PROGRAMAREA MODULARĂ

1. Conţinutul lucrării

În lucrare sunt prezentate conceptele de modul, programare modulară şi vizibilitate a variabilelor.

2. Consideraţii teoretice

2.1. Noţiunea de modul

Un modul conţine funcţii înrudite, în sensul că ele concură la rezolvarea unei subprobleme. De fapt modulele corespund subproblemelor rezultate în urma proiectării *top-down* a unei probleme complexe.

Programarea modulară este stilul de programare care are la bază utilizarea de module. Prin acest stil de programare se poate pune în valoare posibilităţile de "ascundere" a datelor şi funcţiilor împotriva unor accese neautorizate din alte module. Astfel, datele statice declarate în afara funcţiilor modulului, pot fi utilizate în comun de acestea, dar nu pot fi accesate de către funcţiile din alte module.

Recomandarea care se face în scrierea unui program complex este de a-l modulariza, permiţând astfel lucrul în echipă. Modularizarea unui program duce la punerea la punct a programului (implementarea şi testarea) mai rapidă. Modulele puse la punct pot fi utilizate ulterior pentru rezolvarea altor probleme.

Modulul este împărțit în două părți:

- partea publică (fișierul *header* cu extensia *.h*)
- partea privată (fișierul sursă cu extensia *.c*)

În partea publică (fișierul *.h*) se scriu declarații de constante, tipuri, variabile globale, prototipuri de funcții care pot fi utilizate în afara modulului. Fișierul *header* al modulului va conține aceste declarații între directive de preprocesare specifice care previn încărcarea lor multiplă în procesul de compilare. Fișierul *header* al modulului trebuie inclus în fiecare program care depinde de modulul respectiv (folosește funcții sau structuri de date definite în cadrul acestuia).

În partea privată (fișierul *.c*) are loc implementarea funcțiilor declarate în fișierul *header*. În cadrul acestuia trebuie să se includă mai întâi bibliotecile necesare implementărilor și apoi să se includă propriul fișier *header* al modulului. Includerea propriului fișier *header* asigură că prototipurile funcțiilor sunt verificate cu antetul funcțiilor actuale care urmează să fie implementate și permite compilatorului să genereze mesaje de eroare în cazul în care acestea nu coincid (ex. din cauza numărului de parametri diferit, din cauza tipului diferit al parametrilor).

Pentru rezolvarea unor probleme complexe, se scriu mai multe module (perechi de fişiere header şi sursă) sau se refolosesc anumite module existente. În fișierul sursă *main.c* se includ la început fișierele *header* corespunzătoare modulelor care se folosesc, iar apoi în cadrul funcției *main* se apelează funcțiile din module care conduc la rezolvarea problemei respective. În procesul de compilare fiecare modul sursă este compilat separat, rezultând mai multe module obiect (fișiere cu extensia .o). Este compilat și *main.c* rezultând *main.o*. Se *link-editează* aceste module obiect împreună cu *main.o* obținându-se fișierul executabil (cu extenisa *.exe*) care poate fi rulat pentru rezolvarea problemei respective.

2.2. Domeniul de valabilitate al variabilelor

2.2.1. Variabile globale şi externe

Variabilele globale sunt definite la începutul unui fişier sursă, deci înaintea primei funcţii. Ele sunt variabile vizibile din locul respectiv până la sfârşitul fişierului sursă respectiv. Dacă programul are mai multe fişiere sursă, o variabilă globală definită într-un fişier sursă poate fi utilizată în celelalte, dacă este declarată ca externă. Declararea unei variabile externe se poate face:

- după antetul unei funcţii, caz în care variabila globală este valabilă numai în acea funcţie;
- la începutul fişierului sursă, adică înaintea primei funcţii, caz în care este valabilă pentru toate funcţiile din acel fişier.

Observaţie: Se recomandă ca variabilele externe să fie declarate în fiecare funcţie unde se utilizează, evitând erorile care pot apărea prin mutarea ulterioară a unei funcţii în alt modul.

Variabilele globale sunt alocate la compilare, într-o zonă de memorie specială.

