Laborator III

ELEMENTE SPECIFICE C++

1. Preprocesare

1. Un program *C* sau *C++* poate fi prelucrat înainte de a fi supus compilării. O astfel de prelucrare se numeşte *preprocesare*. Ea este realizată automat înaintea compilării şi constă în principiu din substituţii. Preprocesarea asigură:

- Includeri de fişiere cu texte sursă. De exemplu:

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

sau

```
#include "fisier1.cpp"
```

- Definiţii şi apeluri de macrouri. De exemplu:

```
#define PI 3.14159
.....
#undef PI
```

sau

```
#define A 123
#define B A+57
```

sau

```
#define Modul(a) (a)<0 ? -(a) : (a)
```

- Compilare condiţionată, utilizând:

```
#if, #else, #elif, #endif
```

sau

```
#ifdef, #ifndef
```

Utilizarea preprocesării este ilustrată în exemplele următoare.

2. Creaţi un proiect *HelloC*, care să conţină următoarele surse: *Hello.cpp, PrintM.cpp* şi *Question_C.cpp*. Faceţi modificările necesare în fişierul *Hello.hpp* astfel încât liniile *#define IO_CPP* şi *#include <iostream>* să fie comentate, iar linia *#define IO_C* să nu fie comentată. Se reaminteşte faptul că fişierul header *Hello.hpp* nu trebuie inclus explicit în proiect, fiind inclus prin comanda *#include "hello.hpp"*. Rulaţi progarmul pas cu pas şi observaţi comportarea acestuia. La rularea pas cu pas, intraţi în funcţiile *PrintMessage* şi *Question*. Urmăriţi valorile variabilelor locale. Remarcaţi de asemenea modul în care se poate marca un comentariu în *C* şi în *C++*.

3. Creaţi un proiect *HelloCpp*, care să conţină următoarele surse: *Hello.cpp, PrintM.cpp* şi *Question_Cpp.cpp*. Faceţi modificările necesare în fişierul *Hello.hpp* astfel încât liniile *#define IO_CPP* şi *#include <iostream>* să nu fie comentate, iar linia *#define IO_C* să fie comentată. Rulaţi programul pas cu pas la fel ca la punctul anterior. Remarcaţi diferenţele de limbaj între *C* şi *C++*.

