- Să se scrie funcţia pentru aproximarea valorii unei integrale, definită prin funcţia f(x), pe un interval dat, prin metoda trapezelor.

Funcţia are ca parametri de intrare capetele intervalului pe care este definită integrala (*a* şi *b*), numărul de diviziuni ale intervalului (*n*) şi adresa funcţiei care se integrează (*f*). Funcţia returnează, prin numele ei, valoarea aproximativă a integralei. Cu cît numărul de diviziuni este mai mare (lungimea unui subinterval mai mică) cu atît mai bună este aproximarea.

```
double trapez(double a,double b,int n,double (*f)(double))
{ double h,i;
  int j;
  h=(b-a)/n; i=0.0;
  for(j=0;j<=n;j++)
    i+=(*f)(a+j*h);
  i*=h; return i;}
```

- Să se scrie funcţia pentru determinare celui mai mare divizor comun dintre 2 numere naturale.

Funcţia are ca parametri de intrare cele două numere (*a* şi *b*) şi returnează, prin numele ei, valoarea celui mai mare divizor comun.

- *varianta recursivă:*

```
long cmmdc(long a,long b)
{ long c;
  if(a==b) c=a;
  else if(a>b) c=cmmdc(a-b,b);
       else c=cmmdc(a,b-a);
  return c;}
```

- *varianta iterativă:*

```
long cmmdc(long a,long b)
{ long r,d=a,i=b;
  do {r=d%i; d=i; i=r;}
  while(r!=0);
  return d;}
```

- Să se scrie o funcţie eficientă pentru ridicarea unui număr la o putere naturală.

Funcţia are ca parametri baza (*b*) şi exponentul (*e*) şi returnează, prin numele ei, valoarea cerută.

-*varianta iterativă:*

```
long putere(int b,int e)
{ long p=1;
  while(e)
    if(e%2) {p*=b;e--;}
    else {b*=b; e/=2;}
  return p;}
```

-*varianta recursivă:*

```
long putere(int b,int e)
{ long p;
  if(!e) p=1;
  else if(e%2) p=b*putere(b,e-1);
       else p=putere(b,e/2)*putere(b,e/=2);
  return p;}
```

- Să se scrie funcţia pentru calcularea sumei elementelor unui masiv tridimensional. Să se folosească diferite variante pentru transmiterea masivului ca parametru.

Funcţia are ca parametri de intrare masivul tridimensional (*a*) şi dimensiunile sale efective (*m*, *n*, *p*). În prima variantă toate cele trei dimensiuni sînt precizate. În a doua variantă numărul de plane este omis (facilitate permisă în C). În a treia variantă se trimite un pointer spre o matrice. Cele trei variante de transmitere a masivului sînt echivalente.

```
int s1(int a[3][3][3],int m,int n,int p)
{ int s=0,i,j,k;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      for(k=0;k<p;k++)
        s+=a[i][j][k];
  return(s);}
int s2(int a[][3][3],int m,int n,int p)
{ int s=0,i,j,k;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      for(k=0;k<p;k++)
        s+=a[i][j][k];
  return(s);}
int s3(int (*a)[3][3],int m,int n,int p)
{ int s=0,i,j,k;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      for(k=0;k<p;k++)
        s+=a[i][j][k];
  return(s);}
```

- Să se scrie funcţia pentru afişarea conţinutului binar al unei zone de memorie în care se află memorat un şir de caractere.

Funcţia are ca parametru de intrare adresa şirului de caractere care trebuie afişat şi foloseşte o mască pentru a selecta fiecare bit al fiecărui caracter.

```
void bin(char *s)
{ unsigned char masca;
  while(*s)
   {masca=128;
    while(masca)
     {if(*s&masca)putch('1');
      else putch('0');
      masca>>=1;}
    s++;
    printf("\n");}}
```

- Să se scrie funcţia pentru aproximarea valorii soluţiei unei ecuaţii algebrice transcendente prin metoda bisecţiei.

