RO: Functii si clase prietene. Membri statici.

EN: Friend functions and classes. Static members.

Obiective:

- Capacitatea de a utiliza funcţii şi clase prietene (friend) şi membri statici (variabile şi metode) ai unei clase;
- Înţelegerea practică a acestor noţiuni prin implementarea de programe.

Objectives:

- Using friend functions and classes, static class members (variables and methods);
- Practical understanding of the notions mentioned above (programs implementation).

Rezumat:

Funcţia prietenă (*friend*) este o funcţie care poate accesa oricare atribut al unei clase, inclusiv cele private, din afara acelei clase prin intermediul unui obiect al clasei. Existenţa lor se justifică deoarece în unele situaţii scutesc programatorul de munca derivării şi adăugarii unei noi funcţionalităţi. Funcţia *friend* poate fi la rândul ei membră a altei clase.

O clasă prietenă (*friend*) este o clasă care are acces la toti membrii unei alte clase.

Funcţionalitatea specificatorului *static* se referă la existenţa unei singure copii a unui membru, în cazul în care există mai multe obiecte din clasa respectivă. O particularitate în cazul folosirii variabilelor statice este necesitatea redeclarării(alocarii si initializarii) lor în exteriorul clasei. Metodele statice au o serie de caracteristici restrictive:

- au acces doar la alti membri de tip static ai clasei şi bineînţeles, pot lucra cu membri globali;
- nu pot avea pointeri *this*;
- nu pot exista două versiuni ale aceleiaşi metode, una statică şi una ne-statică.