Precizaţi care secţiune din funcţia *Question* (definită în *Question_Cpp.cpp*) este compilată şi de ce. Argumentaţi comentarea şi decomentarea liniilor din *Hello.hpp*.

```
/*********************************************
Hello.hpp
Declaratii si directive #include ...
*********************************************/
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <iostream>


#ifndef _HELLO_INCLUDED_
#define _HELLO_INCLUDED_

//#define IO_C
#define IO_CPP


//Prototipuri de functii ...

//Parametri: pointer la caracter.
//Tipareste mesajul parametru.
void PrintMessage(char *);

//Parametri: pointer la caracter.
//Formuleaza intrebarea furnizata ca parametru si
//returneaza o valoare diferita de zero
//in caz ca raspunsul este 'da' si zero in caz ca
//raspunsul este nu.
int Question(char *);

#endif
```

```
/*********************************************
PrintM.cpp
Definitia functiei PrintMessage ...
*********************************************/
#include "hello.hpp"
using namespace std;
//Variabile locale fisierului sursa.
char Mesaj1[] = "Mesajul este: %s\n";
char Mesaj2[] = "Mesajul este: ";

void PrintMessage(char *Mesaj)
{
#ifdef IO_C
    printf("Mesajul este: %s\n", Mesaj);
    printf(Mesaj1, Mesaj);
#endif
#ifdef IO_CPP
    cout << Mesaj2 << Mesaj << endl;
#endif
}
```

```
#ifdef QUESTION2
int Question(char *Mesaj)
{
    int DaNu;
    cout << Mesaj << " (y/n)";
    while(1)
    {
        DaNu = toupper(_getch());
        if(DaNu == 'Y')
        {
            cout << endl;
            return 1;
        }
        if(DaNu == 'N')
        {
            cout << endl;
            return 0;
        }
    }
}
#endif
```

```
/*********************************************
Question_Cpp.cpp
Definitia functiei Question ...
*********************************************/
#include "hello.hpp"
using namespace std;
//Care din cele doua variante ale functiei
//Question va fi compilata ...
//#define QUESTION1
#define QUESTION2

//Parametri: pointer la caracter.
//Formuleaza intrebarea furnizata ca parametru si
//returneaza o valoare diferita de zero
//in caz ca raspunsul este 'da' si zero in caz ca
//raspunsul este nu.
#ifdef QUESTION1
int Question(char *Mesaj)
{
    int DaNu; //In C++ int este diferit de char !
    //Functia toupper functioneaza cu int.
    cout << Mesaj << " (y/n)";
    do
    {
        //DaNu = toupper(_getch());
        DaNu = toupper(fgetc(stdin));
        //DaNu = toupper(fgetc(stdin));
        /*
        //Are acelasi efect ...
        DaNu = _getch();
        DaNu = toupper();
        */
    }
    while(DaNu != 'Y' && DaNu != 'N');
    cout << endl;
    if(DaNu == 'Y') //DaNu are valoarea Y sau N !
        return 1;
    else
        return 0;
}
#endif
```

```
/*********************************************
Question_C.cpp
Definitia functiei Question ...
*********************************************/
#include "hello.hpp"
using namespace std;
//Parametri: pointer la caracter.
//Formuleaza intrebarea furnizata ca parametru
//si returneaza o valoare diferita de zero
//in caz ca raspunsul este 'da' si zero in caz ca
//raspunsul este nu.
int Question(char *Mesaj)
{
    int DaNu;
    //In C++ int este diferit de char !!!
    //Functia toupper functioneaza cu int.
    printf(Mesaj);
    puts(" (y/n)");
    do
    {
        DaNu = toupper(_getch());
        //DaNu = toupper(fgetc(stdin));
        /*
        //Are acelasi efect ...
        DaNu = _getch();
        DaNu = toupper();
        */
    }
    while(DaNu != 'Y' && DaNu != 'N');
    if(DaNu == 'Y') //DaNu are valoarea Y sau N
!!!
        return 1;
    else
        return 0;
}
```

```
/*********************************************
Hello.cpp
Definitia functiei main ...
*********************************************/
#include "hello.hpp"

//Linie de comentariu C++ ...
/*Comentariu C ...*/
using namespace std;
void main(void)
{
    char Mesaj1[] = "\nHello C++ world !\n";
    //Sir de caractere initializat cu constanta
    //sir de caractere ...
    char Mesaj2[60]; //Sir de 60 de caractere:
    //Mesaj2[0] este primul caracter ...
    char *Mesaj3 = "\nAti introdus mesajul: ";
    //Mesaj3 este pointer la sir constant de
    //caractere ...
    char *Mesaj4 = "\nAti introdus mesajul: %s\n";
    //Mesaj4 este pointer la sir constant de
    //caractere ...
    do
    {
        PrintMessage(Mesaj1);
if(Question("\nVreti sa introduceti un mesaj?"))
        {
#ifdef IO_CPP
            //C++
cout << endl << "Introduceti mesajul pentru C++ !";
  cout << endl;
  cin >> Mesaj2;
            cout << Mesaj3 << Mesaj2 << endl;
#endif
#ifdef IO_C
            //C
       puts("\nIntroduceti mesajul pentru C !");
            scanf("%s", Mesaj2);
            printf(Mesaj4, Mesaj2);
            //Apel de functie ...
#endif
            PrintMessage(Mesaj2);
        }
    }
    while(Question("\nRepetam ?"));
}
```