Funcţia are ca parametri de intrare capetele intervalului în care se caută soluţia (*x0* şi *x1*), numărul maxim de iteraţii (*n*), precizia dorită (*eps*), funcţia asociată ecuaţiei (*f*) şi adresa unde se va înscrie soluţia. Prin numele funcţiei se returnează un cod de eroare cu următoarea semnificaţie: 0 – nu s-a găsit soluţie datorită numărului prea mic de iteraţii sau preciziei prea mari cerute; 1 – s-a obţinut soluţia exactă; 2 – s-a obţinut o soluţia aproximativă; 3 – intervalul dat nu conţine nici o soluţie.

-varianta iterativă

```
int bisectie(float x0,float x1,unsigned n,float eps,float
             (*f)(float), float *sol)
{ int cod=0;
  if ((*f)(x0)*(*f)(x1)>0) cod=3;
  else while((n)&&(!cod))
         {*sol=(x0+x1)/2;
          if((*f)(*sol)==0) cod=1;
          if(fabs(x0-x1)<=eps) cod=2;
```

```
          else {if((*f)(*sol)*(*f)(x0)<0)x1=*sol;
                else x0=*sol;
                n--;}}
  return cod;}
```

-varianta recursivă

```
int bisectie(float x0,float x1,unsigned n,float eps,float
             (*f)(float),float *sol)
{ int cod;
  if ((*f)(x0)*(*f)(x1)>0) cod=3;
  else if (n==0) cod=0;
       else {*sol=(x0+x1)/2;
             if((*f)(*sol)==0) cod=1;
             else if(fabs(x0-x1)<=eps) cod=2;
                  else {if((*f)(*sol)*(*f)(x0)<0)
                          cod=bisectie(x0,*sol,n-1,eps,f,sol);
                        else cod=bisectie(*sol,x1,n-1,eps,f,sol);}
  return cod;}
```

- Să se scrie funcţia pentru calculul lui *n!*, recursiv şi nerecursiv.
  Funcţia are ca parametru de intrare pe *n* şi returnează, prin numele ei, valoarea factorialului.

-varianta recursivă

```
long fact(long n)
{ long f;
  if (n==1)f=1;
  else f=n*fact(n-1);
  return(f);}
```

-varianta iterativă

```
long fact(long n)
{long f=1;
 for(long i=1;i<=n;i++)
   f*=i;
 return(f);}
```

- Să se scrie funcţia pentru calcularea termenului de ordin *n* al şirului Fibonacci, recursiv şi nerecursiv.

Funcţia are ca parametru de intrare indicele termenului pe care trebuie să îl calculeze şi returnează, prin numele ei, valoarea cerută. Indicii termenilor şirului încep de la 1.

-varianta iterativă

```
long fib(int n)
{ long f,a,b;
  int i;
  if ((n==1)||(n==2))f=1;
  else {a=1;b=1;
        for(i=3;i<=n;i++)
         {f=a+b;
          a=b;b=f;}
       }
  return(f);}
```

-varianta recursivă

```
long fib(int n)
{ long f;
  if ((n==1)||(n==2))f=1;
  else f=fib(n-1)+fib(n-2);
  return(f);}
```

- Să se scrie funcţia pentru citirea unui vector de la tastatură.

Funcţia nu are parametri de intrare. Parametrii de ieşire sînt vectorul şi numărul de elemente, pentru care se simulează transferul prin adresă.

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

void citire(int v[],int* n)
{ int i,er, p;
  printf("\nn=");scanf("%d",n);
  for(i=0;i<*n;i++)
   {printf("v(%d)=",i);
    do {p=scanf("%d",&v[i]);}
    while(p!=1);}
}

void main()
{
	int n, v[50], i;
	printf("Introduceti elementele vectorului");
	citire(v, &n);
	printf("Vectorul este");
	for(i=0; i<n; i++)
		printf("v[%d]=%d\t", i, v[i]);
	_getch();
}
```

- Să se scrie funcţia pentru afişarea unui vector pe ecran.

Funcţia are ca parametri de intrare vectorul şi numărul de elemente.

