;
  if (i<0) return TRUE; // sunt egale
  else return a[i]<b[i];
}

class Intreg : public Natural {
protected:
  int  semn;
public:
  Intreg();
  Intreg(char*);
  Intreg(Intreg& x) : Natural(x) {semn=x.get_semn();}
  Intreg(Natural& x) :Natural(x) {semn=0; }
  Intreg(Natural& x, int s) : Natural(x) {semn=s;}
  Intreg& operator=(Intreg&);
  ~Intreg(){};
  int  length() const { return n; }
  int  get_semn() { return semn; }
  friend Natural complement(const Natural& x);
  friend Intreg operator+(Intreg& x, Intreg& y);
  friend ostream& operator<<(ostream& os, const Intreg& x);
}; //definire Intreg

Intreg::Intreg() : Natural() { semn=1; }

Intreg::Intreg(char *sir) {
  semn=0;
  if (sir[0]=='+') {
    semn=0;
    sir++;
  } else if (sir[0]=='-') {
    semn=1;
    sir++;
  }
  n=strlen(sir);
  data=new char[n];
  for (int i=0;i<n;i++)
    data[i]=sir[n-i-1]-'0';
}
```



```
Intreg& Intreg::operator=(Intreg& x) {
  if (this != &x) {
    semn = x.semn;
    resize(x.length());
    memcpy(data,x.data,n);
  }
  return *this;
}

Natural complement(Natural& x){
  Natural z(x);
  int i;
  for(i=0; i<=x.length() && x[i]==0; i++) ;
  z[i++]=10-x[i];
  for( ; i<x.length(); i++)
  z[i]=9-x[i];
  return z;
}

Intreg operator+(Intreg& x, Intreg& y){
  Intreg z;
  Natural nx(x), ny(y), nz;
  int t;
  int ss=x.get_semn()+y.get_semn();
  switch(ss) {
  case 0:
    return Intreg((Natural) x + (Natural) y);
  case 1:
    if (ny < nx)
      if (x.get_semn()) return Intreg(nx-ny,1);
      else return Intreg(nx-ny,0);
    else
      if (y.get_semn()) return Intreg(ny-nx,1);
      else return Intreg(ny-nx,0);
  case 2:
    return Intreg((Natural) x + (Natural) y,1);
  }
}

ostream& operator<<(ostream& out, Intreg& a) {
  int i;
  if (a.get_semn()==1) out<<"-";
  if (a.length()==0) out<<"0";
  else {
    // Se sare peste eventualele zerouri din fata numarului
    for (i=a.length()-1;a[i]==0 && i>0;i--);
    // Se afiseaza numarul
    for (;i>=0;i--)
      out<< ((int) a[i]);
  }
  return out;
}
```

```
int main(void) {
  Intreg a, b, c("1");
  a=Intreg("-110");
  b=Intreg("-19");
  c=a + b + c;
  cout<<"c= "<<c<<endl;
  return 0;
}
```

R8_8. Considerăm clasa:

```
class Punct{
protected:
  float x, y;
public:
  Punct(float px=0, float py=0): x(px), y(py) {}
  float X()const{return x;}
  float Y()const{return y;}
};
```

şi ierarhia de clase: `Forma` ← `Segment` ← `Triunghi` ← `Patrulater`, având ca date membre 1, 2, 3, respectiv 4 puncte în plan, fiecare clasă moştenind datele din clasa ascendentă.

a) Daţi definiţiile claselor şi implementaţi funcţiile membre: constructor de iniţializare, constructor de copiere, operator de atribuire, pedrimetru, arie, operator << Operatorul << afişează tipul figurii, aria şi perimetrul.

b) Definiţi pentru clasa `Segment` două funcţii suplimentare care:
- verifică dacă două segmente se intersectează
- determină punctul de intersecţie a două segmente

c) Definiţi în clasa `Patrulater` o funcţie care verifică dacă cele 4 puncte formează un patrulater convex.

Operatorul <<, în cazul patrulaterului, dacă acesta nu este convex, nu afişează aria, nici perimetrul ci textul `concav`.

Implementarea trebuie să asigure legarea dinamică a funcţiilor.

Într-o implementare corectă, programul de mai jos este executabil.

