Laborator VI

MODIFICATORI

1. Scopul lucrării

Lucrarea de laborator are ca scop aprofundarea noțiunilor legate de modificatorii `const` și `static` prin analiza unui exemplu simplu și propunerea unei teme de casă.

2. Modificatorii `const` și `static`

Modificatorul `const`

Modificatorul `const` determină apariția unui nou tip de date cu proprietatea că obiectul de tipul respectiv este constant în sensul că nu poate fi modificat. Modificatorul `const` se adresează compilatorului, acesta testând tipul respectiv fără a genera un cod special. Compilatorul se asigură că instrucțiunile nu modifică un obiect declarat `const`.

Obiectele declarate `const` pot fi numai inițializate în momentul declarării.

Obiectele `const` sunt utile pentru:

- a defini constante cu nume cu avantajul declarării explicite a tipului;
- ca parametri formali pentru funcții sau operatori în contextul în care pot apare ca parametri reali obiecte anonime;
- ca tip de date returnat.

Modificatorul `static`

Modificatorul `static` determină zona de date în care este memorat obiectul declarat *static*, impunând domeniul *static* de existență, precum și domeniul de vizibilitate.

Un obiect declarat *static* are *domeniul de existență* întregul program, *domeniul de vizibilitate*, unitatea în care este declarat (clasă, funcție, fișier).

Indiferent de locul în care apare declarația obiectului *static*, procesul de inițializare implicat de aceasta are loc o singură dată, în momentul lansării în execuție a programului.

Membrii *statici* ai unei clase sunt *membri comuni* tuturor obiectelor declarate de tipul clasă respectivă. Ei reprezintă zone de memorie comună obiectelor clasei (eventual asociată cu un tip).

Un obiect *static* ca membru al unei clase poate fi accesat prin intermediul oricărui obiect de tipul clasă, sau, de dorit, prin intermediul clasei urmată de operatorul rezoluție.

O funcție membru declarată *static nu* posedă pointerul *this* și poate fi accesată de oricare obiect de tipul clasă, sau, de dorit, prin intermediul clasei urmată de operatorul rezoluție.

3. Program

Următorul program are ca scop experimentarea conceptului `static` și `const`. Construiți un proiect care conține fișierul *static.cpp* și urmăriți modul în care este executat programul.

```
/*********************************************
static.cpp
*********************************************/

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

using namespace std;
class Clasa
{
private:
        static int      NumarObiecte; //Contor pentru numar de obiecte
        int               ValInt; //Valoare intreaga
```

```
public:
        static void PrintNumarObiecte(void);
        Clasa(void);
Clasa(Clasa const &);
Clasa(int);
~Clasa();
Clasa & operator =(Clasa const &); //Supraincarcarea operatorului de atribuire ...
//Supraincarcarea operatorului + ... .
//Operatorul membru va fi mostenit in clasele derivate.
Clasa operator +(Clasa &);
//Supraincarcarea operatorului - ...
//Operatorul "prieten" nu va fi mostenit in clasele derivate.
friend Clasa operator -(Clasa &, Clasa &);
//Supraincarcarea operatorului de insertie.
friend ostream & operator <<(ostream &, Clasa &);
//Supraincarcarea operatorului de extractie.
friend istream & operator >>(istream &, Clasa &);
};

int Clasa::NumarObiecte = 0; //initializarea obiectului membru static ...

Clasa::Clasa(void)
{
    NumarObiecte++; //Este creeat un nou obiect ...
    ValInt = 0;
    PrintNumarObiecte();
}

Clasa::Clasa(Clasa const &Obiect)
{
    NumarObiecte++; //Este creeat un nou obiect ...
    ValInt = Obiect.ValInt;
    PrintNumarObiecte();
}

Clasa::Clasa(int Val)
{
    NumarObiecte++; //Este creeat un nou obiect ...
    ValInt = Val;
    PrintNumarObiecte();
}

Clasa::~Clasa()
{
    NumarObiecte--; //Este distrus un obiect
    PrintNumarObiecte();
}

void Clasa::PrintNumarObiecte(void)
{
    cout << "Numarul curent de obiecte este:" << NumarObiecte << endl;
}

Clasa & Clasa::operator =(Clasa const &Obiect)
{
    ValInt = Obiect.ValInt;
    return *this;
}

Clasa Clasa::operator +(Clasa &Obiect)
{
    Clasa Rezultat;
    Rezultat.ValInt = ValInt + Obiect.ValInt;
    return Rezultat;
}

Clasa operator -(Clasa &Obiect1, Clasa &Obiect2)
{
    Clasa Rezultat;
    Rezultat = Obiect1.ValInt - Obiect2.ValInt;
    return Rezultat;
```

```
}

ostream & operator <<(ostream &Out, Clasa &Obiect)
{
    Out << Obiect.ValInt;
    return Out;
}

istream & operator >>(istream &In, Clasa &Obiect)
{
    In >> Obiect.ValInt;
    return In;
}


void main(void)
{
    Clasa Obiect1;
    cout << "Introduceti o valoare intreaga: ";
    cin >> Obiect1;
    Clasa Obiect2;
    cout << "Introduceti o valoare intreaga: ";
    cin >> Obiect2;
    Clasa Obiect3 = Obiect1; //Este apelat constructorul de copiere ...
    Clasa *PtrObiect; //Pointer la obiect de tipul Clasa ...
    //Se aloca memorie dinamica pentru un obiect.
    //Pentru initializare se foloseste constructorul de copiere.
    PtrObiect = new Clasa(Obiect3);//Se aloca memorie dinamica pentru un obiect
    if(!PtrObiect)
    {
        cout << "Eroare la alocarea dinamica de memorie !!!" << endl;
        return;
    }
     *PtrObiect = Obiect1 + Obiect2;
     cout << Obiect1<< " + " << Obiect2 << " = " << *PtrObiect;
     cout<<endl<<"Apasati o tasta pentru a continua..."<<endl;
     _getch();
     delete PtrObiect;
     Obiect3 = Obiect1 - Obiect2;
     cout << Obiect1<< " - " << Obiect2 << " = " << Obiect3;
     cout<<endl<<"Apasati o tasta pentru a continua..."<<endl;
     _getch();
     Clasa::PrintNumarObiecte(); //Apelul functiei declarata static ...
     //Se aloca memorie dinamica pentru 10 obiecte de tipul clasa ...
     PtrObiect = new Clasa[10];
     if(!PtrObiect)
     {
         cout << "Eroare la alocarea memoriei dinamice !!!" << endl;
         return;
     }
     delete []PtrObiect;
     //delete PtrClasa; //Varianta gresita. De ce ?

}
```

4. Temă

Studiați cursurile, exemplele simple de la curs, și urmăriți programul propus realizând în scris o descriere a funcționării acestuia. Comparați analiza de pe hârtie cu comportamentul programului observat ca rezultat al rulării acestuia.

Modificați clasa *MatrixInt2* din lucrarea V pentru a-i adăuga și funcționalitatea pe care o oferă membrii statici ai clasei *Clasa* din exemplul anterior.