```
void afisare(float v[],int n)
{ int i;
  printf("\n"); for(i=0;i<n;i++) printf("\t%5.2f",v[i]);}
```

- Să se scrie funcţia pentru găsirea elementului minim dintr-un vector.

Funcţia are ca parametri vectorul şi numărul de elemente şi returnează, prin numele ei, elementul minim.

```
float minim(float v[],int n)
{ float m; int i;
  m=v[0];
  for(i=0;i<n;i++)
    if(m>v[i])m=v[i];
  return(m);}
```

- Să se scrie funcţia pentru găsirea elementului minim şi a primei poziţii de apariţie a acestuia într-un vector.

Funcţia are ca parametri vectorul, numărul de elemente şi adresa unde se va reţine prima poziţie de apariţie a minimului. Prin numele funcţiei se returnează valoarea minimului.

```
float minim(float v[],int n,int *poz)
{ float m;
  m=v[0];
  *poz=0;
  int i;
  for(i=0;i<n;i++)
    if(m>v[i]){m=v[i];
               *poz=i;}
  return(m);}
```

- Să se scrie funcţia pentru găsirea elementului minim şi a ultimei poziţii de apariţia a acestuia într-un vector.

Funcţia are ca parametri vectorul, numărul de elemente şi adresa unde se va reţine ultima poziţie de apariţie a minimului. Prin numele funcţiei se returnează valoarea minimului.

```
float minim(float v[],int n,int *poz)
{ float m;
  m=v[0];
  *poz=0;
  int i;
  for(i=0;i<n;i++)
    if(m>=v[i]){m=v[i];
                *poz=i;}
  return(m);}
```

- Să se scrie funcţia pentru găsirea elementului maxim şi a tuturor poziţiilor sale de apariţie într-un vector.

Funcţia are ca parametri vectorul, numărul de elemente, vectorul unde se vor reţine poziţiile maximului şi adresa unde se va scrie numărul de apariţii ale maximului.

```
float minim(float v[],int n,int poz[],int *nrpoz)
{ float m;
  int i;
  m=v[0]; poz[0]=0;
  *nrpoz=1;
  for(i=1;i<n;i++)
    if(m>v[i]) {m=v[i];
                poz[0]=i;
                *nrpoz=1;}
    else if(m==v[i]) {poz[*nrpoz]=i;
                      (*nrpoz)++;}
  return(m);}
```

- Să se scrie funcţia pentru inserarea unui 0 între fiecare două elemente ale unui vector.

Funcţia are ca parametri vectorul şi adresa numărului de elemente ale sale. La adresa respectivă se va reţine noul număr de elemente rezultat în urma prelucrării. Nu se obţine nici un rezultat prin numele funcţiei.

```
void inserare(float v[],int* n)
{ int i,j,k;
  k=*n;
  for(i=0;i<k-1;i++)
   {for(j=*n;j>2*i+1;j--) v[j]=v[j-1];
    v[2*i+1]=0;
    (*n)++;}
}
```

- Să se scrie funcţia pentru crearea unui vector din elementele unui vector dat, inserînd cîte un 0 între fiecare 2 elemente ale acestuia.

Funcţia are ca parametri vectorul iniţial şi numărul său de elemente, vectorul rezultat şi adresa unde se va scrie numărul de elemente ale vectorului rezultat. Nu se întoarce nici o valoare prin numele funcţiei.