```
#include "figuri.h"
void main(){
  Punct A, B(1,2), C(2,-1), D(3,1);
  Segment *ps;
  Patrulater ABCD(A,B,C,D);
  ps=&ABCD;
  float a=ps->arie(); //afisare arie patrulater
  cout << *ps;
}
```


Rezolvare: Pentru a stabili dacă două segmente AB şi CD se intersectează, verificăm dacă punctele A şi B se află în regiuni diferite de plan, determinate de CD, adică:

$$\left(\frac{xA-xC}{xD-xC}-\frac{yA-yC}{yD-yC}\right)\cdot\left(\frac{xB-xC}{xD-xC}-\frac{yB-yC}{yD-yC}\right) < 0$$

$$\left(\frac{xC-xA}{xB-xA}-\frac{yC-yA}{yB-yA}\right)\cdot\left(\frac{xD-xA}{xB-xA}-\frac{yD-yA}{yB-yA}\right) < 0$$

Pentru 4 puncte în plan A, B, C, D există 3 posibilităţi distincte de intersecţie: AB cu CD, AC cu BD şi AD cu BC.

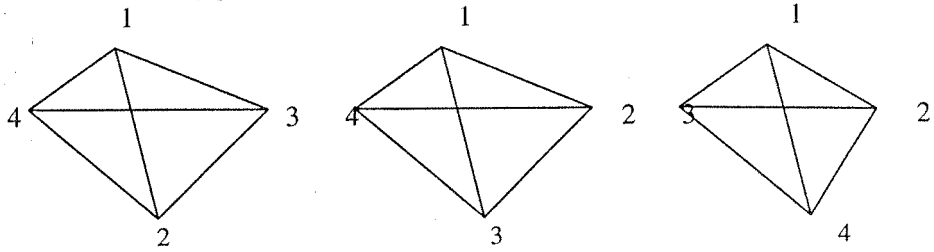
Pentru a determina punctul de intersecţie a două segmente, cu cât mai puţine calcule se foloseşte ecuaţia dreptei $y = mx + n$. Punctul de intersecţie are coordonatele:

$$x=-\frac{n_2-n_1}{m_2-m_1}, \qquad y=\frac{m_2n_1-m_1n_2}{m_2-m_1}$$

Trecerea de la ecuaţia dreptei prin două puncte: $\frac{y-y_1}{y_2-y_1}=\frac{x-x_1}{x_2-x_1}$ la ecuaţia $y = mx + n$ se face cu:

$$m=-\frac{y_2-y_1}{x_2-x_1}, \qquad y=\frac{x_2y_1-x_1y_2}{x_2-x_1}$$

La calculul ariei şi perimetrului patrulaterului convex sunt posibile următoarele 3 situaţii:



8_8.cpp

```
#include <iostream.h>
#include <math.h>
class Punct {
  float x,y;
public:
  Punct(float px=0, float py=0): x(px),y(py) {}
  Punct(const Punct& P): x(P.x), y(P.y) {}
  Punct& operator=(const Punct& P) {
      if ( this != &P) {
        x=P.x;
        y=P.y;
      }
      return *this;
  }
```