2. Supraîncărcarea funcţiilor

Supraîncărcarea funcţiilor presupune existenţa mai multor funcţii diferite, dar cu acelaşi nume, în acelaşi proiect. Restricţia care se impune la supraîncărcarea funcţiilor şi care este şi criteriul după care compilatorul deosebeşte între ele funcţiile cu acelaşi nume, este ca tipul şi/sau numărul de parametri al funcţiilor supraîncărcate să difere. Atenţie: tipul returnat de funcţii nu asigură de regulă suficiente informaţii pentru ca un compilator să poată decide ce funcţie să folosească la apelare. Aşadar, funcţiile supraîncărcate pot să difere şi prin tipul returnat, dar trebuie neapărat să difere prin numărul sau tipul parametrilor.

Cele două exemple prezentate în continuare ilustrează supraîncărcarea funcţiilor. Compilaţi şi rulaţi pas cu pas fiecare din cele două exemple şi urmăriţi cu atenţie diferenţele între funcţiile cu acelaşi nume şi încercaţi să înţelegeţi de ce se apelează o funcţie sau alta. Decomentaţi apelurile de funcţii comentate şi remarcaţi mesajele afişate la încercarea compilării.

Scrieţi şi testaţi un program care să conţină trei funcţii cu acelaşi nume (*Divide*) care să realizeze câtul a două variabile de tip *int*, *double*, respectiv *float*.

```
/*****************************
Supraincarcarea functiilor
addsir.cpp
*****************************/

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

using namespace std;
void adunsir(char *s1, char *s2);
void adunsir(char *s1, int i);

int main()
{
    char sir [80];

    strcpy(sir, "Functii ");
    adunsir(sir, "supraincarcate ");
    cout << sir << "\n";

    adunsir(sir, 100);
    cout << sir << endl;

    _getch();
    return 0;
}

//Functia concateneaza doua siruri
void adunsir(char *s1, char *s2)
{
    strcat(s1, s2);
}

//Functia concateneaza un sir cu un intreg "sir"
void adunsir (char *s1, int i)
{
    char temp[80];

    sprintf(temp, "%d", i);
    strcat(s1,temp);
}
```

```
/*****************************
Supraincarcarea functiilor
medie.cpp
*****************************/

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

using namespace std;
```

```
float Media(int a, int b)
//Media aritmetica a 2 intregi
{
    cout << "media aritmetica a 2 intregi" << endl;
    float Med = float(a+b)/2;
    return Med;
}

float Media(int a, float b)
//Modulul mediei aritmetice ponderate a unui
intreg //cu un real
{
    cout << "modulul mediei aritmetice ponderate";
    cout << " a unui intreg cu un real" << endl;
    double Med = (0.25*a+1.25*b)/1.5;
    return Med;
}

float Media(int a, int b, int c)
//Media geometrica a 3 intregi
{
    cout << "media geometrica a 3 intregi" << endl;
    float exp=1./3;
    float Med=pow(a*b*c,exp);
    return Med;
}

void Media(int a, int b, float c)
//Media armonica a 2 intregi si un real fara
//returnare de valoare
{
    cout << "media armonica a 2 intregi si un ";
    cout << "real fara returnare de valoare";
    cout << endl;
    float Med;

    if (a!=0 && b!=0 && c!=0)
        {
            Med=3/(1/a+1/b+1/c);
            cout << "Media armonica testata Ma=";
            cout << Med << endl;
        }
    else
        {
            cout << "Media armonica nu se poate";
            cout << " calcula!"<<endl;
        }
}
```

```
int main()
{
    double x,z;
    float y;

    cout << endl << "Se apleaza ";
    cout << "Media(7,8)=" << Media(7,8) << endl;

    //cout << endl << "Se apleaza ";
    //cout << "Media(7,8.5)=" << Media(7,8.5);
    //cout << endl;

    cout << endl << "Se apleaza ";
    y=8.5;
    cout << "Media(7,y=8.5)=" << Media(7,y);
    cout << endl;

    cout << endl << "Se apleaza ";
    x = Media(6,8);
    cout << "Media(6,8)=x=" << x << endl;

    cout << endl << "Se apleaza ";
    cout << "Media(5,8,9)=" << Media(5,8,9) <<endl;

    // z este de tip double. Ce se intampla?
    cout << endl << "Se apleaza ";
    z = Media(5,8,9);
    cout << "Media(5,8,9)=z=" << z << endl;

    //cout << endl << "Se apleaza ";
    //float t = Media(5,8,9.5);
    //cout << "Media(5,8,9.5)=t=" << t << endl;

    y=1.7;

    //cout << endl << "Se apleaza ";
    //cout << "Media(3,2,1.7)=" << Media(3,2,1.7);
    //cout <<endl;

    cout << endl << "Se apleaza ";
    Media(3,2,y);

    //cout << endl << "Se apleaza ";
    //Media(3,2,1.7);

    //Atentie la functiile apelate mai jos:

    cout << endl << "Se apleaza ";
    Media(1,5,0);

    cout << endl << "Se apleaza ";
    Media(1,y,0);

    cout << endl << "Se apleaza ";
    Media(1,0,y);

    _getch();
    return 0;
}
```