```
void inserare(float v[],int n,float v1[],int* n1)
{ int i;
  *n1=0;
  for(i=0;i<n-1;i++)
   {v1[2*i]=v[i];
    v1[2*1+1]=0;
    (*n1)+=2;}
    v1[*n1]=v[n-1];
 (*n1)++;}
```

- Să se scrie funcţia pentru compactarea unui vector prin eliminarea dublurilor.

Funcţia are ca parametri vectorul şi adresa unde se află numărul de elemente ale acestuia. La această adresă se va înscrie numărul de elemente rămase după compactare. Funcţia întoarce, prin numele ei, numărul de elemente rămase în vector.

```
int compactare(float v[],int *n)
{ int i,j,k;
  for(i=0;i<*n-1;i++)
    for(j=i+1;j<*n;j++)
      if(v[i]==v[j])
        {for(k=j;k<*n-1;k++)
           v[k]=v[k+1];
         (*n)--;
         j--; }
  return(*n);}
```

- Să se scrie funcţia pentru crearea unui vector din elementele unui vector dat, fără a lua în considerare dublurile.

Funcţia are ca parametri vectorul, numărul său de elemente, vectorul care se va construi, adresa unde se va înscrie numărul de elemente ale vectorului rezultat. Nu se întoarce nici o valoare prin numele funcţiei.

```
void compactare(float v[],int n,float v1[],int *n1)
{ int i,j,k;
  *n1=0;
  for(i=0;i<n;i++)
    {k=0;
     for(j=0;j<*n1;j++)
       if(v[i]==v1[j]) k=1;
     if(!k)
      {v1[*n1]=v[i];
       (*n1)++;}
   }
}
```

- Să se scrie funcţia pentru inversarea ordinii elementelor unui vector.

Funcţia are ca parametri vectorul şi numărul său de elemente. Nu se întoarce nici un rezultat prin numele funcţiei.

```
void inversare(float v[],int n)
{ int i, j;
  float a;
  i=0; j=n-1;
  while(i<j)
  {a=v[i];
   v[i]=v[j];
   v[j]=a;
   i++;
   j--;}
}
```

- Să se scrie funcţia pentru calcularea amplitudinii elementelor unui vector.

Funcţia are ca parametri vectorul şi numărul de elemente şi întoarce, prin numele ei, amplitudinea elementelor.

```
float amplitudine(float v[],int n)
{ int i;
  float min,max;
  min=v[0];
  max=v[0];
  for(i=0;i<n;i++)
```

```
    if(v[i]<min) min=v[i];
    else if(v[i]>max) max=v[i];
  return(max-min);}
```

- Să se scrie funcţia pentru calcularea mediei aritmetice a elementelor unui vector.

Funcţia are ca parametri vectorul şi numărul de elemente şi întoarce, prin numele, ei media aritmetică a elementelor.

```
float mediaa(float v[],int n)
{ int i;
  float s;
  s=0;
  for(i=0;i<n;i++)
    s+=v[i];
  return(s/n);
}
```

- Să se scrie funcţia pentru calcularea mediei armonice a elementelor nenule ale unui vector.

Funcţia are ca parametri vectorul, numărul de elemente şi adresa unde va scrie parametrul de eroare şi întoarce, prin numele ei, media armonică a elementelor nenule. Parametrul de eroare este 1 dacă nu se poate calcula media şi 0 în caz contrar.

```
float mediaarm(float v[],int n,int *er)
{ int i,m;
  float s;
  s=0;m=0;*er=0;
  for(i=0;i<n;i++)
    if(v[i]!=0){ s+=1/v[i];
                 m++;}
  if(s=0) *er=1;
  else s=m/s;
  return(s);
}
```

- Să se scrie funcţia pentru calcularea abaterii medii pătratice a elementelor unui vector.

Funcţia are ca parametri vectorul şi numărul de elemente şi apelează funcţia pentru calculul mediei aritmetice a elementelor vectorului. Valoarea abaterii medii pătratice este returnată prin numele funcţiei.