```
  float X() const { return x; }
  float Y() const { return y; }
} ; //Punct

class Forma {
protected:
  Punct P1;
public:
  Forma(const Punct& P) : P1(P) {}
  Forma(const Forma& F) : P1(F.P1) {}
  Forma& operator=(const Forma& F){
      if ( this != &F )
        P1=F.P1;
      return *this;
  }
  virtual float perimetru() const { return 0; }
  virtual float arie() const { return 0; }
  friend ostream& operator<<(ostream& os, const Forma& F) ;
} ; //Forma

ostream& operator<< (ostream& os, const Forma& F) {
  os << "Punct" << endl;
  os << "Perimetru: " << F.perimetru() << endl;
  os << "Arie: " << F.arie() << endl;
  return os;
}

class Segment : public Forma {
protected:
  Punct P2;
public:
  Segment(const Punct& PA,const Punct& PB) : Forma(PA),
P2(PB) {}
  Segment(const Segment& S) : Forma(S.P1), P2(S.P2) {}
  Segment& operator=(const Segment& S) {
      if ( this != &S ) {
        P1 = S.P1;
        P2 = S.P2;
      }
      return *this;
  }
  float perimetru() const {
      return sqrt((P1.X()-P2.X())*(P1.X()-P2.X())+
            (P1.Y()-P2.Y())*(P1.Y()-P2.Y()));
  }
  float arie() const { return 0; }
```



```
  int intersectie (const Segment& S) {
      return ((P1.X()-S.P1.X())/(S.P2.X()-S.P1.X())-
            (P1.Y()-S.P1.Y())/(S.P2.Y()-S.P1.Y()))*
          ((P2.X()-S.P1.X())/(S.P2.X()-S.P1.X())-
            (P2.Y()-S.P1.Y())/(S.P2.Y()-S.P1.Y())) < 0;
  }
  friend ostream& operator<<(ostream& os, const Segment& S)
;
  Punct punct_intersectie(const Segment& S) ;
}; //Segment

ostream& operator<<(ostream& os, const Segment& S) {
  os << "Segment" << endl;
  os << "Perimetru: " << S.perimetru() << endl;
  os << "Arie: " << S.arie() << endl;
  return os;
}

//Intoarce punctul de intersectie al segmentului cu segmentul S
Punct Segment::punct_intersectie(const Segment& S) {
  float m1=(P2.Y()-P1.Y())/(P2.X()-P1.X());
  float n1=(P2.X()*P1.Y()-P1.X()*P2.Y())/(P2.X()-P1.X());
  float m2=(S.P2.Y()-S.P1.Y())/(S.P2.X()-S.P1.X());
  float n2=(S.P2.X()*S.P1.Y()-S.P1.X()*S.P2.Y())/(S.P2.X()-
          S.P1.X());
  float x =-(n2-n1)/(m2-m1);
  float y =(m2*n1-m1*n2)/(m2-m1);
  Punct P(x,y);
  return P;
}

class Triunghi:public Segment {
protected:
  Punct P3;
public:
  Triunghi(const Punct& A, const Punct& B, const Punct& C):
        Segment(A,B),P3(C) {}
  Triunghi(const Triunghi& T) : Segment(T.P1,T.P2),P3(T.P3) {}
  Triunghi& operator=(const Triunghi& T){
      if ( this != &T ) {
        P1 = T.P1;
        P2 = T.P2;
        P3 = T.P3;
      }
      return *this;
  }
```

```
float perimetru() const {
      Segment AB(P1,P2),BC(P2,P3),CA(P3,P1);
      return AB.perimetru()+BC.perimetru()+CA.perimetru();
  }

  // Vom folosi formula lui Heron
  float arie() const {
      float p = perimetru()/2;
    Segment AB(P1,P2),BC(P2,P3),CA(P3,P1);
    return sqrt(p*(p-AB.perimetru())*(p-BC.perimetru())*
                (p-CA.perimetru()));
  }

  friend ostream& operator<<(ostream& os, const Triunghi& T);
} ; //Triunghi

ostream& operator<<(ostream& os, const Triunghi& T){
  os << "Triunghi" << endl;
  os << "Perimetru: " << T.perimetru() << endl;
  os << "Arie: " << T.arie() << endl;
  return os;
};

