LIMBAJE DE PROGRAMARE

LABORATOR 11

I. ASPECTE TEORETICE

Structuri

1. Introducere

Pe parcursul alcatuirii de programe apare de multe ori necesitatea prelucrarii grupate a mai multor date. Datele se grupeaza pentru a forma multimi de elemente care sa poata fi prelucrate atat element cu element cat si global.
De regula, aceste grupe sunt multimi ordonate de date, adica datele unei astfel de grupe satisfac anumite relatii.

Cel mai simplu mod de grupare al datelor este tabloul. Tabloul este o multime ordonata de date de un acelasi tip, relatia de ordine intre elementele sale fiind definita cu ajutorul indicilor (care determina si dimensiunea tabloului).
Tipul comun elementelor tabloului este si tipul tabloului.

De multe ori este necesar a se grupa date, care nu sunt neaparat de acelasi tip, potrivit unei ierarhii. Datele grupate conform unei ierarhii se numesc structuri.

Un exemplu foarte simplu de structura este data calendaristica. Ea grupeaza trei date elementare: zi, luna si an. Componentele zi si an sunt date de tip intreg, iar componenta luna poate fi un sir de caractere.

2. Declaratia de structura

Formatul cel mai utilizat pentru a declara o structura este:

```
struct nume
        {
           lista_de_declaratii
        } nume1, nume2, ..., numen;
```

unde nume, nume1, ..., numen sunt niste denumiri pentru identificare care pot lipsi, dar nu toate deodata.

Obs.:
* Daca nume este absent, atunci cel putin nume1 trebuie sa fie prezent.
* Daca lista nume1, nume2, ..., numen este absenta, atunci trebuie ca nume sa fie prezent.

Daca nume este prezent, atunci el va defini un tip nou, introdus prin declaratia de structura respectiva.
Astfel nume1, nume2, ..., numen sunt structuri de tipul nume.

O structura de tip nume poate fi declarata si ulterior, utilizand formatul:

```
struct  nume  numele_noii_structuri;
```

Exemple:

a. Se va declara data_nasterii si data_angajarii ca structuri de tipul data_calendaristica (compusa din zi, luna, an):

```
        struct data_calendaristica
                {
                   int zi;
                   char luna[10];
                   int an;
                } data_nasterii, data_angajarii;
```

b. Se poate excepta introducerea data_calendaristica:

```
        struct
                {
                   int zi;
                   char luna[10];
                   int an;
                } data_nasterii, data_angajarii;
```

c. Putem defini tipul utilizator data_calendaristica si ulterior sa declaram data_nasterii si data_angajarii:

```
        struct data_calendaristica
                {
                   int zi;
                   char luna[10];
                   int an;
                } ;
        ...
        struct data_calendaristica  data_nasterii, data_angajarii;
```

Prima declaratie introduce tipul utilizator data_calendaristica, iar declaratia a doua defineste datele data_nasterii si data_angajarii ca fiind structuri de tipul data_calendaristica.

Extrapoland ideile de mai sus se poate usor defini o structura de date personale ale angajatilor unei institutii cuprinzad date ca: nume, prenume, adresa, locul nasterii, data nasterii, data angajarii, studii, sex, etc.

3. Accesul la componentele unei structuri

In general accesul la componentele unei structuri se realizeaza prin constructii de forma:

```
        nume.nume_data
```

sau:

```
        pointer -> nume_data
```

unde:

```
nume       - este numele structurii;
nume_data  - este numele componentei;
pointer    - este un pointer spre structura.
```

4. Declaratii de tip

In limbajul C se poate atribui un nume unui tip, indiferent daca el este un tip predefinit sau unul utilizator, utilizand o constructie de forma:

```
        typedef tip nume_tip;

unde:   tip      - este un tip predefinit sau un tip utilizator;
        nume_tip - numele care se atribuie tipului definit de tip;
```

Exemplu:
Prin declaratia

```
                typedef double REAL;
```

datele

```
                REAL x, y;
```

sunt de tip double.

II. DESFASURAREA LUCRARII

Nota: Primele doua puncte vor fi programe scrise in Turbo C si vor avea extensia .c, iar fisierele alcatuite de la punctul 3 pana la sfarsit, fiind in C++, vor avea extensia .cpp.

1. Scrieti un program care citeste numere complexe de la tastatura si afiseaza modulul lor.
Se va face o definire globala typedef pentru numere complexe (introduse ca o structura).
Pentru un numar complex

```
                        z = x + i*y
```

modulul este radacina patrata din x*x + y*y.
Functia de extragere a radacinii patrate este sqrt si este definita in fisierul header math.h.
Modulul numarului complex va fi calculat prin intermediul unei functii.
Programul va citi si va returna module de numere complexe pana cand se va incerca introducerea unei valori nenumerice de la tastatura, moment in care executia programului va lua sfarsit.

O posibila solutie:

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

typedef struct {
                 double x;
                 double y;
                } COMPLEX;
double dmodul(COMPLEX *z);


void main()  /*citeste numere complexe si afiseaza modulul lor */
{
 COMPLEX complex;
 printf("\nIntroduceti partea reala si partea imaginara ");
 printf("\n ale numarului complex z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
     {
       printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
```

```
       printf("modul=%g \n",dmodul(&complex));
       printf("\n\nIntroduceti partea reala si partea imaginara");
       printf("\n ale numarului complex z = a + ib :");
       printf("\n(Valori nenumerice incheie executia programului)\n");
     }
}

double dmodul(COMPLEX *z)
/*calculeaza si returneaza modulul numarului complex z*/
{
  return sqrt(z->x * z->x + z->y * z->y);
}
```

2. Completati programul de mai sus cu o functie care sa calculeze si argumentul numarului complex.

Daca

```
        z = x + i*y
```

atunci

```
        arg z
```

se calculeaza astfel:

a. Daca x = y = 0, atunci arg z = 0.
b. Daca y = 0 si x != 0
atunci: daca x > 0, arg z = 0;
        altfel arg z = pi = 3.1415926535.
c. Daca x = 0 si y != 0
atunci: daca y >0, arg z = pi/2;
        altfel arg z = 3*pi/2.
d. Daca x != 0 si y != 0 atunci
fie

```
        a = arctg(y/x)
```

Daca   x > 0 si y > 0, atunci arg z = a;
        x > 0 si y < 0, atunci arg z = 2*pi + a;
        x < 0, atunci arg z = pi + a.

Printr-un #define va fi introdusa initial si valoarea lui pi in program.
Functia arctg se gaseste sub numele atan in fisierul math.h .

3. Desfaceti programul de mai sus in trei fisiere cu extensia .cpp urmand principiile cunoscute si apoi lansati compilarea in fisierul ce contine functia main.

4. Scrieti intr-un fisier o functie

```
        void sum_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

care atribuie lui c suma numerelor complexe a si b.

5. Scrieti intr-un fisier o functie

```
        void scad_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

care atribuie lui c diferenta numerelor complexe a si b.

6. Scrieti intr-un fisier o functie

```
        void mul_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

care atribuie lui c produsul numerelor complexe a si b.

7. Scrieti intr-un fisier o functie

```
void div_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

care atribuie lui c rezultatul impartirii numerelor complexe a si b.
Ce masuri trebuie sa luati?

8. Scrieti un program care cere introducerea a doua numere complexe si afiseaza rezultatele adunarii, diferentei, inmultirii si impartirii lor.
Programul apeleaza fisierele scrise la punctele 4 - 7.