```
float abatere(float v[],int n)
{float m,s;
 int i;
 m=mediaa(v,n);
 s=0;
 for(i=0;i<n;i++)
   s+=(v[i]-m)*(v[i]-m);
 return(s);
}
```

- Să se scrie funcţia pentru calculul sumei a două polinoame.

Funcţia are ca parametri gradul primului polinom şi vectorul coeficienţilor săi, gradul celui de al doilea polinom şi vectorul coeficienţilor săi, vectorul în care se vor scrie coeficienţii polinomului rezultat şi adresa la care se va scrie gradul polinomului rezultat.

```
void s_polinom(int n,float a[],int m,float b[],float r[],int* p)
{ int i;
  *p=m>n?m:n;
  for(i=0;i<=*p;i++)
    r[i]=(i>n?0:a[i])+(i>m?0:b[i]);}
```

- Să se scrie funcţia pentru calcul produsului dintre două polinoame.

Funcţia are ca parametri gradul primului polinom, vectorul cu coeficienţii săi, gradul celui de al doilea polinom, vectorul cu coeficienţii săi, vectorul în care se vor scrie coeficienţii polinomului rezultat şi adresa la care se va scrie gradul polinomului rezultat.

```
void p_polinom(int n,float a[],int m,float b[],float r[],int* p)
{ int i,j,tt;
  float t[100];
  *p=0; tt=0;
  for(i=0;i<=n;i++)
   {tt=m+i;
    for(j=0;j<=m;j++)  t[j+i]=b[j]*a[i];
    for(j=0;j<i;j++) t[j]=0;
    s_polinom(tt,t,*p,r,r,p);
    getch();}
}
```

- Să se scrie funcţia pentru construirea unei noi matrice cu liniile şi coloanele unei matrice care nu conţin o anumită valoare.

Funcţia are ca parametri matricea iniţială şi dimensiunile sale (numărul linii-lor şi numărul coloanelor), noua matrice (adresa unde vor fi scrise elementele sale), adresele unde se vor scrie dimensiunile sale, numărul liniilor şi numărul de coloane-lor. Sînt apelate funcţiile *compactare* (exerciţiul 2.1.ix) şi *căutare* (exerciţiul 2.1.xx).

```
void nenule(float a[][20],int m,int n,float b[][20], int *p,int *q)
{ int i,j,k,r,s,l[20],c[20];
  k=0;
  r=0;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      if(!a[j][i])
       {l[k++]=i;
        c[r++]=j;}
  compactare(l,&k);
  compactare(c,&r);
  *p=0;*q=n;
  for(i=0;i<m;i++)
    if(cautare(l,k,i)==-1)
      {for(j=0;j<n;j++)
         b[*p][j]=a[i][j];
       (*p)++;}
  for(j=0;j<n;j++)
    if(cautare(c,r,j)!=-1)
      {for(s=j;s<n-1;s++)
         for(i=0;i<*p;i++)
           b[i][s]=b[i][s+1];
       n--;}
  *q=n;}
```

- Să se scrie funcţia pentru ridicarea la putere a unei matrice pătrate.

Funcţia are ca parametri matricea iniţială, dimensiunea ei, puterea la care se ridică şi matricea în care va scrie rezultatul. Sînt folosite funcţiile *copiere* (pentru copierea unei matrice în alta) şi *produs* (pentru înmulţirea a două matrice – exerciţiul 2.4.x).

```
void copiere(float a[][20],int m, float b[][20])
{ int i,j;
  for(i=0;i<m;i++)
    for(j=0;j<m;j++)
      b[i][j]=a[i][j];}

