RO: Constructori. Destructori. Tablouri de obiecte.

EN: Constructors. Destructors. Object arrays.

Obiective:

- Înţelegerea teoretică şi practică a noţiunilor de constructori, destructor, tablouri de obiecte.
- Scrierea de programe simple, după modelul programării prin abstractizare a datelor, care exemplifică noţiunile menţionate mai sus.

Objectives:

- Understanding the constructors, destructor and object arrays.
- Writing some simple OOP programs that use the notions mentioned above.

Rezumat:

Constructorii pot fi:

- impliciţi
- expliciţi
  - vizi: fară nici un argument
  - cu parametri: iniţializează anumite variabile din clasă şi/sau efectuează anumite operaţii conform variabilelor primite ca argumente.

Incepand cu *C++1y* s-a introdus posibilitatea de a pastra constructorul implicit vid, ca si constructor *default* in cazul in care se defineste un constructor explicit cu parametrii in clasa, cu constructia:
Class_name( ) = default, unde *Class_name* e numele clasei.

Un tip special de constructor este *constructorul de copiere*, a cărui menire este de a crea copia unui obiect. Constructorii de copiere sunt obligatorii doar în cazul în care clasa respectivă conține atribute de tip pointeri ce necesită alocare dinamică pentru a se rezerva spaţiu. Mai sunt si constructori speciali de *conversie*.

Destructorul :

- poate fi definit: dacă se doreşte efectuarea anumitor operaţii în momentul distrugerii obiectului. In cazul atributelor de tip pointeri alocați dinamic, se definește un destructor care realizează eliberarea acestor pointeri.
- poate să lipsească: se apelează un destructor implicit de către compilator.

Pointerul *this* pointează spre membrii instanţei curente a clasei (este apelabil doar din interiorul clasei). Este util mai ales când în interiorul unei metode dintr-o clasă este necesar să facem distincţie între variabile ce aparţin unor obiecte diferite, pointerul *this* indicând întotdeauna obiectul curent.

Pentru a crea tablouri de obiecte avem nevoie de un constructor fără parametrii (implicit vid sau unul explicit echivalent, inclusiv *explicit default*). Dacă se foloseşte un constructor cu parametri, fiecare obiect din tablou poate fi iniţializat indicând o listă de iniţializare.