class Patrulater:public Triunghi {
protected:
  Punct P4;
public:
  Patrulater(Punct& A,Punct& B,Punct& C,Punct& D)
          :Triunghi(A,B,C),P4(D) {}
  Patrulater(const Patrulater& P)
          :Triunghi(P.P1,P.P2,P.P3),P4(P.P4) {}
  Patrulater& operator=(const Patrulater& P) {
      if( this != &P ) {
        P1 = P.P1;
        P2 = P.P2;
        P3 = P.P3;
        P4 = P.P4;
      }
      return *this;
  }

int este_convex() {
      Segment S12(P1,P2),S13(P1,P3),S14(P1,P4),
            S23(P2,P3),S24(P2,P4),S34(P3,P4);
      return  S12.intersectie(S34) || S13.intersectie(S24) ||
              S23.intersectie(S14);
  }
```



```
  float perimetru() const {
      Segment S12(P1,P2),S13(P1,P3),S14(P1,P4),
            S23(P2,P3),S24(P2,P4),S34(P3,P4);
      if( S12.intersectie(S34) )
        return S13.perimetru()+S23.perimetru()+
             S24.perimetru()+S14.perimetru();
      if( S13.intersectie(S24) )
        return S12.perimetru()+S23.perimetru()+
             S34.perimetru()+S14.perimetru();
      if( S14.intersectie(S23) )
        return S12.perimetru()+S24.perimetru()+
             S34.perimetru()+S13.perimetru();
      return 0;
  }
  float arie() const {
      Segment S12(P1,P2),S13(P1,P3),S14(P1,P4),
            S23(P2,P3),S24(P2,P4),S34(P3,P4);
      if( S12.intersectie(S34) ) {
        Triunghi T1(P1,P3,P4),T2(P2,P3,P4);
        return T1.arie()+T2.arie();
      }
      if( S13.intersectie(S24) ) {
        Triunghi T1(P1,P2,P4),T2(P2,P3,P4);
        return T1.arie()+T2.arie();
      }
      if( S14.intersectie(S23) ) {
        Triunghi T1(P1,P2,P3),T2(P2,P3,P4);
        return T1.arie()+T2.arie();
      }
      return 0;
  }

  friend ostream& operator<<(ostream& os, const Patrulater&
P);
}; //Patrulater

ostream& operator<<(ostream& os, const Patrulater& P){
  os << "Patrulater" << endl;
  os << "Perimetru: " << P.perimetru() << endl;
  os << "Arie: " << P.arie() << endl;
  return os;
}

void main() {
  Punct A,B(1,2),C(2,-1),D(3,1);
  Segment* ps;
  Patrulater ABCD(A,B,C,D);
  ps=&ABCD;
  float a=ps->arie();
  cout << "a=" << a <<endl;
  cout << *ps;
}
```

Probleme propuse

P8_1. Definiți și implementați clasa `Dreptunghi`, având ca date membri: `Lungimea` și `Latimea` și ca funcții membri: un constructor, `SetLungime`, `SetLatime`, `GetLungime`, `GetLatime`, `Arie` și `Perimetru`.

P8_2. Specificați, proiectați și implementați o clasă `Punct3`, ce folosește la reprezentarea punctelor (prin 3 coordonate: *x, y, z*) în spațiul 3-dimensional.

Asigurați următoarele funcții membri:

- constructor pentru setarea unui punct într-o poziție specificată (implicit în `0,0,0`)
- mutarea unui punct cu valori specificate pentru cele 3 direcții
- aflarea coordonatelor unui punct
- rotația unui punct cu un unghi specificat de-a lungul uneia dintre axele *x, y* sau *z*.

Coordonatele noului punct *x', y', z'* sunt:

- rotație în jurul axei *x*:

$$x' = x$$
$$y' = y \cdot \cos(t) - z \cdot \sin(t)$$
$$z' = y \cdot \sin(t) + z \cdot \cos(t)$$

- rotație în jurul axei *y*:

$$x' = x \cdot \cos(t) + z \cdot \sin(t)$$
$$y' = y;$$
$$z' = -x \cdot \sin(t) + z \cdot \cos(t)$$

- rotație în jurul axei *z*:

$$x' = x \cdot \cos(t) - y \cdot \sin(t)$$
$$y' = x \cdot \sin(t) + y \cdot \cos(t)$$
$$z' = z$$

Se vor defini ca funcții nemembri:

- distanța între două puncte
- testul dacă 2 puncte se confundă, prin redefinirea operatorului ==
- afișarea coordonatelor unui punct la ieșirea standard, prin redefinirea operatorului <<

Se va defini ca funcție prieten citirea coordonatelor unui punct de la intrarea standard, prin redefinirea operatorului >>

P8_3. Proiectați și implementați o clasă `Aleator`, care generează o secvență de numere întregi pseudoaleatoare, folosind metoda congruenței liniare.

În această metodă se folosesc 4 întregi: sămânța, înmulțitorul, incrementul și modulul. Cu formula:

```
(samanta * înmultitor + increment) % modul
```

se generează câte un număr aleator, care devine sămânța. În acest mod se generează "modulo" numere diferite.

Constructorul clasei are ca argumente: sămânța inițială, înmulțitorul, incrementul și modulul.

Se vor prevedea funcții membri pentru:

- schimbarea sămânței
- generarea următorului număr din secvența de numere aleatoare

P8_4. Definiți și implementați clasa `Calendar` având ca date membri: `An`, `Luna`, `Zi`, `NumeZi` (enumerarea de Luni până Duminică) și ca funcții membri:

- un constructor
- obținerea datei curente
- modificarea datei curente
- incrementarea datei curente
- afișarea datei curente.

Se va defini o funcție prieten pentru citirea datei curente.

Se vor defini de asemenea funcții nemembru cu 2 parametri date calendaristice:

- `Anterior()` care întoarce rezultatul boolean true/false după cum data argument 1 este înaintea sau după data argument 2 și
- `Interval()` care întoarce numărul de zile cuprins între cele două date argumente.

P8_5. O mulțime de întregi poate fi reprezentată printr-un vector de biți în care bitul *i* este `1` dacă întregul *i* aparține mulțimii și `0` în caz contrar. Numerotarea biților în vector se face de la stânga la dreapta și dimensiunea vectorului se numește reprezentantul mulțimii. De exemplu mulțimea `{1,5,7}` având reprezentantul `10` arată astfel:

```
0123456789
0100010100
```

În locul unui vector de biți vom folosi un vector de octeți (caractere). Pentru o mulțime cu reprezentantul *r* se alocă *r*/8+1 octeți.

Definiți *li* implementați clasa `Mulțime`, care ne permite lucrul cu mulțimi. Ca date membri se prevăd:

- reprezentantul mulțimii
- un pointer la vectorul de octeți, alocat dinamic ce reprezintă mulțimea.

Ca funcții membri avem:
- un constructor de inițializare, având ca parametru reprezentantul mulțimii (implicit 0)
- un constructor de copiere, având ca parametru un obiect mulțime
- un destructor
- o funcție care testează apartenența unui întreg la mulțime
- o funcție care adaugă un întreg la mulțime
- o funcție care scoate un întreg din mulțime
- operatorul de atribuire supraîncărcat
- operatorul *= supraîncărcat, realizând intersecția mulțimii cu mulțimea dată ca parametru
- operatorul += supraîncărcat, realizând reuniunea mulțimii cu mulțimea dată ca parametru

Ca funcții prieten se prevăd:
- operatorul * supraîncărcat pentru a realiza intersecția a două mulțimi date ca parametri
- operatorul + supraîncărcat pentru a realiza reuniunea a două mulțimi date ca parametri.

Indicație: Un element `x` al mulțimii este reperat în vectorul mulțime printr-un număr de octet (`no = x / 8`) și un număr de bit în octet (`nb = x % 8`). Prezența lui `x` în mulțime este indicată prin valoarea 1 a bitului `nb` din octetul `no` al vectorului mulțime. Pentru test se folosește masca `1 << 7-nb`.

P8_6. O expresie pătratică de o variabilă are forma: ax^2+bx+c în care numerele *a, b, c* (coeficienții) au valori fixate, iar variabila `x` poate lua diferite valori.
Specificați, proiectați și implementați clasa `Parabola`, care poate păstra informații asupra unei expresii pătratice.
Un constructor implicit setează cei 3 coeficienți la zero.
Se vor prevedea funcții membru pentru:
- schimbarea coeficienților
- aflarea valorii curente a coeficienților
- evaluarea unei expresii pătratice pentru un x dat
- determinarea numărului de rădăcini reale a unei expresii pătratice

Se vor redefini operatorii + și * ca funcții nemembri pentru:
- adunarea a două expresii pătratice:

