Laborator 1 - Structurarea datelor

Tema 1.1

Să se analizeze programul EX1.C din Anexa 1.

Consideraţii teoretice 1.1

Configurări mediu

În MinGW Developer Studio pentru a putea face o aplicație trebuie creat prima dată un proiect. Proiectele sunt colecții de fișiere sursă (*.cpp) care vor fi compilate separat iar din toate fișierele va fi realizat un singur fișier executabil. De aceea în fișierele sursă din cadrul unui proiect trebuie să existe doar o singură funcție main.

Pentru a realiza o aplicație consolă în MinGW în meniul *File->New* se alege tab-ul *Projects* de unde se alege *Win32 Console Application*. Fișierele C sau CPP realizate în cadrul proiectului se vor găsi în secțiunea *SourceFile*, iar fișierele antet (*.h) în secțiunea *HeaderFile.*

În cazul fişierelor antet (cele cu extensia .H), definite de utilizator, pentru a specifica compilatorului să caute aceste fișiere în directorul proiectului, includerea acestora se va face în forma următoare: `#include "nume_fisier.h".`

Dacă se folosește simbolurile "<" și ">" la includerea fișierelor acestea vor fi căutate în calea specificată de mediul de programare.

Pentru a configura MinGW să lucreze cu modul grafic trebuie făcute următoarele configurații în meniul *Project-> Settings -> Link*, apoi în câmpul *Extra linking options* trebuie introdus textul:

```
-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32
```

Modul grafic

Un program poate folosi monitorul în două moduri de lucru:

- *Modul de lucru text* – este modul în care ecranul este împărţit într-o matrice de "căsuţe" de dimensiune 80 x 25. În fiecare căsuţă se poate afişa un caracter.
- *Modul de lucru grafic* – este modul de lucru în care ecranul este împărţit într-o matrice de puncte (pixeli) de dimensiune 640 x 480. În acest mod de lucru pe ecran pot fi afişate atât caractere (de această dată caracterele pot fi scrise de dimensiuni diverse şi cu fonturi diferite), cât şi diferite obiecte grafice (puncte, linii, cercuri, etc.).

În ambele cazuri ecranul are coordonata 0,0 în colţul din stânga sus.

Câteva funcţii folosite pentru lucrul în modul grafic:

- `void initgraph(int *graphdriver, int *graphmode, char *pathtodriver)` – funcţia folosită pentru iniţializarea modului grafic prin încărcarea driver-ului grafic de pe disc. Parametrii funcţiei sunt: `graphdriver` – întreg care specifică driver-ul grafic care va fi folosit; `graphmode` – întreg care specifică modul graphic iniţial; `pathtodriver` – specifică calea în care funcţia va căuta mai întâi driver-ul grafic (fişier cu extensia .BGI). Dacă nu este găsit în directorul specificat, funcţia va căuta driver-ul în directorul curent. Dacă însă calea este nulă atunci fişierul de driver trebuie să se afle în directorul curent.

- `int graphresult(void)` – funcţia returnează o valoare care reprezintă codul de eroare al ultimei operaţii grafice (0 – operaţia s-a efectuat cu succes, altă valoare – operaţia nu a avut succes).
- `void closegraph(void)` – funcţia care închide modul grafic şi revine în modul text.
- `void setcolor(int culoare)` – funcţie care setează culoarea cu care se va desena în continuare, parametrul `culoare` reprezintă culoarea.
- `void circle(int x,int y,int r)` – funcţie care desenează un cerc cu coordonatele centrului `x`, `y` şi de rază `r`.

Variabile

Variabilele se declară, de obicei, în trei locuri: în interiorul funcţiilor (variabile locale), în cadrul definiţiei parametrilor funcţiei (parametri formali) şi în afara oricărei funcţii (variabile globale). Variabilele locale sunt declarate în interiorul unei funcţii, ele fiind accesibile doar instrucţiunilor care se află în interiorul blocului în care sunt declarate variabilele. Parametri formali ai funcţiei se comportă ca orice altă variabilă locală din acea funcţie. Variabilele globale sunt cunoscute în întreg programul şi pot fi utilizate de către orice zonă a codului.