void putere(float a[][20],int m, int p,float b[][20])
{ int i,j,k;
  float c[20][20];
  for(i=0;i<m;i++)
    for(j=0;j<m;j++)
      c[i][j]=(i==j);
  for(i=0;i<p;i++)
   {produs(c,m,m,a,m,m,b,&m,&m);
    copiere(b,m,c);}
}
```

- Să se scrie funcţia pentru rezolvarea unui sistem algebric liniar de *n* ecuaţii cu *n* necunoscute, calculînd în acelaşi timp inversa matricei sistemului şi determinantul acesteia.

Funcţia are ca parametri matricea sistemului, gradul sistemului, vectorul termenilor liberi, limita sub care pivotul este considerat *0* (pivot 0 înseamnă o coloană nulă deci matrice neinversabilă şi sistem cu o infinitate de soluţii), matricea în care se va scrie inversa matricei sistemului şi vectorul în care se va scrie soluţia sistemului. Prin numele funcţiei se întoarce valoarea determinantului, care are şi rol de parametru de eroare (deteminant nul înseamnă matrice neinversabilă).

```
float inversa(float a[][20],int n,float b[],float eps,
              float inv[][20],float x[])
{ float c[20][20],e[20][20],d,aux;
  int i,j,k,p;
  d=1;                          //construire matrice de lucru
  for(i=0;i<n;i++)
   {for(j=0;j<n;j++)
     {c[i][j]=a[i][j]; c[i][j+n]=(i==j);}
    c[i][2*n]=b[i];}
  afisare(c,n,2*n+1);
  i=0;
  while((i<n)&&d)               //pivotare si calcul determinant
   {p=i;//cautare pivot pe coloana i
    for(j=i+1;j<n;j++)
      if(abs(c[j][i])>abs(c[p][i]))p=j;
    if(abs(c[p][i])<eps)d=0;    //aducere pivot in pozitie
    else{if(p!=i){aux=c[p][i];c[p][i]=c[i][i];c[i][i]=aux;d=-d;}
         d=d*c[i][i];
    for(j=0;j<n;j++) e[j][i]=0; //pivotare
    for(j=i;j<2*n+1;j++) e[i][j]=c[i][j]/c[i][i];
    for(j=0;j<n;j++)
      for(k=i;k<2*n+1;k++)
        if(j!=i)e[j][k]=(c[j][k]*c[p][i]-c[j][i]*c[i][k])/c[p][i];
    for(k=0;k<n;k++)
      for(j=i;j<2*n+1;j++)
        c[k][j]=e[k][j];}
    i++;}
  for(i=0;i<n;i++)              //separare rezultate
   {for(j=0;j<n;j++)
      inv[i][j]=c[i][j+n];
    x[i]=c[i][2*n];}
  return d;}
```

- Să se scrie funcţia pentru interclasarea elementelor a doi vectori sortaţi crescător.

Funcţia are ca parametri primul vector, numărul său de elemente, al doilea vector, numărul său de elemente, vectorul în care va scrie rezultatul şi adresa la care va scrie numărul de elemente din vectorul rezultat. Nu se întoarce nici un rezultat prin numele funcţiei.

```
void interclasare(float v[],int m,float w[],int n,float r[],int* p)
{int i,j;
 i=0;j=0;*p=0;
 while((i<m)&&(j<n))
   if(v[i]<w[j])
     r[(*p)++]=v[i++];
   else
     r[(*p)++]=w[j++];
 if(i==m)  for(i=j;i<n;i++)
             r[(*p)++]=w[i];
 else      for(j=i;j<m;j++)
             r[(*p)++]=v[j];
}
```

- Să se scrie funcţia pentru calcularea produsului scalar dintre doi vectori.

Funcţia are ca parametri cei doi vectori şi numărul de elemente ale fiecăruia şi adresa unde va scrie parametrul de eroare. Prin numele funcţiei se întoarce produsul scalar. Parametrul de eroare are valoarea 0, dacă se calculează produsul, sau 1, dacă vectorii au lungimi diferite.

```
float prod_scal(float v[],int n,float v1[],int n1,int *er)
{ float p;
  int i;
  if(n1!=n)*er=1;
  else{*er=0;
       p=0;
       for(i=0;i<n;i++)
         p+=v[i]*v1[i];}
  return(p);}
```

- Să se scrie funcţia pentru calcularea produsului vectorial dintre doi vectori.

Funcţia are ca parametri cei doi vectori, numărul de elemente ale fiecăruia şi vectorul în care va scrie rezultatul. Prin numele funcţiei se întoarce parametrul de eroare. Parametrul de eroare are valoarea 0, dacă se calculează produsul, sau 1, dacă vectorii au lungimi diferite.