2.2.2.Variabilele locale

Variabilele declarate într-o funcţie sau într-o instrucţiune compusă au valabilitate numai în unitatea declarată. Ele pot fi:

a) **automatice** – care sunt alocate pe stivă la execuţie. Ele îşi pierd existenţa la revenirea din funcţie sau la terminarea instrucţiunii compuse. Declararea lor este cea obişnuită. Exemplu:
int a,b,c;

b**) statice** - care sunt alocate la compilare într-o zonă specială. Declararea se face cu ajutorul cuvântului cheie **static** înaintea tipului variabilei. Exemplu:
static int x,y,z;

Declararea unei variabile statice poate fi făcută:

- la începutul fişierului sursă (deci înaintea primei funcţii); în acest caz variabila statică respectivă este valabilă în tot fişierul sursă respectiv, dar nu poate fi declarată ca externă în alte fişiere;
- în corpul unei funcţii, caz în care este valabilă numai în ea sau în instrucţiunea compusă unde a fost declarată.

c) **variabile registru** – care sunt alocate în registrele procesorului. Ele pot fi numai variabile int, char şi pointer. Se recomandă declararea ca variabile registru, variabilele des utilizate în funcţia respectivă. Numărul variabilelor registru este limitat. Dacă s-au declarat mai multe, cele care nu pot fi alocate în registre vor fi alocate pe stivă ca variabile automatice.

Declararea variabilelor registru se face cu ajutorul cuvântului cheie **register**. Exemplu:
register int x;

Alocarea este valabilă numai în funcţia în care au fost declarate.

2.3. Exemplu de program modularizat

Următorul program calculează inversa şi determinantul unei matrice pătrate cu elemente numere reale. Programul a fost modularizat astfel:

- modulul „matrice” (fişierele *matrice.h, matrice.c*) – conţine funcţia de citire a dimensiunilor şi elementelor unei matrice, funcţia de afişare a unei matrice, funcţia de înmulţire a două matrice, funcţia de calcul a inversei unei matrice pătrate şi a determinatului atașat;
- programul principal (fișierul *main.c*) – conține funcția *main* în care sunt apelate funcții din cadrul modulului „matrice”.

Conținutul fișierului *matrice.h*:

```
#ifndef MATRICE_H_INCLUDED
   #define MATRICE_H_INCLUDED
   #define NMAX 10
   void citire_matrice(int *n,int *m,double a[NMAX][NMAX]);
   void afisare(int n,int m,double a[NMAX][NMAX],char ch);
   void produs(int n,int m,int p,double a[NMAX][NMAX],
                  double b[NMAX][NMAX],double c[NMAX][NMAX]);
   void invers(int n,double a[NMAX][NMAX],double eps,
                  double b[NMAX][NMAX],double *det_a,int *err);
#endif // MATRICE_H_INCLUDED
```

Conținutul fișierului *matrice.c*:

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

/* Afisarea si citirea unei matrice de n*m elemente de tip double */
void afisare(int n,int m,double a[NMAX][NMAX],char ch)
{
   int i,j;
   printf("\nMATRICEA  %c\n",ch);
   for(i=0; i<n; i++)
   {
      for(j=0; j<m; j++)
         printf("%8.2f ",a[i][j]);
      printf("\n");
   }
}

void citire_matrice(int *n,int *m,double a[NMAX][NMAX])
{
   int i,j;
   printf("\nIntroduceti nr.linii n=");
   scanf("%d",n);
   printf("\nIntroduceti nr.coloane m=");
   scanf("%d",m);
```

```
   printf("\nIntroduceti elementele matricei\n");
   for (i=0; i<*n; i++)
      for(j=0; j<*m; j++)
      {
         printf("a[%d,%d]=",i,j);
         scanf("%lf",&a[i][j]);
      }
   printf("\n");
}

/*Produsul matricelor a[n][m] si b[m][p] de tip double rezultand matricea c[n][p] */
void produs(int n,int m,int p,double a[NMAX][NMAX], double
b[NMAX][NMAX],double c[NMAX][NMAX])
{
   int i,j,k;
   double s;
   for(i=0; i<n; i++)
      for(j=0; j<p; j++)
      {
         s=0.0;
         for(k=0; k<m; k++)
            s=s+a[i][k]*b[k][j];
         c[i][j]=s;
      };
}