Funcţii

Forma generală a definirii unei funcţii:
```
specificator_de_tip nume_functie(lista_de_parametrii)
 { corpul_functiei }
```

`specificatorul_de_tip` specifică tipul de date pe care îl returnează funcţia. Dacă nu este specificat nici un tip, compilatorul presupune că funcţia returnează un rezultat de tip `int`. `lista_de_parametrii` este o listă separată prin virgule de tipul şi numele parametrilor formali. Parametrii formali primesc valorile argumentelor atunci când este apelată funcţia, compilatorul verificând în momentul apelului corespondenţa între parametrii actuali şi cei formali.

În unele cazuri este preferat modelul care implică folosirea prototipului funcţiei (declararea ei), definirea funcţiei urmând a fi făcută ulterior. Forma generală a prototipului unei funcţii:
```
specificator_de_tip nume_functie(lista_de_parametrii);
```

În general funcţiile pot primi argumente în două feluri: apel prin valoare şi apel prin pointer.

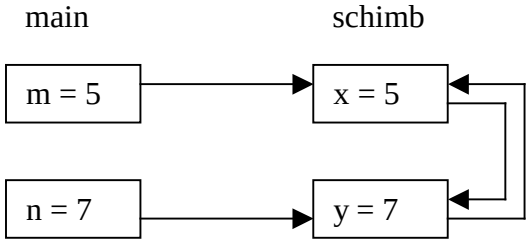
Să luăm următorul exemplu: dorim să facem o funcţie care să interschimbe două valori. O primă variantă (scrisă rapid) ar arăta în felul următor:

```
#include <iostream.h>
void schimb(int x, int y)
{
 int r;

   r = x;
   x = y;
   y = r;
}

void main()
{
 int m=5,n=7;

   cout<<m<<"  "<<n<<"  ";
   schimb(m,n);
```

```
   cout<<m<<"  "<<n;
}
```

Vom avea însă următoarea surpriză programul va afişa: 5 7 5 7. Ce s-a întâmplat de nu s-au schimbat totuşi valorile între ele?

În funcţia `main` am definit cele două variabile `m, n` pentru care se alocă memorie. În momentul în care apelăm funcţia `schimb(m,n)`, această funcţie foloseşte variabilele locale `x, y, r` pentru care se alocă memorie (pe stivă) atâta timp cât va rula această funcţie. Variabila `x` va primi valoarea lui `m`, iar variabila `y` va primi valoarea lui `n`. Apoi se face schimbarea de valori între variabilele locale `x, y`, fără ca valorile din zona de memorie alocată pentru variabilele `m, n` să fie afectate. La revenirea din funcţia `schimb` cele două variabile `m, n` vor avea tot aceleaşi valori ca mai înainte.



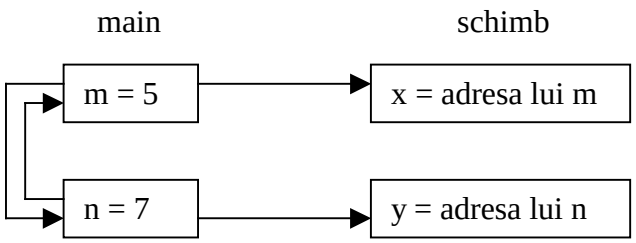
Schimbarea valorilor se face pentru variabilele locale x, y; variabilele m, n din main rămân neschimbate

Acesta a fost un exemplu de *apel prin valoare* (în urma căruia nu am reuşit să schimbăm valorile variabilelor folosite în apelarea funcţiei).

Pentru ca totuşi să reuşim ce ne-am propus vom folosi *apel prin pointer*. Exemplul nostru va arăta astfel:

```
#include <iostream.h>
void schimb(int *x, int *y)
{
 int r;

    r = *x;
   *x = *y;
   *y =  r;
}

void main()
{
 int m=5,n=7;

   cout<<m<<"  "<<n<<"  ";
   schimb(&m,&n);
   cout<<m<<"  "<<n;
}
```

Scris în acest fel programul va face ceea ce ne dorim. De fapt ce s-a modificat? Funcţia `schimb` are acum ca şi parametrii doi pointeri în care se vor memora adresele variabilelor `m, n`. De această dată în funcţia `schimb` se vor schimba valorile de la adresele de memorie date de `x, y`, respectiv se vor schimba valorile din zona de memorie unde sunt memorate variabilele `m, n`.