```
int prod_vect(float v[],int n,float v1[],int n1,float r[])
{ int i,er;
  if(n1!=n)er=1;
  else{er=0;
       for(i=0;i<n;i++)
       r[i]=v[i]*v1[i];}
  return(er);}
```

- Să se scrie funcţia pentru căutarea unui element într-un vector nesortat.

Funcţia are ca parametri vectorul, numărul de elemente şi valoarea căutată. Prin numele funcţiei se întoarce poziţia primei apariţii a elementului în vector sau –1 dacă elementul nu este găsit.

```
int cautare(float v[],int n,float x)
{ int i,er;
  er=-1;
  for(i=0;i<n;i++)
    if((v[i]==x)&&(er==-1)) er=i;
  return(er);}
```

- Să se scrie funcţia pentru căutarea unui element într-un vector sortat.

a) *Varianta iterativă*: funcţia are ca parametri vectorul, numărul de elemente şi valoarea căutată. Prin numele funcţiei se întoarce poziţia elementului găsit sau –1, dacă elementul nu a fost găsit.

```
int cautare_bin(float v[],int n,float x)
{ int i,j,er,p;
  er=-1;
  i=0;j=n-1;
  while((i<=j)&&(er==-1))
```

```
   {p=(i+j)/2;
    if(v[p]==x) er=p;
    else if(v[p]<x)i=p+1;
         else j=p-1;}
 return(er);}
```

b) *Varianta recursivă*: funcţia are ca parametri vectorul, capetele intervalului în care face căutarea (iniţial *0* şi *n-1*) şi valoarea căutată. Prin numele funcţiei se întoarce poziţia elementului găsit sau –1, dacă elementul nu a fost găsit.

```
int cautare_bin_r(float v[],int s,int d,float x)
{ int i,j,er,p;
  p=(s+d)/2;
  if(s>d)er=-1;
  else if(v[p]==x) er=p;
       else if(v[p]<x) er=cautare_bin_r(v,p+1,d,x);
            else       er=cautare_bin_r(v,s,p-1,x);
  return(er);}
```

- Să se scrie funcţia pentru determinarea numerelor naturale prime mai mici decît o valoare dată (maxim 1000) prin metoda ciurului lui Eratostene.

Funcţia are ca parametri limita maximă, vectorul în care va scrie numerele prime mai mici decît acea limită şi adresa unde va scrie numărul de numere găsite. Parametrul de eroare (1 dacă limita este mai mare de 1000, 0 dacă nu sînt erori) este întors prin numele funcţiei.

```
int eratostene(int x,int v[],int *y)
{ int i,er,q,v1[1000];
  if(x>500)er=1;
  else{er=0;
       for(i=0;i<x;i++) v1[i]=i;
         for(i=2;i<=sqrt(x);i++)
           {q=2*i;
            while(q<x)
              {v1[q]=0; q+=i;}
           }
       *y=0;
       for(i=0;i<x;i++)
         if(v1[i]) v[(*y)++]=v1[i];}
  return er;}
```

- Să se scrie funcţia pentru determinarea valorii unui polinom într-un punct dat.

Funcţia are ca parametri gradul polinomului *n*, vectorul coeficienţilor *a* (în ordine, primul coeficient fiind cel al termenului liber, în total *n+1* elemente) şi punctul în care se calculează valoarea polinomului. Prin numele funcţiei se întoarce valoarea calculată.

```
float polinom(int n,float a[],float x)
{ int i;
  float p;
  p=a[n];
  for(i=n;i>0;i--)
    p=p*x+a[i-1];
  return p;}
```