Programarea calculatoarelor
Limbajul C

CURS 4

Exemple tablouri

Exemplu matrice

```
#define N 10
#define M 10

int main(){
    int matrice[N][M];
     int i, j, n, m;
    //citire cu validare a ordinului matricii
     do {
            printf ("Introduceti nr de linii (1..%d) si de coloane:
                        (1..%d)“, N,M);
            fflush(stdin);
            scanf ("%d%d", &n, &m);
      } while( n<0 || n>N || m<0 || m>M);
```

Exemplu matrice - continuare

```
  for ( i=0; i<n; i++ )
     for ( j=0; j<m; j++ )
     {
           printf( "elem[%d][%d]:“ ,i ,j );
           scanf( "%d", &matrice[i][j] );
     }


  printf("Matricea:\n");
  for ( i=0; i<n; i++ ){
     for ( j=0; j<m; j++ )
           printf ( "%d\t“ ,matrice[i][j] );
     printf (“\n”);
  }
 return 0;
}
```

Şiruri de caractere

- Declarare şiruri
  - #define MAX_SIR 100
  - char s[MAX_SIR];

  sau:
  - char s[100];
- determinarea lungimii şirului

```
lg =0;
while (s[lg]!='\0')
   lg++;
```

Exerciții

- Testarea proprietății de palindrom

```
i=0;
j=lg-1;
while (s[i]==s[j] && i<j){
    i++;
    j--;
}
if (i >=j)
    printf("\nSirul este palindrom.\n");
else
    printf("\nSirul nu e palindrom.\n");
```

Exerciții

1. Program pentru căutarea secvențiala într-un vector a unei valori (prima şi ultima apariție).
2. Program pentru citirea unui vector de întregi şi extragerea elementelor distincte într-un al doilea vector, care se va afişa.
3. Program pentru calculul normei pe linii a unei matrice de numere reale. Norma este valoarea maximă dintre sumele valorilor absolute ale elementelor din fiecare linie.

Căutarea secvențiala într-un vector a unei valori (prima şi ultima apariție).

```
#include<stdio.h>
#include<stdlib.h>
#define N 10

int main(){
      int v[N], x, i, n, prim,ultim;
      printf("Introduceti nr de elemente:");
      scanf("%d", &n);
      for ( i = 0; i < n; i++) {
          printf("elem[%d]=",i);
          scanf("%d", &v[i]);
      }
      printf("Introduceti elem cautat:");
      scanf("%d", &x);
```

Continuare exemplu

```
    prim=-1; ultim=-1;
    for ( i = 0; i < n; i++)
        if ( v[i]==x )  {
                        prim=i;
                        break;
        }
    for ( i = n-1; i >= 0; i--)
        if ( v[i]==x ) {
                        ultim=i;
                        break;
       }
    if ( prim!= -1 ) printf ("prima aparitie in pozitia %d\n", prim);
    else printf("elem nu exista in sir!\n");
    if ( ultim!= -1 ) printf ("ultima aparitie in pozitia %d\n", ultim);
    return 0;
}
```

Extragere elemente distincte dintr-un vector

```
#define MAX 30
#include <stdio.h>
void main () {
  int a[MAX];              /* un vector de intregi*/
  int b[MAX];              /* aici se pun elementele distincte din a*/
  int n, m, i, j, gasit;   /* n= dimensiune vector a,
                              m=dimensiune vector b*/

  printf("n="); scanf("%d",&n);      /* dimensiune vector*/
  printf ("%d numere intregi:\n",n);
     /* citire vector*/
  for (i=0;i<n;i++) scanf("%d",&a[i]);

  m=0;
```

Extragere elemente distincte dintr-un vector

```
    /* caută pe a[i] in vectorul b*/
  for (i=0;i<n;i++) {
    gasit=0;
    for (j=0;j<m && !gasit; j++)
      if (a[i]==b[j])  gasit=1;
    if (!gasit) {
      b[m++] =a[i];
    }
  }
    /* scrie vector b*/
  for (j=0;j<m;j++)
    printf ("%3d",b[j]);
}
```