Schimbarea valorilor se face pentru locaţiile de memorie a căror adrese sunt în x şi în y (respectiv locaţiile care conţin varibilele m şi n)

Lucrul cu tastatura

Pentru a afla codul generat la apăsarea unei taste se foloseşte funcţia `getch()`. Aceasta aşteaptă apăsarea unei taste şi întoarce:

- pentru tastele normale codul ASCII al tastei apăsate.
- pentru tastele speciale valoarea 0 urmată, la o nouă apelare a funcţiei `getch()`, de un cod al tastei apăsate. Taste speciale sunt de exemplu: săgeţile, F1 – F12, tastele INS, DEL, PAGE UP, PAGE DOWN, HOME şi END.

Indicaţii 1.1

- ⇨ Atenţie, nu uitaţi de fişierul EX1.H
- ⇨ Atenţie la codul asociat TAB
- ⇨ Atenţie la restricţiile compilatorului de C.

Tema 1.2

Să se modifice programul aşa încât să gestioneze 4 (patru) cercuri.

Indicaţii 1.2

- ⇨ Se vor declara variabilele x4, y4, r4, c4 şi se va modifica corespunzător programul.
- ⇨ Atenţie la codul asociat TAB.

Tema 1.3

Să se modifice programul aşa încât să gestioneze 10 (zece) cercuri.

Consideraţii teoretice 1.3

Vectori

Un vector (o matrice cu o singură dimensiune) este o colecţie de variabile de acelaşi tip, apelate cu acelaşi nume. Accesul la un anumit element al vectorului se face cu ajutorul unui indice. Un vector ocupă locaţii de memorie contigue (consecutive). Forma generală pentru declararea unui vector este:

```
tip nume_variab[marime];
```

Vectorii trebuiesc declaraţi explicit astfel încât compilatorul să aloce spaţiu în memorie pentru ei. `tip` declară tipul de bază al vectorului, care este tipul fiecărui element al său. `marime` indică numărul de elemente al vectorului. Apelarea unui element al vectorului se va face astfel:

```
nume_varib[indice]
```

unde `indice` va putea lua valori de la 0 la `marime-1`.

Culori

Culorile pot fi privite ca numere întregi în plaja 0-15. În tabelul următor sunt prezentate corespondenţele în numere pentru cele 16 culori definite simbolic în `graphics.h`.

```
 0 BLACK            1 BLUE             2 GREEN            3 CYAN
 4 RED              5 MAGENTA          6 BROWN            7 LIGHTGRAY
 8 DARKGRAY         9 LIGHTBLUE       10 LIGHTGREEN      11 LIGHTCYAN
12 LIGHTRED        13 LIGHTMAGENTA    14 YELLOW          15 WHITE
```

Indicaţii 1.3

- Se vor folosi patru vectori, câte unul pentru fiecare variabilă `x`, `y`, `r` respectiv `c`.
- La iniţializare se recomandă folosirea unei bucle for pentru a iniţializa fiecare cerc, iar valorile atribuite să fie dependente de contorul buclei (de exemplu `x[k]=200+10*k`)
- Datorită ordinii realizate în date se poate renunţa la `switch(CercCurent)` şi înlocui cu o singură linie cu apel de `Muta` şi indexare (după variabila `CercCurent`) în tablourile corespunzătoare
- Nu se va modifica funcţia `Muta`

Tema 1.4

Să se modifice programul astfel încât să folosească o matrice care să memoreze datele cercurilor.

Consideraţii teoretice 1.4

Matrici

Compilatorul de C admite declararea de matrici multidimensionale. Cea mai simplă formă de matrice multidimensională este cea bidimensională, care poate fi privită ca un vector de vectori. Definirea unei matrici se va face în felul următor:

```
tip nume_matrice[marime1][marime2];
```

`tip` reprezintă tipul de date pentru elementele matricii. Matricele bidimensionale sunt stocate în forma rând-coloană, unde primul indice (`marime1`) indică rândul iar al doilea (`marime2`) indică coloana. Accesul la un element al matricii se face sub forma:

```
nume_matrice[indice1][indice2];
```

unde `indice1` va lua valori între `0` şi `marime1`, iar `indice2` va lua valori între `0` şi `marime2`. În ceea ce priveşte stocarea efectivă în memorie a matricii, `indice2` se modifică mai repede decât `indice1` atunci când sunt parcurse succesiv elementele matricii.