Exemple:

```
//Exemplul 1- functii friend – echivalent functie globala
//Media.h
class MyClass {
private:
	int x, y;
public:
	friend double media(MyClass);
	void setX(int x) {
		this->x = x;
	}
	int getX() {
		return x;
	}
	void setY(int y) {
		this->y = y;
	}
	int getY() {
		return y;
	}
};
double media(MyClass x) {		// implementarea  functiei friend
	double rez;
	rez = (x.x + x.y) / 2.;
	return rez;
}

double media_global(MyClass x) {  // implementarea  functiei globale
	double rez;
	rez = (x.getX() + x.getY()) / 2.;
	return rez;
}
```

```
//main()
#include <iostream>
using namespace std;
#include "Media.h"

int main() {
	MyClass ob1;

	int x, y;
	cout << "\nValoarea lui x: ";
	cin >> x;
	cout << "\nValoarea lui y: ";
	cin >> y;
	ob1.setX(x);
	ob1.setY(y);
	cout << "\nMedia aritmetica a celor doua numere cu functie friend este: " << media(ob1);
	cout << "\nMedia aritmetica a celor doua numere cu functie globala este: " << media_global(ob1);
}

//*****************************************************************************
//Exemplul 2 – operatii de adunare pentru numere complexe cu functii prieten
//Complex.h

class Complex {
	double r, i;
public:
	Complex( ) {
		r = 0.;
		i = 0.;
	}
	void setR(double re) {
		r = re;
	}
	void setI(double im) {
		i = im;
	}
	double getR( ) {
		return r;
	}
	double getI( ) {
		return i;
	}
	friend Complex operator_plus(Complex, Complex);
};//class

Complex operator_plus(Complex j, Complex k) {
	Complex l;
	l.r = k.r + j.r;
	l.i = k.i + j.i;
	return l;
}

//main()
#include <iostream>
using namespace std;
#include "Complex.h"

int main( ) {
	Complex a, b, c;
	a.setR(10.);
	a.setI(20.);
	b.setR(20.);
	b.setI(30.);
```

```
	c = operator_plus(a, b);
	cout << "Suma este: " << c.getR( ) << "+i*" << c.getI( ) << endl;
	}

//*****************************************************************************
//Exemplul 3.a - variabile statice public
//MyClass.h
class MyClass
{
	int x;
public:
	static int n; //public
	MyClass(int v) {
		cout << "\nApel constructor cu valoarea: " << v << endl;
		x = v;
		n++; }
	int getX( ) { return x; }
		~MyClass() {
		n--;
		cout << "\nApel destructor: n = " << n << endl;
	}
};
int MyClass::n;   // re-declarare variabila static ce va fi astfel vizibila, =0

//main( )
#include <iostream>
using namespace std;
#include "MyClass.h"

int main( ) {
	cout << "Acces prin numele clasei: n = " << MyClass::n << endl;
		MyClass a(3);
	cout << "Date membre: " << "x = " << a.getX( ) << ", n = " << MyClass::n << endl;

	cout << "\nAcces prin obiectul a: n = " << a.n << endl;
		MyClass b(5);
		cout << "Date membre: " << "x = " << b.getX( ) << ", n = " << b.n << endl;
	cout << "\nAcces prin numele clasei: n = " << MyClass::n << endl;
}
//Exemplul 3.b - variabile statice private si metode statice public
//MyClass.h

class MyClass {
	int x;
	static int y;//private static attribute
```

```
public:
	void setX(int a) {
		x = a;
	}
	static void setY(int b) {
		y = b;
	}
	int getX( ) {
		return x;
	}
	static int getY( ) {//atribut static accesat
		return y;
	}
};

int MyClass::y;   //variabila statica va fi astfel vizibila  si initializata cu 0

//main( )
#include <iostream>

using namespace std;

#include "MyClass.h"

int main( ) {
	MyClass ob1, ob2;
	ob1.setX(1);
	ob1.setY(2);//nerecomandat
	MyClass::setY(3);
	cout << "\nValoarea lui x(nonstatic): " << ob1.getX( );
	cout << "\tValoarea lui y(static) prin class: " << MyClass::getY( );
	cout << "\nValoarea lui y(static) prin obiect: " << ob1.getY( );//nerecomandat
	ob2.setX(5);
	ob2.setY(6);//nerecomandat
	MyClass::setY(7);
	cout << "\nValoarea lui x(nonstatic): " << ob2.getX( );
	cout << "\tValoarea lui y(static) prin class: " << MyClass::getY( );
	cout << "\nValoarea lui x(nonstatic): " << ob1.getX( );
	cout << "\tValoarea lui y(static) prin class: " << MyClass::getY( );
	}
//*****************************************************************************************
//Exemplul 4 - metoda statica v_calend actioneaza asupra atributelor statice zz,ll,aa din clasa Dc
//Dc.h

class Dc {
	int zi, luna, an;
	static int zz, ll, aa;
public: Dc(int z = 15, int l = 4, int a = 2019) {
	zi = z;
	luna = l;
	an = a;
	}
	static int v_calend(Dc *d);
};//Dc

int Dc::zz, Dc::ll, Dc::aa;

int Dc::v_calend(Dc *d) {
	static int t_nrz[ ] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };
	int z, l, a;
	if (d->an<1600 || d->an>4900) return 0;
	if (d->luna<1 || d->luna>12) return 0;
if (d->zi<1 || d->zi>t_nrz[d->luna] + (d->luna == 2 && (d->an % 4 == 0 && d->an % 100 || d->an % 400 == 0)))
return 0;
```

```
        z = Dc::zz = d->zi;
        l = Dc::ll = d->luna;
        a = Dc::aa = d->an;
        //...
        if (a<1600 || a>4900) return 0;
        if (l<1 || l>12) return 0;
        if (z<1 || z>t_nrz[l] + (l == 2 && (a % 4 == 0 && a % 100 || a % 400 == 0))) return 0;
        return 1;
}//v_calend

//main( )
#include <iostream>

using namespace std;

#include "Dc.h"

int main(  ) {
        int z, l, a;
        cout << "\n Dati ziua= ";
        cin >> z;
        cout << "\n Dati luna= ";
        cin >> l;
        cout << "\n Dati anul= ";
        cin >> a;
        Dc data_calend(z, l, a);
        if (Dc::v_calend(&data_calend) == 0) cout << "\n Data eronata";
        else  cout << "\n Data  este corecta";
        return 0;
}//main

//*************************************************************************
//Exemplul 5 - clasa Obiect in care se defineste variabila contor var_static de tip static intreg

class Obiect {
        int valoare;
        static int var_static;
        public:
        Obiect(int v) {
                valoare = v;
                var_static ++;
        }
        int getValoare( ) {
                return valoare;
        }
        static int getVar_static( ) {
                return var_static;
        }
};
int Obiect:: var_static;//re-declarare si initializare implicita cu 0
```

Teme:

1. Construiţi o aplicaţie în care clasa *OraCurenta* are ca atribute *private* ora, minutele şi secundele şi metode *publice* de tip set/get pentru atributele clasei. Adaugaţi o functie *friend* clasei prin care să se poată copia conţinutul unui obiect *OraCurenta* dat ca si parametru, într-un alt obiect instanţă a aceleiaşi clase care va fi returnat de functie, ora fiind insa modificata la Greenwich Mean Time. Utilizati timpul curent al calculatorului.
2. Scrieţi o aplicaţie C/C++ în care clasa *Calculator* are un atribut privat *memorie_RAM* (int) şi o funcţie prietenă *tehnician_service( )* care permite modificarea valorii acestui atribut. Functia *friend* va fi membra intr-o alta clasa, *Placa_de_baza* care are o componentă *denumire_procesor (sir de caractere)*. Scrieţi codul necesar care permite funcţiei prietene *tehnician_service( )* să modifice (schimbe) valoarea variabilei *denumire_procesor* si *memorie _RAM*.
3. Definţi o clasă numită *Repository* care are două variabile private de tip întreg. Clasa mai conţine un constructor explicit vid si unul cu 2 parametri şi o metodă accesor care afişează valorile variabilelor din clasă. Scrieţi o

clasă numită *Mathemathics, friend* cu prima clasă, care implementează operaţiile aritmetice elementare (+, -, *, /) asupra variabilelor din prima clasa. Fiecare metoda primește ca parametru un obiect al clasei *Repository*.

4. Scrieţi o aplicaţie C/C++ care defineşte într-o clasă variabila publică contor *var_static* de tip static întreg. Aceasta se va incrementa în cadrul constructorului. După o serie de instanţieri, să se afişeze numarul de obiecte create (conţinutul variabilei *var_static*).
5. Rezolvați problema 4 în cazul în care variabila statică este de tip *private*. Definiți o metodă accesor care returnează valoarea contorului.
6. Scrieţi o aplicaţie C/C++ în care să implementați clasa *Punct* cu atributele *private x* si *y*. Implementați o funcție *friend* care să calculeze aria și perimetrul a doua forme geometrice definite de două puncte, considerând că aceasta are doua puncte ca si parametrii *P0(x0,y0)* și *P1 (x1, y1)*. Adăugați funcției un parametru întreg *figura* prin care să indicați cele doua figuri geometrice ce sunt definite de punctele *(x0, y0)* si *(x1, y1)*. Astfel, pentru un cerc, *figura=1*, coordonatele *(x0, y0)* și *(x1, y1)* vor reprezenta două puncte complementare pe cerc (diametrul). Dacă este triunghi dreptunghic, punctele definesc ipotenuza iar *figura* va fi =2; Celelalte laturi ale triunghiului se vor determina pornind de la cele doua puncte. Coordonatele punctelor si apoi selecția figurii geometrice se va realiza introducând de la tastatura parametrii.
7. La un chioșc se vând ziare, reviste și carți. Fiecare publicație are un nume, o editură, un număr de pagini, un număr de exemplare per publicație și un preț fără TVA. Scrieți clasa care modelează publicațiile. Adăugați un membru static *valoare_tva* (procent) și o metodă statică pentru modificarea valorii TVA-ului. Să se calculeze suma totală cu TVA pe fiecare tip de publicație (ziare, reviste și carți) și prețul mediu pe pagina la fiecare publicație în parte. Modificați TVA-ul și refaceți calculele. Afișați editurile ordonate în funcție de încasări.
8. Considerati clasa *Fractie* care are doua atribute intregi *private a* si *b* pentru numarator si numitor, doua metode de tip *set( )* respectiv *get( )* pentru fiecare din atributele clasei. Declarati o functie *friend simplifica( )* care are ca si parametru un obiect al clasei, returnand un alt obiect simplificat. Considerați o variabila *private* statica întreagă *icount*, care va fi initializata cu *0* si incrementata in cadrul constructorilor din clasa. Definiti un constructor explicit fara parametri care initializeaza *a* cu 0 si *b* cu 1, si un constructor explicit cu doi parametri care va putea fi apelat daca se verifica posibilitatea definirii unei fractii (*b*!=0). Definiti un destructor explicit care afiseaza si decrementeaza contorul *icount*. Definti o functie *friend _f_aduna_fractie(...)* care are ca si parametri doua obiecte de tip *Fractie* si returneaza suma obiectelor in alt obiect *Fractie*. Analog definiti functii *friend* 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 functiei *simplifica( )*. Printr-o metoda accesor, afisati contorul *icount*. Efectuati operatiile implementate prin functiile *friend* ale clasei, initializand alte 4 obiecte cu rezultatele obtinute. Afisati rezultatele si contorul dupa ultima operatie folosind o metoda accesor adecvata.
9. Considerați problema referitoare la gestiunea CNP-ului dintr-un laborator anterior, la care sa validați complet CNP-ul (zi corecta funcție de luna corecta si an corect, inclusiv anii bisecți), in care sa introduceți cu confirmare mai multe date de tip *Person,* afișând la final câte obiecte au data introdusă corect. Contorizati folosind atribute statice private.