Exemple:

```
//1. Implementarea clasei Rectangle cu constructor default explicit si tablouri de obiecte pentru
//efectuarea unor operatii elementare cu forme geometrice dreptunghiulare
// Rectangle.h - clasa Rectangle
class Rectangle {
        //private members implicit
        int height;
        int width;
public:
        // public members
        Rectangle(int h , int w );//explicit constructor
        Rectangle() = default;//since C++1y, to define arrays
        int det_area(void);
        void setHeight(int);
        int getHeight() { return height; }
```

```
        void setWidth(int);
        int getWidth() { return width; }
        ~Rectangle(void); // explicit destructor
};

Rectangle:: Rectangle(int h, int w)     // define explicit constructor
{
        height = h;
        width = w;
}

Rectangle:: ~Rectangle(void)    // destructor
{
        cout << "\nCall destructor...";
        height = 0;
        width = 0;
}

int Rectangle:: det_area(void) {
        return height * width;
}

void Rectangle:: setHeight(int init_height)
{
        height = init_height;
}
void Rectangle:: setWidth(int init_width)
{
        width = init_width;
}

//main
#include <iostream>
using namespace std;
#include "Rectangle.h"

int main()
{
        int i;
cout << "\n\nArray of objects initialized at declaration using the explicit constructor\n";
        Rectangle group1[4] = {
                Rectangle(10,10),
                Rectangle(20,10),
                Rectangle(30,10),
                Rectangle(40,10)
        };

        for (i = 0; i < 4; i++)
        cout << "\nRectangle area: " << group1[i].det_area() <<" with sides, width= " <<
group1[i].getWidth() << " and height= " << group1[i].getHeight() << endl;
        cout << "\n.....\n\n";
        // objects array
        Rectangle group2[4];// explicit default constructor
cout << "\nArray of objects assigned with set methods using the explicit default constructor \n";
        for (i = 0; i < 4; i++){
        group2[i].setHeight(i + 10);
        group2[i].setWidth(10);
                }
        for (i = 0; i < 4; i++)
```

```
        cout << "\nRectangle area : " << group2[i].det_area() << " with sides, width= " <<
group2[i].getWidth() << " and height= " << group2[i].getHeight() << endl;
        cout << "\n.....\n\n";

        // dynamic array
        Rectangle* group3 = new Rectangle[4];//explicit default constructor
        cout << "\nDynamic Array of objects assigned with set methods \n";
        for (i = 0; i < 4; i++){
        (group3 + i)->setHeight(i + 10);
        (group3 + i)->setWidth(10);
        }
        for (i = 0; i < 4; i++)
        cout << "\nRectangle area : " << group3[i].det_area() <<" with sides, width= " <<
group3[i].getWidth() << " and height= " << group3[i].getHeight() << endl;
        delete[] group3;
        cout << "\n.....\n\n";
        // dynamic array
        Rectangle* group4 = new Rectangle[4];//explicit default constructor
        cout << "\n Array of objects assigned with constructor with parameters\n";
        group4[0] = Rectangle(5, 10);
        group4[1] = Rectangle(15, 20);
        group4[2] = Rectangle(25, 30);
        group4[3] = Rectangle(35, 40);
        for (i = 0; i < 4; i++)
cout << "\nRectangle area : " << (group4 + i)->det_area() <<" with sides, width= " << (group4 + i)
->getWidth() << " and height= " << (group4 + i) ->getHeight() << endl;
        delete[] group4;
        cout << "\n.....\n\n";
}//main

//2.  Implementarea unei clase numita Stiva pentru simularea lucrului cu stiva
//Stiva.h

const int dim_char = 256;

class Stiva {
        //membri privati
private:
        int dim;
        char* stack;
        int next;
        //membri publici
public:
        Stiva();
        Stiva(int);
        Stiva(const Stiva&);
        ~Stiva();

        int push(char c);
        int pop(char& c);
        int isEmpty(void);
        int isFull(void);
};

// constructori
Stiva::Stiva() {
        next = 0;
        dim = dim_char;