Indicaţii 1.4

- Se va folosi o singură matrice, cu 4 coloane, corespunzătoare variabilelor `x`, `y`, `r` şi `c`.
- Nu se va modifica funcţia `Muta`

Tema 1.5

Să se definească o structură CERC, aferentă unui cerc, şi să se modifice corespunzător programul.

Consideraţii teoretice 1.5

Structuri

O structură este un grup de variabile unite sub acelaşi nume, ce pun la dispoziţie un mod convenabil de păstrare a informaţiilor legate între ele. O declarare de structură formează un şablon care poate fi folosit pentru a crea variabile de acel tip. Membrii (elementele sau câmpurile) unei structuri sunt variabilele care fac parte din structură. Forma generală a definirii unei structuri este:

```
struct nume_structura
{
 tip1 nume_membru1;
 tip2 nume_membru2;
 ...
} variabile_structura;
```

unde `nume_structura` este numele structurii şi `variabile_structura` este lista de variabile de tipul structurii. Cele două pot fi omise, dar nu ambele simultan. Atenţie declaraţia se termina cu punct şi virgulă, deoarece este o instrucţiune. Având definită o structură, definirea unei variabile de tipul structurii se va face astfel:

```
struct nume_structura nume_variabila;
```

Forma generală de acces la un membru al structurii este:

```
nume_variabila.nume_membru
```

În cazul unei variabile pointer la o structură accesul la un membru al structurii se va face astfel:

```
nume_pointer->nume_membru
```

Limbajul C permite definirea explicită a noi nume de tipuri de date prin utilizarea cuvântului cheie `typedef`. Forma generală a instrucţiunii `typedef` este:

```
typedef tip_vechi tip_nou;
```

Dacă `tip_vechi` este o structură, noul tip de date definit se poate folosi fără a fi nevoie să se specifice de fiecare dată cuvântul `struct` la definirea variabilelor.

Indicaţii 1.5

- ⇨ Se va defini un vector de elemente de tip `CERC`
- ⇨ Se va modifica funcţia `Muta` astfel încât să primească doar un parametru referitor la `CERC` şi deplasările (în total 3 parametrii).
- ⇨ Atenţie la declararea (prototipul), definirea şi apelul corespunzător al funcţiei `Muta`.

Tema 1.6

Să se renunţe la variabilele `dx` şi `dy` din `main` fără a se modifica însă funcţia `Muta`.

Indicaţii 1.6

- ⇨ Se vor lua în considerare apeluri ale funcţiei `Muta` ori de câte ori este cazul.

Tema 1.7

Să se introducă doua funcţii `Sterge` şi `Afiseaza` care să realizeze ştergerea şi respectiv afişarea unui cerc primind ca şi parametru cercul. Acestea vor fi singurele funcţii care mai conţin funcţiile `setcolor` şi `circle`.

Tema 1.8

Să se adauge o facilitate suplimentară de modificat raza cercului curent: la apăsarea ‚1' raza creşte cu 10 iar la apăsarea ‚2' raza scade cu 10.

Consideraţii teoretice 1.8

Tastele ‚1' şi ‚2' sunt taste normale:

Indicaţii 1.8

- Implementarea va fi asemănătoare funcţiei `Muta`, iar numele funcţiei va fi `Creste`.

Anexa 1

ex1.c

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

#include "ex1.h" //includerea bibliotecii proprii ex1.h care trebuie
                 //sa fie in directorul curent.

// prototipuri de functii

void OurInitGraph(void);
void Muta(int *x,int *y,int r,int c,int dx,int dy);

// variabile Globale, pentru un cerc vom avea urmatoarele date:
//  - x, y coordonatele centrului cercului
//  - r raza cercului
//  - c culoarea cercului

// se vor defini 3 cercuri

int x1,y1,r1,c1, x2,y2,r2,c2, x3,y3,r3,c3; //variabilele pentru cele 3 cercuri

void main()
{
// variabile locale

int CercCurent=0;
// este variabila care ne pastreaza cercul curent ea va putea lua 3 valori:
// 0 - pentru cercul x1, y1, r1, c1
// 1 - pentru cercul x2, y2, r2, c2
// 2 - pentru cercul x3, y3, r3, c3
int gata=0;
int dx,dy;

