RO: Clase, obiecte, membri
EN: Classes, objects, class members

Obiective:

- Înţelegerea teoretică şi practică a noţiunilor de clasă, obiect, membru al unei clase, specificatori de vizibilitate.
- Scrierea de programe simple, după modelul programării obiectuale folosind ADT, care exemplifică noţiunile menţionate mai sus.

Objectives:

- Theoretical and practical undestanding of classes, objects, class members and access specifiers.
- Writing simple programs, following the Object Oriented Programming model using ADT, that exemplify the aspects mentioned above.

Rezumat:

Programarea orientată pe obiecte implică definirea *tipurilor abstracte de date* în vederea definirii obiectelor, a mesajelor precum şi a mecanismului de moştenire. Noţiunile de bază ale OOP se referă în principal la *definirea claselor de obiecte*, a *moştenirii* şi a *polimorfismului*.

Clasele C++ reprezintă tipuri noi de date, care conţin metode (functii membre) şi variabile, un fel de „matriţe" ce sunt folosite pentru a defini metodele şi variabilele unui obiect anume, „turnat" după modelul clasei.

Procesul de creare al unui obiect se numeşte *instanţiere* caz în care pentru fiecare variabilă se vor preciza nişte valori concrete (explicit sau implicit).

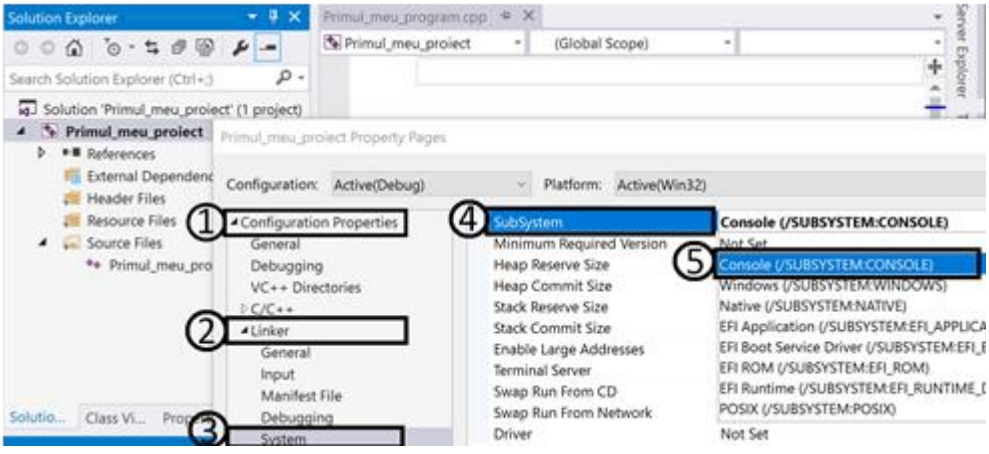
Metodele unei clase pot să fie definite în interiorul ei (metode inline) sau în afară, caz în care este necesară indicarea clasei prin operatorul rezoluţie sau scop (::).

Accesul la variabilele şi metodele membre ale unei clase se poate controla prin utilizarea *specificatorilor de vizibilitate*: ***public, private, protected***.

Funcţiile membre (metode) *constructor* sunt metode care sunt apelate în momentul instanţierii unor obiecte, ele nu au specificat nici un fel de tip returnat, au numele identic cu cel al clasei din care fac parte. Rolul contructorilor este de a iniţializa variabilele din acea clasă şi de a aloca spaţiu de memorie corespunzător obiectului instanţiat şi a variabilelor folosite în cadrul clasei.

Destructorul clasei este o metodă care, analog cu constructorul, poate fi recunoscută prin faptul că are acelaşi nume cu clasa din care face parte, este precedat de caracterul ~ şi are rolul de a elibera spaţiul de memorie alocat pentru obiectul instanţiat şi a variabilelor folosite în cadrul clasei.

Pentru a vedea modul de apel al destructorului la versiunile anterioare VC++19 va trebui sa setati proiectul astfel incat sa ramana pe ecran rezultatele obtinute conform:



Exemple:

```
//1. Implementarea clasei Rectangle pentru efectuarea
//unor operatii elementare cu forme geometrice dreptunghiulare
// Rectangle.h - clasa Rectangle
class Rectangle {
        // membri private
        int height;
        int width;
public:
        // membri publici
        Rectangle(int h = 10, int w = 10); // constructor explicit cu val. implicite
        int det_area(void);
        void setHeight(int);
        void setWidth(int);
        ~Rectangle(void); // destructor explicit
};

Rectangle::Rectangle(int h, int w)     // definire constructor explicit
{
        height = h;
        width = w;
}
Rectangle::~Rectangle(void)    // destructor
{
        cout << "\nApel 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( ) {
        // objects of Rectangle type
        Rectangle rect, square;//inst.cu val implicite
        cout << "\nObiecte(acces cu operatorul .): \n";
        cout << "\tValori implicite: \n";
        cout << "\t\tAria dreptunghiului: " << rect.det_area( ) << endl;
        cout << "\t\tAria patratului: " << square.det_area( ) << endl;
        cout << "\tValori oarecare: \n";
        rect.setHeight(12);
        rect.setWidth(8);
        cout << "\t\tAria dreptunghiului: " << rect.det_area( ) << endl;
        square.setHeight(8);
        square.setWidth(8);
```

```
        cout << "\t\tAria patratului: " << square.det_area( );
        cout << "\n.....\n\n";
// pointer to Rectangle type
        Rectangle *point;
        cout << "\nPointers to objects (access with  operatorul ->):";
        point = new Rectangle;//inst. with implicit values
        cout << "\n\tImplicit values: \n";
        cout << "\t\tArea rectangle(square):" << point->det_area( ) << endl;
        cout << "\tAny values: \n";
        point->setHeight(12);
        point->setWidth(12);
        cout << "\t\tArea (rectangle)square: " << point->det_area( ) << endl;
        delete point;
        cout << "\n.....\n\n";
// reference to type Square
        Rectangle &ref_square = square;
        ref_square.setHeight(17);
        ref_square.setWidth(17);
        cout << "\t\tAria patratului folosind referinte: " << ref_square.det_area( );
        cout << "\n.....\n\n";
        }//main
//*****************************************************************************
```

//2. Implementarea clasei Point

```
#include <stdio.h>//e recomandat a folosi mecanismul de I/E din C++

class Point {
    int x, y;
public :
    Point(int a=0, int b=0) {
        x=a; y=b;
        }
    void setX(int a){
                x=a;
        }
    int getX(){
                return x;
        }
    void setY(int b){
                y=b;
        }
    int getY(){
                return y;
        }
};

int main( ){
        Point p1;           // echivalent cu Point p1(0,0);
        printf("x = %d\t y = %d\n", p1.getX(), p1.getY());//recomandat I/E C++
        Point p2(10);               // echivalent cu Point p2(10,0);
        Point p3(10,10);
        p1.setX(20);
        p1.setY(30);
        //afisati noile valori ale atributelor x si y ale punctului p1
}
```

Cerință: Modificati operatiile de I/E folosind cele specifice C++. Adaugati și testati o metodă pentru calculul razei vectoare (distantei dintre doua puncte).

```
//*****************************************************************************
```
//3. Implementarea clasei *Complex* cu metode accesor si mutator pentru atributele real si imaginar si o metoda care implementeaza operatia de adunare

```
#include <iostream>
using namespace std;

class Complex{
        double re, im;
public:
        Complex(double x=0.0, double y=0.0) {
                re = x;
                im = y;}
        double modul(){
                return sqrt(re*re + im*im);}
        double faza();

        void setRe(double real){
                re=real;}
        double getRe(){
                return re;}
        void setIm(double imaginar){
                im=imaginar;}
        double getIm(){
                return im;}
        void ad_complex(Complex b){//recomandat a returna un obiect
                re += b.re;
                im +=b.im;}

        void sc_complex(Complex b); //recomandat a returna un obiect

};//class

int main( ){
        Complex obiect_1,obiect_2;
        double aux;
        cout <<"\nPartea reala a obiectului 1 este \t";
        cin >> aux;
        obiect_1.setRe(aux);
        cout <<"Partea imaginara a obiectului 1 este \t";
        cin >> aux;
        obiect_1.setIm(aux);
        //completati citirea valorilor reale si imaginare pentru obiectul 2 de tip complex
        //....
        obiect_1.ad_complex(obiect_2);//adunarea celor doua numere se face in obiectul 1-curent
        cout << "\nPartea reala a obiectului 1 este \t\n" <<obiect_1.getRe();
        cout << "\nPartea imaginara a obiectului 1 este \t \n" <<obiect_1.getIm();
                //apelati metoda de scadere dupa definirea ei
        }
```

Cerință: să se adauge și să se testeze o metodă pentru calculul fazei.