```
Parabola operator +(const Parabola &p1, const Parabola &p2);
```

- înmulțirea unei expresii pătratice cu o constantă

```
Parabola operator *(double k, const Parabola &p);
```


P8_7. Proiectați și implementați clasa `Vector` care să permită lucrul cu vectori de elemente reale.
Constructorul clasei va avea un argument - dimensiunea vectorului și va aloca memorie pentru vector (în lipsa argumentului se ia implicit dimensiunea 10)
Se va asigura un destructor și un constructor de copiere.
Se vor prevedea funcții membri pentru:
- determinarea dimensiunii vectorului
- determinarea lungimii vectorului
- redefinirea operatorilor `+=` , `-=`, pentru adunarea și scăderea vectorului cu un alt vector dat ca argument

Se va redefini operatorul >> ca funcție prieten pentru citirea unui vector de la intrarea standard
Se vor asigura funcții nemembru pentru:
- testul de egalitate a doi vectori (redefinirea operatorului ==)
- scrierea unui vector la ieșirea standard (redefinirea operatorului <<)
- redefinirea operatorilor + , −, pentru a permite operații cu două argumente vectori
- redefinirea operatorului * pentru a permite calculul produsului scalar a doi vectori

P8_8. Proiectați și implementați clasa `Matrice` care să permită lucrul cu matrice pătrate de elemente reale.
Constructorul clasei va avea un argument - dimensiunea, adică numărul de linii și de coloane al matricei, va aloca memorie pentru matrice (în lipsa argumentului se ia implicit dimensiunea 10) și va permite accesul la elementele individuale prin indexare.
Se va asigura un destructor și un constructor de copiere.
Se vor prevedea funcții membri pentru:
- determinarea dimensiunii matricei
- calculul determinantului matricii
- redefinirea operatorilor `+=`, `-=` , `*=` pentru adunarea, scăderea și înmulțirea unei matrici cu o altă matrice dată ca argument
- redefinirea operatorului `/=` pentru calculul inversei matricei

Se va redefini operatorul >> ca funcție prieten pentru citirea unei matrici de la intrarea standard
Se vor asigura funcții nemembru pentru:
- testul de egalitate a două matrice (redefinirea operatorului ==)
- scrierea unei matrice la ieșirea standard (redefinirea operatorului <<)
- redefinirea operatorilor +, − , * pentru a permite operații cu două argumente matrice.

P8_9. Considerăm clasa `Forma` cu derivarea `Forma` ← `Dreptunghi`, `Forma` ← `Cerc`