3. Argumente implicite

Argumentele implicite sunt o altă caracteristică a limbajului *C++*. Dacă supraîncărcarea funcţiilor permite programatorului să utilizeze aceleaşi nume pentru mai multe funcţii, argumentele implicite permit unei funcţii să fie utilizată ca şi cum ar fi mai multe funcţii.

La declararea funcţiei se pot atribui valori pentru o parte sau toţi parametrii funcţiei. Respectivii parametri se vor numi în continuare *impliciţi* şi pot lipsi la apelul funcţiei, valorile lor fiind date de cele implicite. Şi în acest caz însă există o restricţie: parametrii impliciţi sunt întotdeauna plasaţi la sfârşitul listei de parametri formali ai funcţiei. De asemenea, nu sunt admise situaţii de genul: ultimul parametru implicit să fie prezent în apelul funcţiei, iar penultimul să lipsească.

Urmăriţi exemplul următor pas cu pas şi analizaţi fiecare apel de funcţie.

Gândiţi şi scrieţi un program care să conţină o funcţie cu *parametri impliciţi*, care să realizeze în mod implicit ridicarea la pătrat a unui întreg sau ridicarea la oricare altă putere (număr întreg), dacă se specifică valoarea acesteia.

```
/*****************************
Parametri Impliciti
salut.cpp
*****************************/
```

```
#include <iostream>
#include <stdio.h>
#include <conio.h>
using namespace std;

//Atentie la spatiul dintre * si = :
void Afis(char *Sir0, char *, char * ="studentule", char * ="!");
//Ultimii 2 parametri sunt impliciti

//Este gresita varianta de mai jos?
//void Afis(char *Sir0, char *Sir1, char *Sir2="studentule", char *Sir3="!");

//Ce greseala sesizati in varianta urmatoare?
//void Afis(char *Sir0, char *Sir1, char *Sir2="studentule", char *Sir3);

void main()
{
    char Nume[20];

    cout << "Introduceti un nume: ";
    cin >> Nume;
    cout << endl;

    Afis("0-> ","Salut ");
    Afis("1-> ","Salut ",Nume);
    Afis("2-> ","Spor la lucru, ");
    Afis("3-> ","Iti place C++, ",Nume,"?");

    //Apeluri incorecte:
    //Afis("4-> ",);
    //Afis("5-> ","Iti place C++, ", ,"?");

    _getch();
}

void Afis(char *Sir0, char *Sir1, char *Sir2, char *Sir3)
{
    cout << Sir0 << Sir1 << Sir2 << Sir3 <<endl;
}
```

4. Referinţe

Spre deosebire de *C*, *C++* oferă posibilitatea utilizării *apelului prin referinţă*. O referinţă este un alias sau un alt nume pentru un obiect existent. De exemplu, fie variabila:

```
int n = 10;
```

Putem declara o referinţă a acesteia:

```
int& r = n;
```

Acum *r* este un *alias* pentru *n*; amândouă identifică acelaşi obiect şi sunt interschimbabile. Prin urmare:

```
r = -10;
```

are ca efect schimbarea valorii atât pentru *r* cât şi pentru *n* în *-10*.