Norma unei matrice

```
#include <math.h>
#include <stdio.h>
#define M 20                    /* dimensiuni maxime matrice*/
int main () {
  int nl, nc;                   /* nl linii, nc coloane*/
  float a[M][M] ;
  int i, j; float s, norma;
  printf("nr.linii: "); scanf("%d", &nl);
  printf("nr.coloane: "); scanf("%d", &nc);
  if (nl >M || nc >M) {
        printf("Eroare: dimensiuni >M \n"); return;
  }
  for (i=0; i<nl; i++)
       for (j=0; j<nc; j++)
         scanf ("%f", &a[i][j]);
```

Norma unei matrice

```
 norma=0;
  for (i=0; i<nl; i++) {
        s=0;
        for (j=0; j<nc; j++)
              s=s+ fabs (a[i][j]);
        if (norma < s)
              norma=s;
  }
      /* afisare matrice*/
  for (i=0; i<nl; i++) {
        for (j=0; j<nc; j++)
              printf ("%.2f ", a[i][j]);
        printf ("\n");
  }
  printf ("Norma este:%.2f \n", norma);
}
```

Programarea calculatoarelor
Limbajul C

CURS 4

Funcții

Modularizarea programelor

- Un program mare poate fi mai uşor de scris, de înțeles şi de modificat dacă este modular (format din module funcționale relativ mici = funcții)
- Funcția main: funcția principală a unui program C
- Funcție (în C) = piesa de bază a conceptului de modularitate al programării

Funcție C:

- poate executa o sarcină precisă.
- poate fi reutilizată în mai multe aplicații
  - reduce efortul de programare al unei noi aplicații.
- poate fi scrisă şi verificată separat de restul aplicației
  - reduce timpul de punere la punct a unei aplicații mari (deoarece erorile pot apare numai la comunicarea între funcții corecte).
- modificările se fac numai în anumite funcții şi nu afectează alte funcții (care nici nu mai trebuie recompilate)
  - întreținerea unei aplicații este simplificată

Exemplu

- Care ar fi funcțiile pentru operații cu mulțimi de numere întregi necesare pentru o implementare cât mai completă a acestora?
  - inițializare mulțime vidă
  - verificare apartenență la o mulțime
  - adăugare element la o mulțime (dacă nu există deja)
  - afişare mulțime (între acolade)
  - reuniune
  - intersecție
  - diferență de mulțimi, etc

Declarare şi implementare funcții

- *tip_rezultat_returnat nume(lista_parametrii_formali)*

  {

  declarare variabile locale

  instrucțiuni funcție

  }

- Lista parametrilor formali cuprinde declarația parametrilor formali, separați prin virgulă:

  tip_p1 nume_p1, tip_p2 nume_p2, ..., tip_pn nume_pn

- funcțiile nu pot fi definite încuibat (ca in Pascal)!

Tipul unei funcții

- dacă *tip_rezultat_returnat* este:
  - int, float, etc - funcția este întreagă, reală, etc adică funcție cu tip
  - void - funcția este void sau funcție fără tip

- dacă funcția nu returnează nici un rezultat şi nu primeşte parametri, definiția va fi:

```
void nume_funcție (void){
    definirea variabilelor locale
    prelucrari  /*instrucțiuni*/
}
```

Numărarea şi afişarea numerelor prime mai mici ca un întreg dat n.

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

int prim (int numar){
    int div;
    for (div=2; div<=sqrt(numar); div++)
        if (numar % div ==0) return 0;
    return 1;
}
```

Apel funcție

- pentru funcție fără tip (void):

  nume_funcție (lista_parametrii_efectivi);

- pentru funcție cu tip T, unde t este de tipul T:

  t = nume_funcție (lista_parametrii_efectivi);

  - *Sau poate apare într-o expresie în care se foloseşte rezultatul ei:*

  if(prim(m) == 1) …

- *parametri actuali=parametri efectivi*

Numararea si afisarea numerelor prime mai mici ca un întreg dat n.