Homework:

1. Implement a C++ application that defines the class called *CurrentHour* with *hour, minute, second* as *private* attributes. The class has *public* setter/getter methods for each attribute. Add a *friend* function that copies the content of a *CurrentHour* object used as parameter into another instance of the class that will be returned by the function, hour being modified to Greenwich Mean Time. Use the computer local current time.
2. Write a C++ application in which the class *Calculator* has a private attribute called *RAM_memory* (int) and a *friend* function named *service_technician( )* that can modify the attribute's value. The *friend* function will be member in the class *Motherboard*, that encapsulates the *processor_type* variable (string of characters). Write the code that allows the modifying of the *processor_type*'s value and the *RAM_memory* from the friend function.
3. Define a class called *Repository* that has 2 integer private variables. The class contains an empty constructor and another one with 2 parameters. An accesor method that displays the variables values is also included in the class. Write another class called *Mathematics* which is friend to the first one. This class contains the implementation of the elementary arithmetical operations (+, -, *, /) applied to the values stored in the first class. Each arithmetical method receives as parameter an object instantiated from the first class.
4. Write a C++ application that stores inside a class a public static integer variable called *var_static*. The variable is incremented each time the class's constructor is called. After instantiating several objects, display their number using the value of the static variable.
5. Implement the 4-th problem by changing the static variable's access modifier to *private*. Define a method that returns the counter's value.
6. Write a C++ application that defines a class named *Poin*t with the private attributes *x* and *y*. Implement a *friend* function that calculate the area and perimeter of different shapes objects defined by two *Point*

parameters *P0(x0, y0)* and *P1(x1, y1)*. Introduce another parameter named *shape* that controls the way the two points are used. If we consider a circle, the points delimit the circle's diameter (*shape=1*). If we are dealing with a right triangle (*shape=2*), the points determine the hypotenuse, and the other sides are determined considering the 2 initial points. The point's coordinates and the shape selection will be realized using parameters introduced from KB.

7. A kiosk sells newspapers, magazines and books. Each publication has a name, an editorial house, a number of pages, the number of copies and a price (no VAT). Write the class that models the publications. Introduce a static member named *VAT_value* (percentage) and a static method that modifies the value of this variable. Determine the total income and the average price per page for each publication type. Modify the VAT and redo the calculations. Order the printing houses by the total income and display the result.
8. Consider the *Fraction* class that has two *private* integer attributes *a* and *b* for the denominator and nominator. Use two setter and getter methods for all the class's attributes. Declare a *friend* function named *simplify( )* that receives as parameter a *Fraction* object and returns the corresponding simplified object. Consider a *private* static integer variable *icount* that will be initialized with 0 and incremented in the class's constructors. Define an explicit constructor without parameters that initializes *a* with 0 and *b* with 1 and another explicit constructor with two integer parameters. For this constructor is verified if b!=0 before to be called. Define an explicit destructor that displays and decrements the value of *icount.* Define a *friend* function *f_add_fraction(...)* that returns an object reflecting the sum of the objects received as parameters. Implement similar functions for fractions subtraction, multiplication and division. Instantiate two *Fraction* objects and read the appropriate data from the keyboard. Display the initial attributes and the ones obtained after simplifying. Call the implemented *friend* functions and store the results into another different four objects. Display the results and the objects counter using the corresponding accesor methods.
9. Consider the CNP management problem from a previous laboratory where you fully validate the CNP (correct day based on the correct month and the correct year, including the leap years) in which you enter more than one data of *Person* type in a loop with confirmation, showing at the end how many objects have the correct data entered. Use private static attributes as counters.