// cod

   OurInitGraph(); //se face trecerea in modul grafic

//urmeaza introducerea valorilor initiale pentru cercuri
//(initializarea cercurilor)

   x1=100;y1=200;r1= 25;c1=YELLOW; //initializarea variabilelor pt. cerc 1
   setcolor(c1);                   //se seteaza culoarea c1
   circle(x1,y1,r1);               //se deseneaza cercul cu centrul x1, y1
                                   //    si de raza r1

   x2=300;y2=200;r2= 50;c2=RED;    //initializarea variabilelor pt. cerc 2
   setcolor(c2);                   //se seteaza culoarea c2
   circle(x2,y2,r2);               //se deseneaza cercul cu centrul x2, y2
                                   //    si de raza r2

   x3=500;y3=200;r3=100;c3=BLUE;   //initializarea variabilelor pt. cerc 3
   setcolor(c3);                   //se seteaza culoarea c3
   circle(x3,y3,r3);               //se deseneaza cercul cu centrul x3, y3
                                   //    si de raza r3
```

```
// se intra intr-o bucla in care se ramane atata timp cat gata=0
   while(!gata)
     switch(getch())      //se asteapta apasarea unei taste
      {
       case ESC:          //daca s-a apasat ESC atunci se iasa din bucla
         gata=1;          //prin valoarea 1 pe care o primeste gata
         break;
       case TAB:          //daca s-a apasat TAB se trece la urmatorul cerc
         CercCurent++;    //trecem la cercul urmator
         CercCurent%=3;   //dupa cercul 2 urmeaza cercul 0
         break;
       case 0:            //la apasarea unei sageti se genereaza intai 0.
         switch(getch())  //apoi codul specific
          {
           case LEFT:  dx=-10;dy=  0;break;
           case RIGHT: dx= 10;dy=  0;break;
           case UP:    dx=  0;dy=-10;break;
           case DOWN:  dx=  0;dy= 10;break;
           default:    dx=  0;dy=  0;break;
          }
//in functie de sageata apasata se dau valori deplasarilor:
//  - dx pe orizontala (valoarea negativa insemnand deplasare la stanga)
//  - dy pe verticala  (valoarea negativa insemnand deplasare in sus)
//pasul de deplasare pe orizontala si pe verticala este de 10 pixeli

//dupa stabilirea valorilor pentru deplasari se muta cercul curent
         switch(CercCurent)
          {
           case 0:                       //CercCurent=0
             Muta(&x1,&y1,r1,c1,dx,dy); //mutarea cercului 1
             break;
           case 1:                       //CercCurent=1
             Muta(&x2,&y2,r2,c2,dx,dy); //mutarea cercului 2
             break;
           case 2:                      //CercCurent=2
             Muta(&x3,&y3,r3,c3,dx,dy); //mutarea cercului 3
             break;
          }
         break;
       }
   closegraph();        //inchiderea modului grafic
}

void Muta(int *x,int *y,int r,int color,int dx,int dy)
//****************************************************
//functie care face mutarea unui cerc cu deplasamentele
//dx (pe orizontala) si dy (pe verticala)
{
  setcolor(BLACK);  //sterge cercul de pe pozitia veche, prin
  circle(*x,*y,r);  //scrierea unui cerc de culoarea fondului (negru) peste el

  *x += dx;         //sunt modificate coordonatele centrului cercului
  *y += dy;

  setcolor(color);
  circle(*x,*y,r);  //desenare in noua pozitie a cercului
}

void OurInitGraph()
//*********************
// functie care face trecerea din modul text in modul grafic
{
```

```
int gdriver = DETECT, gmode, errorcode;  //declarare de variabile locale

  initgraph(&gdriver,&gmode,"");   //initializeaza modul grafic
  errorcode = graphresult();       //citeste rezultatul initializarii
   if (errorcode != grOk)          //se testeaza daca a aparut o eroare la
                                   //initializare, in caz de eroare se afiseaza
    {                              //eroarea si apoi se inchide programul
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
    }
}
```

ex1.h

```
//***************************************************************************

// in acest fisier sunt definite ca si constante codurile ASCII pentru
// tastele folosite in program

#define ESC   27
#define TAB    9
#define LEFT  75
#define RIGHT 77
#define UP    72
#define DOWN  80
```