/*Calculul inversei unei matrice si a determinantului atasat */
void invers(int n,double a[NMAX][NMAX],double eps,
                  double b[NMAX][NMAX],double *det_a,int *err)
{
   int i,j,k,pozmax;
   double amax,aux;
   /* Initializarea matricei b cu matricea unitate */
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         if(i==j) b[i][j]=1.0;
         else b[i][j]=0.0;
   /* Initializarea determinantului */
   *det_a=1.0;
   /* Se face 0 sub diagonala principala si 1 pe ea */
   k=0; /*k=nr.liniei */
   *err=0;
   while((k<n) && (*err==0))
   {
      /*Calcul element pivot*/
      amax=fabs(a[k][k]);
      pozmax=k;

      for(i=k+1; i<n; i++)
         if (fabs(a[i][k]) >amax)
```

```
        {
            amax=fabs(a[i][k]);
            pozmax=i;
        };
    /*Interschimbarea liniei k cu pozmax in matr. a si b */
    if( k!=pozmax)
    {
        for(j=0; j<n; j++)
        {
            aux=a[k][j];
            a[k][j]=a[pozmax][j];
            a[pozmax][j]=aux;
            aux=b[k][j];
            b[k][j]=b[pozmax][j];
            b[pozmax][j]=aux;
        };
        *det_a=-*det_a;
    };

    if( fabs(a[k][k]) <eps) *err=1;
    else
    {
        *det_a =*det_a*a[k][k];
        aux=a[k][k];
        for(j=0; j<n; j++)
        {
            a[k][j]=a[k][j] / aux;
            b[k][j]=b[k][j] / aux;
        };
        for(i=0; i<n; i++)
            if(i!=k)
            {
                aux=a[i][k];
                for(j=0; j<n; j++)
                {
                    a[i][j]=a[i][j]-a[k][j]*aux;
                    b[i][j]=b[i][j]-b[k][j]*aux;
                }
            }
    }
    k++;
  }
}
```

Conținutul fișierului *main.c*:

```
/*Program de calcul a inversei unei matrice si a determinantului atasat */
#include <stdio.h>
#include "matrice.h"
```

```
int main()
{
   int i,j,n,m,err;
   double eps,det_a,a[NMAX][NMAX],a1[NMAX][NMAX];
   double b[NMAX][NMAX],c[NMAX][NMAX];
   citire_matrice(&n,&m,a);
   afisare(n,m,a,'A');
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         a1[i][j]=a[i][j];
   eps=1.0e-6;
   invers(n,a1,eps,b,&det_a,&err);
   if(err==1) printf("\nMATRICEA A ESTE SINGULARA");
   else
   {
      printf("\nMATRICEA INVERSA B=A^(-1)\n");
      afisare(n,n,b,'B');
      printf("\nDETERMINANTUL MATRICEI A ESTE %8.4f",det_a);
      produs(n,n,n,a,b,c);
      printf("\nVERIFICARE C=A*B REZULTA MATRICEA UNITATE!");
      afisare(n,n,c,'C');
   }
   return 0;
}
```

3. Mersul lucrării

3.1. Se vor compila, executa şi analiza programele modularizate date ca exemple în lucrare și în materialul de curs.

În continuare se va scrie câte un program modularizat pentru rezolvarea următoarelor probleme:

3.2. Să se scrie un program pentru calculul c.m.m.d.c. şi a c.m.m.m.c a două polinoame.

3.3. Să se implementeze noţiunea de mulţime şi operaţiile permise asupra sa.

3.4. Să se implementeze noţiunea de număr complex şi operaţiile permise asupra sa.

3.5. Să se implementeze noţiunea de fracţie şi operaţiile permise asupra sa.

3.6. Dându-se forma postfixată a unei expresii aritmetice care conţine numai numere întregi şi operatori +,-,*,/, să se scrie un program pentru evaluarea sa.