        stack = new char[dim];
}
```

```
Stiva::Stiva(int dim_i) {
	next = 0;
	dim = dim_i;
	stack = new char[dim];
}

//constructor de copiere
Stiva::Stiva(const Stiva& instack) {
	next = instack.next;
	dim = instack.dim;
	stack = new char[instack.dim];
	for (int i = 0; i < instack.next; i++)
		stack[i] = instack.stack[i];
}

// destructor
Stiva :: ~Stiva() {
	delete[] stack;
}

// test stiva goala
int Stiva::isEmpty() {
	if (next <= 0)return 1;
	else return 0;
}

// test stiva plina
int Stiva::isFull() {
	if (next > dim) return 1;
	else return 0;
}

// introducere in stiva
int Stiva::push(char c) {
	if (isFull())return 0;
	*(stack + next) = c;
	next++;
	return 1;
}

// extragere din stiva
int Stiva::pop(char& c) {
	if (isEmpty())return 0;
	next--;
	c = *(stack + next);
	return 1;
}

// program de test
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include "Stiva.h"

int main() {
	unsigned int i;
	char buf1[dim_char], buf2[dim_char], buf3[dim_char];
	strcpy_s(buf1, "Bafta in sesiune !");
	cout << endl << "Sir initial: " << buf1 << endl;
```

```
	Stiva mesaj1;
	for (i = 0; i < (strlen(buf1)); i++)
		mesaj1.push(buf1[i]);
	i = 0;
	while (!mesaj1.isEmpty())
		mesaj1.pop(buf2[i++]);
	buf2[i] = '\0';
	cout << endl << " Push() -> Pop() -> Inversare: " << buf2 << endl;

	// repopulare mesaj1
	for (i = 0; i < (strlen(buf1)); i++)
		mesaj1.push(buf1[i]);

	// constructor de copiere -init
	Stiva mesaj2(mesaj1);
	i = 0;
	while (!mesaj2.isEmpty())
		mesaj2.pop(buf3[i++]);
	buf3[i] = '\0';
	cout << endl << "Copie obiect precedent: " << buf3 << endl;

	char sTest[15] = "Sir_de_test";
	cout << endl << "Sir de test: " << sTest << endl;

	Stiva mesaj3(strlen(sTest) + 1);//al 2-lea constructor
	for (i = 0; i < (strlen(sTest)); i++)
		mesaj3.push(sTest[i]);

	// constructor de copiere -init cu assign
	Stiva mesaj4 = mesaj3;
	i = 0;
	while (!mesaj4.isEmpty())
		mesaj4.pop(sTest[i++]);
	sTest[i] = '\0';
	cout << endl << "Copie sir initial de test extras cu Pop(): " << sTest << endl;
}//main
//*****************************************************************************
// 3. Exemplu de utilizare a constructorului de copiere pentru un punct caruia i se asociaza un text
//CPunctText.h

const int dim_sir = 20;

class CPunctText {
	int x;
	int y;
	int lungime_sir;//atribut redundant
	char *sNume;
public:
	//constructor explicit vid
	CPunctText( );
	//constructor cu parametri
	CPunctText(int ix, int iy, const char *sText = "Punct");
	//constructor de copiere
	CPunctText(const CPunctText &pct);
	//destructor:
	~CPunctText( );
	void afis() {
		cout << "\nObiectul are x= " << x;
		cout << "\nObiectul are y= " << y;
```

```
                cout << "\nObiectul are sirul =  " << sNume;
        }//afis
};

CPunctText::CPunctText( ) {
        cout << "\n constructor explicit vid";
        lungime_sir = dim_sir;
        sNume = new char[lungime_sir];
}

CPunctText::CPunctText(int ix, int iy, const  char *sText) {
        cout << "\n constructor cu parametri";
        lungime_sir = strlen(sText) + 1;// pentru |0
        sNume = new char[lungime_sir];
        x = ix;
        y = iy;
        strcpy(sNume, sText);
}

CPunctText::CPunctText(const CPunctText &pct) {
        cout << "\n constructor de copiere";
        sNume = new char[pct.lungime_sir];
        x = pct.x;
        y = pct.y;
        lungime_sir = pct.lungime_sir;
        strcpy(sNume, pct.sNume);
}

CPunctText::~CPunctText() {
        cout << "\n destructor";
        delete[ ] sNume;
}

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

#include "CPunctText.h"

int main( ) {
        CPunctText cpt1(1, 2, "Punct1");//apel constructor cu parametri
        CPunctText cpt2(cpt1);       //apel constructor de copiere
        CPunctText cpt3 = cpt2;       //apel constructor de copiere
        CPunctText cpt4(4, 5);       //apel constructor cu parametri

        cpt3.afis();
        cpt4.afis();
        }//main
//Recomandare: introduceti metode de tip getter in locul metodei afis() -vezi tema 1
```