//4. Implementarea partiala a clasei *Complex* care returneaza obiect rezultat la efectuarea de operatii:

```
class Complex {
            float re, im;
public:
            void setRe(float x);
            void setIm(float x);
            float getRe();
            float getIm();
            Complex sum(Complex c);
```

```
    Complex difference(Complex c);
    Complex multi(Complex c);
    Complex div(Complex c);
    };

Complex Complex::sum(Complex c) {
Complex rez;
rez.im=(c.im + im);
rez.re=(c.re + re);
return rez;
}
...
```

Teme:

1. Să se scrie o aplicaţie C/C++ care foloseşte o structură de date cu numele *Scerc* care conţine *raza* ca şi o variabilă de tip întreg. Într-un program C/C++, declaraţi două variabile *c1, c2* de tip *Scerc* şi calculaţi aria şi circumferinţa lor pentru valori ale razei introduse de la tastatură cu două metode definite. Aceleaşi cerinţe vor fi implementate în aceiasi aplicaţie folosind o clasă numită *Cerc* cu atributul *raza* de tip *private*, clasă ce va conţine pe lângă metodele de calcul ale ariei şi perimetrului un constructor explicit cu parametru, un destructor şi o metodă de afişare raza.
   Extindeţi aplicaţia astfel încât să definiţi mai multe obiecte de tip *Cerc* la care să accesaţi metodele specifice folosind obiectele instanţiate, pointeri la obiecte, referinţe la obiecte. Introduceţi o metodă de tip accesor, *getRaza()* care permite accesul la data privată *raza* si care sa o folositi pentru a afisa in *main()* raza obiectelor.
2. Să se definească o clasă numită *myString* (într-un fişier numit *strClass.h*) care să fie compusă din metodele specifice care efectuează următoarele operaţii pe şiruri de caractere:
   - determină lungimea şirului primit la intrare.
   - determină ultima poziţie de apariţie a unui anumit caracter din şirul de intrare.
   - returnează şirul primit la intrare, scris cu caractere majuscule.
   - returnează şirul primit la intrare, scris cu caractere minuscule.
   - returnează numărul de apariţii ale unui anumit caracter din şirul primit.
3. Să se scrie programul care citeşte de la tastatură un şir de maxim 10 caractere şi care, pe baza clasei implementate anterior, efectuează asupra şirului de intrare operaţiile definite în cadrul clasei.
4. Să se scrie programul care implementează clasa *Numar* cu un atribut de tip *int* val şi care, în cadrul funcţiei *main()*, declară un obiect de tipul clasei şi apoi un pointer la acesta, prin intermediul cărora se va afişa pe ecran rezultatul adunării a două numere de tip *Numar* cu valorile preluate de la tastatură in cadrul unor obiecte *Numar*. Implementati metoda *int suma_nr(Numar)* care realizeaza suma în cadrul clasei şi o returneaza ca un *int*, metoda care insumeaza cele doua obiecte (curent si parametru). Implementati metoda in cadrul clasei si alta metoda cu acelas scop, dar nume diferit, in afara clasei.
5. Să se definească o clasă care implementează metodele:
   - *int plus(int x, int y)***,** care returnează suma valorilor primite la apelul metodei;
   - *int minus(int x, int y)***,** care returnează diferenta valorilor primite la apelul metodei;
   - *int inmultit(int x, int y)***,** care returnează produsul valorilor primite la apelul metodei;
   - *float impartit(int x, int y)***,** care returnează catul valorilor primite la apelul metodei;

   şi apoi să se scrie aplicaţia care utilizează această clasă. Considerati si cazul in care in cadrul clasei aveti atributele de tip *int x* si *y*, caz in care metodele nu vor mai avea parametrii.
   Observaţie: În cazul împărţirii, trebuie verificată validitatea operaţiei (împărţitor diferit de zero). În cazul în care operaţia este imposibilă, trebuie afişat un mesaj de eroare.
6. Să se creeze o clasă care să modeleze numerele complexe. Scrieţi un program care utilizează această clasă si defineste doua obiecte afisand caracteristicile obiectelor si rezultatele operatiilor definite (Folositi exemplul 3 cu rezultat in obiectul curent).
7. Să se scrie un program care implementează clasa *Aritmetica* cu două atribute *a* și *b* de tip numeric (*int, float* sau *double*) si metode setter si getter adecvate. Implementați metoda *suma()* în interiorul clasei și metoda *diferenta()* ce apartine de asemenea clasei, dar e definita în afara clasei, metode care vor fi apelate prin intermediul unui obiect al clasei *Aritmetica*. În funcția principală *main()* instanțiați trei obiecte de tip *Aritmetica*. Modificati atributele a si b la fiecare obiect in parte folsind metodele de tip setter. Aplicați asupra lor operațiile de adunare și

scădere pe care le-ați implementat prin metodele *suma()* și *diferenta()*. Metodele returneaza valorile numerice corespunzatoare operatiei folosind cele doua atribute ale clasei valori ce le veti afisa in *main()*. La fiecare grup de operatii adunare/scadere afisati valorile atributelor obiectului folosind metodele de tip getter.