```
int main () {
        int n,m,contor ;
            /* contor de numere prime*/
        printf ("n= "); scanf ("%d",&n);
        contor=0;
        for (m=2;m<=n;m++)   /* incearca numerele m*/
            if ( prim(m) == 1 ){       /* dacă m este prim*/
              printf ("%d\n",m);
              contor++;
            }
        printf ("există  %d numere prime mai mici decat %d\n", contor,n);
        return 0;
 }
```

Apel funcție

- La apelul unei funcții, se execută corpul său, după care se revine în funcția apelantă, la instrucțiunea următoare apelului.
- Parametrii efectivi sunt expresii, care trebuie să corespundă ca numar şi tipuri (eventual prin conversie implicită) cu parametrii formali.
- Parametrii efectivi sunt transmişi prin valoare, la apelul funcției (valorile lor sunt depuse pe stiva). Modificarea valorii lor de către funcție nu este vizibilă în exterior.

Prototipul unei funcții

- O funcție poate fi apelată, dacă în fața apelului există definiția sau cel putin declarația funcției (prototipul, antetul acesteia).
- Început fişier sursă sau fişier header:

  tip nume(lista_tipuri_parametrii_formali);
- În prototip, numele parametrilor formali pot fi omişi, apărând doar tipul fiecăruia.
- *Prototipul implicit* este:

  int nume_funcție(void);

Utilizare prototip funcții

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

int prim (int numar);
//sau: int prim (int);

int main(){
…
}

int prim (int numar){
      int div;
      for (div=2; div<=sqrt(numar); div++)
            if (numar % div ==0) return 0;
      return 1;
}
```

Apel funcție

- La apelul unei funcții, pe stivă se crează o *înregistrare de activare*, care cuprinde, de jos în sus:
  - adresa de revenire din funcție
  - valorile parametrilor formali
  - variabilele locale.

Return

- Revenirea dintr-o funcție se face la
  - întâlnirea instrucțiunii return,
  - terminarea execuției corpului funcției (funcția poate să nu conțină instrucțiunea return doar în cazul funcțiilor void - care nu returnează nici un rezultat).
- Forme ale instrucțiunii return:
  - *return;* //dacă funcția e void
  - *return expresie;* /*expresia e de acelaşi tip cu tip_rezultat (eventual prin conversie implicită) */

Parametri formali şi parametri efectivi

Exemplu:

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

int factorial (int n)          //n este parametru formal de tip întreg
{
   int i, fact=1;
   for (i=2; i<=n; i++)
            fact=fact*i;
   return fact;   //funcția factorial returnează valoarea lui n!
}
```

Parametri formali şi parametri efectivi

```
int main(void)
{
   int v;
   printf("3!=%d\n",factorial(3));
            /*în funcția printf apelam funcția factorial având ca parametru efectiv
               valoarea 3*/
   printf("Introd o valoare:");
   scanf("%d",&v);
   printf("%d!=%d\n",v,factorial(v));
            /*funcția factorial este apelata având ca parametru efectiv valoarea citita
               în variabila v*/

   return 1;
}
```

Problemă rezolvată

Se citeşte de la tastatură un număr natural n. Să se determine toate numerele prime mai mici decât n cu proprietatea că x = invers(x) (palindrom).

Ex: n=1000, x={2,3,5,7,11,101, 131,etc}

Se va rezolva în trei variante:

a) fără funcții auxiliare

b) se va utiliza o funcție de calcul a inversului unui număr.

c) se vor utiliza doua funcții: o funcție de calcul a inversului unui număr şi o funcție care testează dacă un număr este prim.

Rezolvare a

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(){
    int n,i,x,y,z;
    scanf("%d",&n);
    for(x=1;x<n;x++){
        for(i=2;i<=sqrt(x);i++)
                if(x%i==0) break;
```

Rezolvare a

```
    if(i>sqrt(x)) { //este prim
            y=0;
            z=x;
            while(z){
                    y=y*10+z%10;
                    z/=10;
            }
            if(y==x)  printf("%d ",x);
        }
    }
   return 0;
}
```

Rezolvare b

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int inv (int x){
    int y=0;
    while(x){
            y=y*10+x%10;
            x/=10;
            }
    return y;
}
```

Rezolvare b

```
int main(){
    int n,i,x;
    scanf("%d",&n);
    for(x=1;x<n;x++){
        for(i=2;i<=sqrt(x);i++)
                if(x%i==0) break;

        if(i>sqrt(x))
            if(inv(x)==x)  printf("%d ",x);

    }
    return 0;
}
```