Teme:

1. Modificaţi exemplul 3 astfel încât să permită obtinerea unui nou punct, avand coordonatele obtinute prin adunarea coordonatelor a două astfel de puncte. Numele noului punct va fi rezultat prin concatenarea numelor celor două puncte. Adaugati si testati o metoda care calculeaza distanta de la un punct la origine. Modificați clasa astfel încât sa eliminați metoda *afis()* folosind in schimb metode accesor adecvate. Eliminați de asemenea atributul *lungime_sir* modificând adecvat metodele clasei. Testati utilizand si functii specifice sirurilor de caractere din VC++1y/2z (*strcpy_s( ) si strcat_s( )*).

2. Să se scrie o aplicație C/C++ care să modeleze obiectual un tablou unidimensional de numere reale. Creați două instanțe ale clasei și afișați valorile unui al 3-lea tablou, obținute prin scăderea elementelelor corespunzătoare din primele 2 tablouri. Dacă tablourile au lungimi diferite, tabloul rezultat va avea lungimea tabloului cel mai scurt.
3. Modelați clasa *Student* care să conțină atributele private *nume, prenume, note (tablou 7 valori int), grupa*. Alocați dinamic memorie pentru *n* studenți. Calculați media cu o metoda din clasa și sortați studenții după medie, afisând datele fiecărui student (*nume, prenume, grupa, medie*). Implementati si destructorul clasei care să afișeze un mesaj.
4. Să se scrie o aplicaţie în care se modelează clasa *Student* cu *nume, prenume, numar note* si *notele* din sesiunea din iarnă declarat printr-un pointer de tip *int*. Să se afișeze numele studenţilor din grupă care au restanțe și apoi numele primilor 3 studenți din grupă în ordinea mediilor, care se va afisa si ea.
5. Să se scrie o aplicaţie C/C++ în care se citeşte de la tastatură un punct prin coordonatele *x, y, z*. Să se scrie o metodă prin care să se facă translaţia punctului cu o anumită distanţă pe fiecare dintre cele trei axe. Să se verifice dacă dreapta care unește primul punct și cel rezultat în urma translaţiei trec printr-un al treilea punct dat de la consolă.
6. Definiți o clasă *Complex* modelată prin atributele de tip double *real, imag* și un pointer de tip *char* către numele fiecărui număr complex. În cadrul clasei definiți un constructor explicit cu doi parametri care au implicit valoarea 1.0 și care alocă spațiu pentru *nume* un șir de maxim 7 caractere, de exemplu *"c1"*. De asemenea, definiți un constructor de copiere pentru clasa *Complex*. Clasa va mai conține metode mutator/setter și accesor/getter pentru fiecare membru al clasei, metode care permit operațiile de bază cu numere complexe și un destructor explicit. Definiți cel mult 10 numere complexe într-un tablou. Calculați suma numerelor complexe din tablou, valoare ce va fi folosită pentru a inițializa un nou număr complex, cu numele *"Suma_c"*. Realizați aceleași acțiuni făcând diferența și produsul numerelor complexe.
7. Consideram clasa *Fractie* care are doua atribute intregi *private a* si *b* pentru numarator si numitor, doua metode de tip *set( )* respectiv *get( )* pentru atributele clasei publice si o metoda *simplifica( )* publica care simplifica obiectul curent *Fractie* de apel, returnând un alt obiect simplificat. Metoda *simplifica( )* va apela o metoda *private cmmdc( )* pentru simplificarea fracției. Definiți un constructor explicit fara parametri care initializeaza *a* cu 0 si *b* cu 1, si un constructor explicit cu doi parametri care va fi apelat dupa ce s-a verificat posibilitatea definirii unei fractii (*b*!=0). Definiți o metoda *aduna_fractie( )* care are ca si parametru un obiect de tip *Fractie* si returneaza *suma* obiectului curent de apel cu cel dat ca si parametru, ca si un alt obiect de tip *Fractie*. Analog definiti metode pentru *scadere, inmultire* si *impartire*. Instantiati doua obiecte de tip *Fractie* cu date citite de la tastatura. Afisati atributele initiale si cele obtinute dupa apelul metodei *simplifica( )*. Efectuati operatiile implementate prin metodele clasei si afisati rezultatele.
8. Considerand problema precedenta adaugati in clasa *Fractie* un atribut pointer la un sir de caractere, *nume* care va identifica numele unei fractii. Constructorul fara parametri va aloca dinamic un sir de maxim 20 de caractere initializat cu numele, *"Necunoscut"*, cel cu parametrii va contine un parametru suplimentar cu numele implicit, *"Necunoscut"* care va fi copiat in zona rezervata ce va fi de doua ori dimensiunea sirului implicit. Pentru acest atribut se vor crea metode accesor si mutator, care sa afiseze numele unui obiect de tip *Fractie* respectiv care sa poata modifica numele cu un nume specificat (*Sectiune_de_aur, Numar_de_aur*, etc.). De asemenea se va implementa si un copy constructor si un destructor. In programul principal instantiati doua obiecte de tip *Fractie*, unul folosind constructorul fara parametri, celalalt folosind constructorul cu parametri, valorile parametrilor fiind introduse de la tastatura. Modificati atributele primului obiect, folosind metode de tip *setter*. Initializati un al treilea obiect de tip *Fractie* folosind copy constructorul. Afisati atributele acestui obiect obtinut folosind metodele de tip *getter*.