8. Pornind de la clasa *Complex, ex.4,* să se implementeze operațiile de adunare, scădere, înmulțire și împărțire pentru numere complexe prin metode corespunzătoare implementate la alegere în clasă și/sau în afara ei. Testați aceste metode prin instanțierea unor obiecte. Metodele vor returna obiecte de tip *Complex* si in *main()* vor fi afisate rezultatele folosind metode accesor.
9. Declarati o clasa *Fractie* care are doua atribute intregi de tip *private a* si *b* pentru numarator si numitor. Definiti doua metode de tip *set()* respectiv *get()* pentru fiecare din atributele clasei. Instantiati doua obiecte de tip *Fractie* si afisati atributele initiale si cele obtinute dupa folosirea metodelor *set()*. Definiti o metoda *simplifica()* apelata cu un obiect pentru care au fost apelate metodele de tip *set(),* care determina divizorii numitorului si numaratorului, ii afiseaza si realizeaza simplificarea fractiei, afisand in metoda si rezultatul obtinut (noua fractie numarator/numitor).

Homework:

1. Write a C/C++ application that uses a data structure named *Scircle*. The structure has an integer variable that represents the circle's *radius*. In the *main()* function that uses the *Scircle* structure, declare 2 *Scircle* variables (*c1* and *c2*) and calculate the area and the perimeter (using 2 methods) for 2 circles with the value of the *radius* read from the keyboard.

Perform the same operations using a class called *Circle* (the *radius* is a *private* member of the class). The class contains, besides those 2 methods defined in the first application for area and perimeter, an explicit constructor with 1 parameter, a destructor and another method *displayRadius()* that will be display the radius.

Extend the application by defining several *Circle* objects, pointers to the declared objects and refferences to those objects.

Introduce an accessor method, *getRadius()* method that allows access to the private *radius* attribute to display in the *main( )* the radius of objects.

2. Declare a class called *String* and store it in a file named *strClass.h*. The methods in the class perform the following tasks:

a. determine the length of the array of characters received as parameter;

b. determine the index of the last occurence of a certain character in the array;

c. return the array of characters received as parameter, all letters being transformed into capital letters.

d. return the number of occurences of a certain character;

3. Write a program that reads from the keyboard an array of maximum 10 characters and performs all the operations implemented in the class defined at problem 3.

4. Write a program that implements a class called *Number* with an attribute of *int* type, val. In function *main()*, declare two *Number* objects and a pointer to one object of this type, used to call a method *int sum_nr(Number)* from the class that will calculate and return the sum of 2 *int* numbers read from the keyboard, associated to the instantiated objects (current, parameter). Implement the method inside and outside the class (with different names).

5. Define a class that implements the following methods:

- *int plus(int x, int y)***,** which returns the sum of *x* and *y*;
- *int minus(int x, int y)***,** which returns the difference between *x* and *y*;
- *int inmultit(int x, int y)***,** which returns the result of *x* multiplied by *y*;
- *float impartit(int x, int y)***,** which returns the quotient of *x* and *y*;

Write the application that uses this class. Consider also the case that the class contains two *int* attributes, *x* and *y*, and the class methods will have no parameters.

Remark: prevent the division of a number by 0. If this situation occurs, display an error message.

6. Define a class that manages complex numbers. Write the program that uses this class considering 2 objects and displays the object's characteristics and the results of the defined operations. (Use exemple 3 with result in the current object).

7. Write a program that implements a class named *Arithmetics* that has two numeric (*int, float* or *double*) attributes *a* and *b*. The class will contai setter and getter methods for attributes. Inside the class, implement a method named *sum( ).* The method *difference()* that also belongs to the class is

implemented outside the class, methods that will be called using *Arithmetics* objects. Create 3 objects instantiated from the *Arithmetics* class and make use of the defined methods. The methods return the numerical values corresponding to the operation using the two attributes of the class results that will be displayed in *main()*. For each addition / subtraction operation group, display the values of the object attributes using the getter methods.

8. Starting with the *Complex* class ex.4, implement the addition, subtraction, multiplication and division of complex numbers. The methods can be implemented inside or outside the class. Test the defined methods by using them upon some created objects. The methods will return *Complex* objects and in *main()* with getter methods will be displayed the results.

9. Declare a class named *Fraction* that has two *private* integer attributes *a* and *b* representing a fraction's numerator and denominator. Define two setter and getter methods for the class's attributes. Create 2 *Fraction* instances and display the initial attributes and the ones established after using the setter methods. Define a method named *simplify()* that determines and displays all the common dividers of the nominator and denominator, simplifies the fraction and prints in the method the final result (new fraction numerator/ denominator) .