Cu alte cuvinte, valoarea este una singură, aflată într-o anumită locaţie de memorie, însă adresa locaţiei de memorie poate fi memorată sub mai multe nume.

Programatorii în *C* sunt obişnuiţi cu mecanismul *apelului prin valoare*, ceea ce înseamnă că pentru a modifica valorile parametrilor săi, funcţia trebuie să aibă de fapt drept parametri poineri. Este cazul funcţiei *Interschimb()*, care realizează interschimbarea a două variabile şi este scrisă mai jos în *C*:

```
void Interschimb(int* a, int* b)
      {
            int tmp;
            tmp = *a;
            *a = *b;
            *b = tmp;
```

```
}
```

Funcţia se apelează astfel:

```
int x = 1;
int y = 2;
Interschimb(&x,&y);
```

În cazul neatenţiei manevrării pointerilor şi referinţelor, rezultatele pot fi dezastroase, erorile rezultate putând avea efecte necontrolabile.

Aceeaşi funcţie de interschimbare a două variabile poate fi scrisă în *C++*, utilizând apelul prin referinţă:

```
void Interschimb(int &a, int &b)
	{
		int tmp;
		tmp = a;
		a = b;
		b = tmp;
	}
```

Apelul funcţiei se face astfel:

```
int x = 1;
int y = 2;
Interschimb(x,y);
```

O altă utilzare, care se poate dovedi foarte eficientă în anumite cazuri, a referinţelor în *C++* este aceea din exemplul următor:

```
int& FindbyIndex(int* Vector, int Index)
	{
		return Vector[index];
	}
```

Astfel este posibilă o apelare de genul următor:

```
FindbyIndex(A,3) = 25;
```

are ca rezultat atribuirea valorii *25* elementului cu indexul *3* din vectorul *A*.

Scrieţi un program *C++* care să realizeze ordonarea crescătoare a elementelor unui vector de întregi, folosind o funcţie cu *apel prin referinţă*.

5. Structuri de date

După cum se va vedea, structurile de date reprezintă trecerea către clasele din C++. Tipul de date enumerat este utilizat adesea când se lucrează cu înşiruiri de date simple ce nu necesită funcţii complexe pentru manevrarea acestora, care să justifice implementarea unor clase cu funcţii membru.

La tipul enumerat valorile posibile sunt înşiruite explicit prin nume şi nu prin tip de date. În acest sens, exemplul de mai jos este concludent.

```
#include <iostream>
#include <conio.h>
using namespace std;

enum ZilePeSapt {Luni, Marti, Miercuri, Joi, Vineri, Sambata, Duminica};
enum LuniPeAn {Ian, Feb, Mar, Apr, Mai, Iun, Iul, Aug, Sep, Oct, Nov, Dec};

struct Data
	{
		ZilePeSapt NumeZi;
		int Zi;
		LuniPeAn Luna;
		int An;
	};

int main()
{
	LuniPeAn OLuna, AltaLuna;
```

```
	Data DataNasterii, DataOarecare;

	OLuna = Ian;
	AltaLuna = Oct;

	DataNasterii. NumeZi = Luni;
	DataNasterii.Zi = 25;
	DataNasterii.Luna = AltaLuna;
	DataNasterii.An = 1982;

	DataOarecare = DataNasterii;

	cout << "Anul nasterii: " << DataNasterii.An << endl;
	_getch();
	return 0;
}
```

Un caz special de utilizare a tipului *enumerat* este pentru a defini un tip de date *boolean*, pe care compilatoarele mai vechi de *C++* nu-l au implementat:

```
		enum Boolean {False, True};
			......
		Boolean Flag;
```

În versiunile mai noi de compilatoare există tipul de date *bool*. O variabilă de tip *bool* poate lua valorile *true* sau *false*, care convertite în *int* devin, atenţie!, *0* sau respectiv *1*.