Homework:

1. Modify example 3 in order to allow the addition of two *CPunctText* points. The name of the new point will be created from the names of the compounding points. Add a method that returns the distance from a point to origin. Modify the class so that you remove the *afis ()* method by using appropriate getter methods instead. Also remove the *lungime_sir* attribute by appropriately modifying the class methods. Test using the string specific functions of VC ++ 1y/2z (*strcpy_s ( )* and *strcat_s ( )*).
2. Write a C/C++ application that models in OOP a real numbers one dimensional array. Instantiate two objects of this class with the same length *n* and store in a third one the results

of subtracting each of the two real number arrays' elements. If the source arrays have different lengths, the result has the length of the shortest array.

3. Create a class named *Student* that has as private attributes the *name*, *surname*, some *marks (array 7 int values)*, the *group*. Allocate the necessary amount of memory for storing *n* students. Determine the average mark with a method from the class for each student and use it for sorting the students. Display the ordered array (*name, surname, group, average_mark*). The destructor will display a message.
4. Model in OOP a class named *Student* containing *name*, *surname* the *number of marks* and the *marks* from the winter session exams specified as an *int* pointer. Display the name of the students who have arears exams and the first three students in the group based on the media that will be also displayed.
5. Write a C/C++ application that reads a point from the keyboard by giving the *x, y* and *z* coordinates. Write a method that moves the point with a given distance on each of the three axes. Verify if the line between the first and the second position of the point crosses a third given point.
6. Define a class called *Complex* that stores the double variables *real, imag* and a pointer of character type that holds the name of the complex number. Define an explicit constructor with 2 parameters that have 1.0 as implicit value. The constructor also initializes the pointer with a 7 characters wide memory zone. Define a copy constructor for this class. Implement the mutator/setter and accessor/getter methods for each variable stored inside the class. All the operations related to the complex numbers are also emulated using some specific methods. An explicit destructor method is also part of the class. Define an array of not more than 10 complex numbers. Determine the sum of all the numbers in this array and use this value for initializing a new instance of the class named *complex_sum*. Repeat this action for all the rest of the operations implemented inside the class.
7. Consider a class named *Fraction* that has two private integer attributes *a* and *b* for the denominator and nominator, two *set( )* and *get( )* methods and a method *simplify( )* that will simplify the current calling *Fraction* object and will return as result a *Fraction* object. *simplify( )* method will call a private *cmmdc( )* method to simplify the fraction. Define an explicit constructor without parameters that initializes *a* with 0 and *b* with 1. Define another explicit constructor that receives 2 integer parameters. For this constructor is verified if *b*!=0 before to be called. Define a method named *add_fraction( )* that returns the object obtained by *adding* the current object with the one received as parameter, as a *Fraction* object. Define in the same manner the methods that *subtract, multiply* and *divide* two fractions. Instantiate two *Fraction* objects having the corresponding data read from the keyboard. Display the initial attributes and the ones obtained after simplifying the fractions. Call the methods that apply the implemented arithmetical operations and display the results.
8. Considering the previous task add in the *Fraction* class another attribute consisting in a character array pointer (*name*) that identifies a fraction. The constructor without parameters will allocate a max 20 characters memory zone difined with "*Unknown*", the parameterized constructor will have another last implicit parameter initialized with "*Unknown*" that will represent the fraction's name and the reserved space will be twice the string dimension. Implement setter and getter methods for the *name* attribute. Implement a copy constructor and a destructor. In the *main( )* function create two *Fraction* objects, one using the constructor without parameters and the other using the parameterized constructor. Modify the attributes of the first object using *setter* methods. Create a third object using the copy constructor. Display the attributes of this last object using the getter methods.