```
class Forma {
protected:
  double x,y;
public:
  Forma(double h=0, double v=0);
  virtual double Arie() const;
  virtual double Perimetru()const=0;
};
class Dreptunghi : public Forma {
public:
  Dreptunghi(double h=0, double v=0);
  virtual double Arie() const;
  virtual double Perimetru()const;
};
class Cerc : public Forma {
protected:
  double raza;
public:
  Cerc(double h=0, double v=0, double r=0);
  virtual double Arie() const;
  virtual double Perimetru()const;
};
```

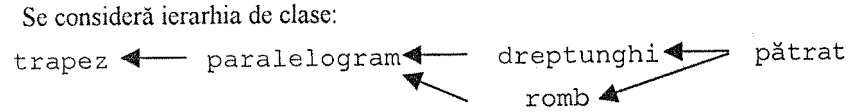
Definiți și implementați funcțiile din cele 3 clase.

P8_10. Considerăm derivarea

`Dreptunghi` ← `Paralelipiped`.

Implementați constructori pentru clasele respective și funcțiile `Arie()` și `Volum()`, pentru `Dreptunghi` se ia volumul 0.

P8_11. Se consideră ierarhia de clase:



Toate figurile au două laturi paralele cu axa Ox.

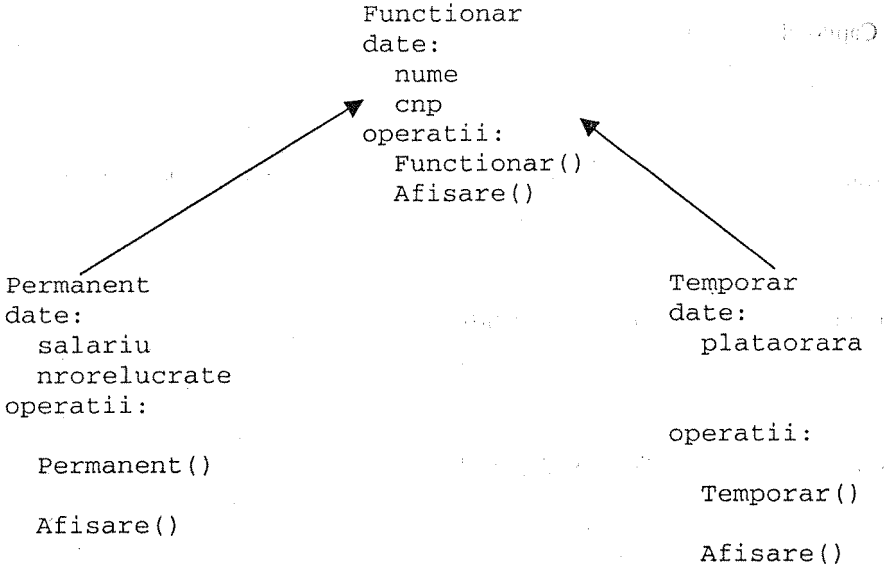
Pătratul are ca date membri coordonatele colțului stânga-jos și lungimea laturii, dreptunghiul are, în plus, lungimea celeilalte laturi, rombul adaugă la membrii date – coordonatele colțului opus, iar trapezul are ca membru

suplimentar – cea de-a doua bază. Funcțiile membri conțin, în afara constructorilor, funcții pentru calculul ariei și perimetrului.

O dată membru - `valid` , are valoarea 1 dacă figura respectivă este specificată corect. Constructorii verifică paralelismul laturilor.

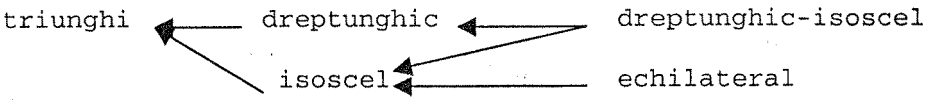
Definiți și implementați ierarhia de clase.

P8_12. Considerăm derivarea:



Funcția `Afisare()` din clasa de bază tipărește `nume` și `cnp`, iar în clasele derivate se tipărește în plus `salariul`. Definiți și implementați aceste clase.

P8_13. Definiți și implementați ierarhia de clase:



Clasele `echilateral` și `dreptunghic-isoscel` au ca membru dată o latură (cateta pentru cel de-al doilea). Clasele `dreptunghic` și `isoscel` au suplimentar a doua latură (cateta cealaltă, respectiv baza), iar clasa `triunghi` are cea de-a treia latură. Definiți constructori și funcții pentru calculul perimetrului și ariei.

Tabla de materii