În continuare se prezintă o aplicaţie a structurilor de date, în care este disponibil un pachet de cărţi, care urmează a se amesteca. Rulaţi programul pas cu pas şi urmăriţi structura pachetului de cărţi şi felul în care se modifică acesta.

```
/*****************************
Structuri de date:
Declaratii si incluziuni
carti.hpp
*****************************/

#include <iostream>
#include <stdlib.h>
#include <conio.h>

enum Culori {Inima, Frunza, Romb, Trefla};
struct Carte
	{
	    int Numar;
	    Culori Culoare;
	};

const int Jack = 11;
const int Dama = 12;
const int Rege = 13;
const int As = 14;

void Init(Carte *Pachetul);
//Initializarea pachetului
void AfisareCarte(Carte c);
void AfisarePachet(Carte *Pachetul);
void Amestec(Carte *Pachetul);
//Amesteca pachetul
```

```
/*****************************
Structuri de date
Functia principala
carti.cpp
*****************************/

#include "Carti.hpp"
using namespace std;
void main()
{
    Carte Pachet[52];
    //Pachet este un sir de 52 de carti

    Init(Pachet);
    cout << "Pachetul de carti ordonat:" <<
endl;
    AfisarePachet(Pachet);
    Amestec(Pachet);
    cout <<endl <<"Pachetul de carti
amestecat:";
    cout << endl;
    AfisarePachet(Pachet);
    _getch();
}
```

```
/*****************************
Structuri de date
functii.cpp
*****************************/
#include "carti.hpp"
using namespace std;

void Init(Carte* Pachetul)
//Initializeaza pachetul de carti
{
        int num;
        int j = 0;

        for (num=2; num<=14; num++)
          {
           Pachetul[j].Numar = num;
           Pachetul[j].Culoare = Inima;
           Pachetul[j+13].Numar = num;
           Pachetul[j+13].Culoare = Frunza;
           Pachetul[j+26].Numar = num;
           Pachetul[j+26].Culoare = Romb;
           Pachetul[j+39].Numar = num;
           Pachetul[j++ +39].Culoare= Trefla;
                }
};

void AfisarePachet(Carte *Pachetul)
//Afiseaza toate cartile din pachet
{
        for(int j=0; j<52; j++)
        {
            AfisareCarte(Pachetul[j]);
            cout << " ";
            if ( !((j+1)%13) )   cout << endl;
          //salt la rand nou la fiecare 13 carti
        }
};
```

```
void AfisareCarte(Carte c)
//Afiseaza o carte din pachet
{
    if(c.Numar>=2 && c.Numar<=10) cout <<
c.Numar;
    else switch(c.Numar)
        {
            case Jack: cout << "J"; break;
            case Dama: cout << "Q"; break;
            case Rege: cout << "K"; break;
            case As:   cout << "A"; break;
        }
    switch(c.Culoare)
        {
            case Inima:  cout << "i"; break;
            case Frunza: cout << "f"; break;
            case Romb:   cout << "r"; break;
            case Trefla: cout << "t"; break;
        }
};

void Amestec(Carte *Pachetul)
/*Amesteca pachetul:
Fiecare carte este interschimbata cu o alta
din pachet aleasa in mod aleator */
{
  for (int j=0; j<52; j++)
  {
double i;
i = double(rand())/(double)(RAND_MAX+1);
        int k = int (52*i);
        Carte Temp = Pachetul[j];
        Pachetul[j] = Pachetul[k];
        Pachetul[k] = Temp;
    }
};
```