Laborator 7:Membri Statici. Membri Const

Membri Statici

In contextul paradigmei de programare orientate obiect avem posibilitatea sa ne definim membri ai unei clase ca fiind statici. In sectiunea imediat urmatoare vom explica si exemplifica acest lucru.

Pentru a putea defini un membru al unei clase ca fiind static (fie variabila membru sau functie membru) trebuie precedata declararea acelui membru de cuvantul **static**. Cand un membru al unei clase este definit ca fiind static, atunci compilatorul retine faptul ca exista o singura copie a acelei variabile indiferent de cate instante ale clasei respective exista. De asemenea, membrul static al clasei respective, poate fi accesat indiferent de obiectul apelant.

Practic, spre deosebire de obiectele obisnuite din cadrul clasei respective, variabile membru statice ale clasei respective sunt instantiate o singura data si initializate cu valoarea 0, lucrul acesta executandu-se la definirea clasei, inainte ca sa se instantieze orice obiect.

Variabilele membru statice nu sunt definite(nu se aloca spatial necesar din memorie) in momentul declararii lor in cadrul clasei. Pentru a putea fi definite, sunt redeclarate in exteriorul clasei respective, folosind operatorul de rezolutie ca in exemplul de mai jos.

Exemplu:

```
#include <iostream>
using namespace std;
class MyShared {
      static int a;
      int b;
public:
      void set(int i, int j) {a=i; b=j;}
      void show();
} ;
int MyShared::a; // definirea lui a
void MyShared::show()
{
      cout << "This is static a: " << a;
      cout << "\nThis is non-static b: " << b;
      cout << "\n";
}
int main()
{
      MyShared x, y;
      x.set(1, 1); // a primeste valoarea 1
      x.show();
      y.set(2, 2); // a isi schimba valoarea in 2
      y.show();
```

```
	x.show();
	system("PAUSE");
	return 0;
}
```

Dat fiind faptul ca variabilele membru statice sunt declarate si initializate indiferent daca sunt instantiate sau nu obiecte, putem accesa variabilele respective, fara a ne folosi de vreun obiect ci prin intermediul operatorului de rezolutie cu trimitere la clasa respectiva.

Exemplu:

```
#include <iostream>
using namespace std;
class MyShared {
	int b;
public:
	static int a;
	void set(int i, int j) {a=i; b=j;}
	void show();
} ;
int MyShared::a; // definirea lui a
void MyShared::show()
{
	cout << "This is static a: " << a;
	cout << "\nThis is non-static b: " << b;
	cout << "\n";
}
int main()
{
	MyShared::a = 20;
	MyShared x;
	x.show();
	system("PAUSE");
	return 0;
}
```

Conceptul de membru static, poate fi extins si asupra functiilor membru din cadrul unei clase. Ele sunt definite in aceeasi maniera ca celelalte functii cu exceptia faptului ca antetul functiei este precedat de cuvantul **static**. Totusi sunt cateva restrictii care sunt aplicate metodelor statice. In cadrul acestor functii se poate opera numai asupra variabilelor membru statice. In cadrul acestor functii nu exista pointerul **this**, deoarece aceste metode se apeleaza indiferent daca exista sau nu un obiect apelant. O metoda statica nu poate avea natura virtuala si nu pot exista 2 versiuni ale aceleasi functii, dintre care una sa fie statica si una non – statica .

Practic, metodele statice sunt utilizate indeosebi pentru a accesa variabile membru statice ce au fost declarate ca fiind private(principiul incapsularii).

Exemplu:

```
#include <iostream>
using namespace std;
class MyShared {
```

```
	int b;
	static int a;
public:
	static void setStatic(int i){a = i;}
	static void showStatic();
	void setB(int j) {b = j;}
	void showB();
} ;
int MyShared::a; // definirea lui a
void MyShared::showStatic()
{
	cout << "This is static a: " << a;
	cout << "\n";
}
void MyShared::showB()
{
	cout << "\nThis is non-static b: " << b;
	cout << "\n";
}
int main()
{
	MyShared::setStatic(20);
	MyShared x;
	x.showStatic();
	system("PAUSE");
	return 0;
}
```

Membri Const

Puteti sa declarati variabile membru constante in cadrul unei clase. Acestea vor avea aceeasi valoarea in decursul existentei unui obiect si nu pot fi modificate. Pentru a putea declara astfel de obiecte, definitia lor trebuie precedata de cuvantul **const**. De asemeni ele pot fi initializate in cadrul constructorului de copiere prin intermediul listei de initializare. Pentru detalii, aveti exemplul de mai jos:

Exemplu:

```
#include <iostream>
using namespace std;
class MyClass
{
	const int i;
	int j;
public :
	MyClass() : i(0)
	{
		j = 20;
	}
	int get()
	{
		return i;
	}
```

```
};
int main()
{
	MyClass c;
	cout<<c.get();
	system("PAUSE");
	return 0;
}
```

In cadrul unei clase, aveti de asemenea posibilitatea sa declarati functiile membru ca fiind constante. Pentru a putea defini o functie ca fiind constanta, trebuie sa specificati cuvantul **const** dupa antetul functiei respective. Acest lucru impune ca in cadrul acelei functii sa nu puteti modifica valoarea din cadrul pointerului **this**. Practic, nu aveti posibilitatea sa modificati variabilele membru ale obiectului apelant.

In situatia in care aveti declarate obiecte constant din cadrul unei clase, prin intermediul acelor obiecte puteti apela numai functiile membru constante ale clasei respective, in schimb, daca aveti obiecte non – constante, puteti apela si metode constante si non – constante.

Exemplu:

```
#include <iostream>
using namespace std;
class MyClass
{
	int a;
public :
	MyClass()
	{
		a = 0;
	}
	void setA(int v)
	{
		a = v;
	}
	int getA() const
	{
		return a;
	}
};
int main()
{
	MyClass a;
	a.setA(201);
	cout<<a.getA()<<endl;
	system("PAUSE");
	return 0;
}
```

Exercitii

1. Definiti clasa MyClass, in cadrul careia sa aveti 2 variabile membru private de tipul int, contor si data.
   Definiti constructor de initializare in cadrul careia setati valoarea varibilei data la 0, si de asemeni, metode set si get ce respecta principiul incapsularii asupra variabilei membru data. Metoda get sa fie definita de asa natura astfel incat sa nu se poata modifica valoarea variabilei data din cadrul MyClass.
   Variabila contor, trebuie sa actioneze ca o variabila de monitorizare asupra obiectelor instantiate din cadrul clasei MyClass, si anume : ori de cate ori instantiati un obiect, incrementati variabila contor si ori de cate ori distrugeti un obiect, decrementati variabila contor. Creati o metoda statica, showContor(), prin intermediul careia se afiseaza valoarea variabilei contor.
   Creati o functie, MyFunc, in cadrul careia definiti un vector de elemente de tipul MyClass pentru care setati valori variabilei data. Apelati metoda showContor() inainte de crearea vectorului si dupa crearea vectorului pentru a putea valida daca intr-adevar aveti numarul dorit de obiecte MyClass, conform dimensiunii vectorului.