Lab 4 – P2

Cuprins

1. Constructorul de copiere

Constructorul de copiere este un constructor special, folosit pentru a crea un nou obiect, copie a unui obiect existent. Acest constructor are un singur argument – o referință către obiectul ce va fi copiat. Dacă în clasă nu există constructorul de copiere scris explicit, compilatorul generează implicit un constructor de copiere. Forma generală a constructorului de copiere este următoarea:

```
MyClass(const MyClass& other );
```

Constructorul de copiere implicit copiază bit cu bit fiecare membru al obiectului parametru în membrul corespunzător al obiectului de inițializat. De exemplu, pentru clasa `Data` din exemplele de la curs:

```
class Data
{
	unsigned char zi, luna;
	int an;
public:
	void afisare();
	Data(unsigned char z, unsigned char l, int a);
	Data();
	Data(Data &);
};
```

constructorul de copiere poate avea următoarea formă:

```
Data::Data(Data & d)
{
	zi=d.zi;
	luna=d.luna;
	an=d.an;
}
```

Constructorul de copiere are un rol special în C++. El este apelat automat în următoarele situații:

1. La declararea unui obiect, inițializat cu un alt obiect. Exemplu:

```
Data d1(1,2,1900);
Data d2(d3)// constr. de copiere este folosit pentru a crea obiectul d2

Data d3 = d3;  // constr. de copiere este apelat
z = x;       // Operatorul de atribuire (=), nu se apeleaza constructori
```

2. La transferul parametrilor unei funcții prin valoare.
3. O funcție returnează un obiect.

Să urmărim programul de mai jos:

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

class Complex {
	int re, im;
public:
	Complex(){}
	Complex(int r, int i) {
		re = r;
		im = i;
	}

	Complex aduna(Complex c2) {
		Complex rez;
		rez.re = re+c2.re;
		rez.im = im+c2.im;
		return rez;
	}

	void afisare() {
		cout<<"("<<re<<","<<im<<")"<<endl;
	}
};

int main() {
	Complex c1(5,3);
	Complex c2(2,-3);
	Complex c3;
	c3 = c1.aduna(c2);
	c1.afisare();
	c2.afisare();
	c3.afisare();
	return 0;
}
```

La apelul funcției `aduna()`, constructorul de copiere este apelat de 2 ori. O dată la transmiterea parametrului `c2`, și a doua oară la returnarea rezultatului `rez`.

În total, există 3 membri generați de compilator în mod implicit:

1. Constructorul implicit, în cazul în care nu este definit nici un alt constructor;
2. Constructorul de copiere implicit, in cazul in care nu este scris explicit un constructor de copiere;
3. Operatorul de atribuire.

Dacă programatorul are nevoie ca oricare dintre acești trei membri să fie definit altfel decât în modul implicit, îl poate defini în forma dorită.

Un caz special este atunci când clasa conține câmpuri de tip pointer. În acest caz utilizatorul trebuie să definească constructorul de copiere în mod explicit. Constructorul de copiere trebuie sa aloce memoria necesară pentru câmpuri de tip pointer și va inițializa memoria alocată.

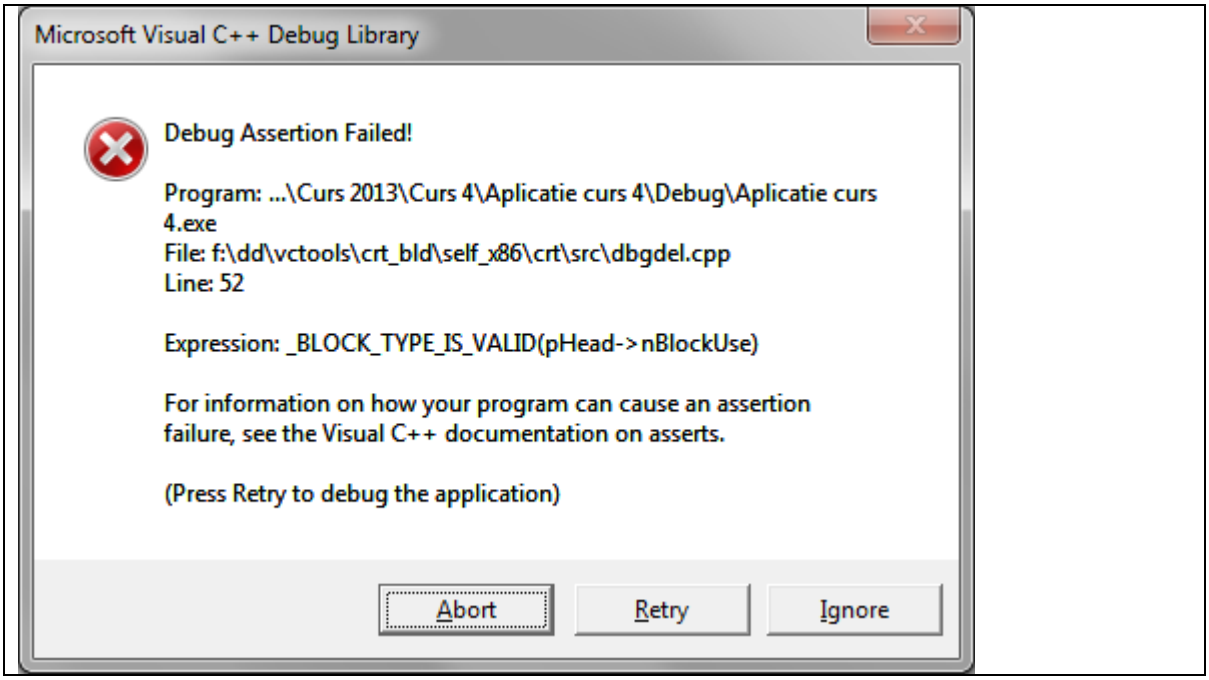
Consideram exemplul clasei `Persoana`:

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

class Persoana
{
```

```
	char * nume;
	int varsta;
public:
	Persoana();
	Persoana (char * n, int v);

	~Persoana ();
	void afiseaza();
	Persoana schimbaVarsta(int);
};

Persoana::Persoana()
{
	nume = new char [1] ();
	varsta = 0;
}
Persoana::Persoana(char * n, int v)
{
	nume = new char [strlen (n) + 1];
	strcpy_s (nume, strlen (n) + 1, n);
	varsta = v;
}

Persoana::~Persoana ()
{
	if (nume)
		delete [] nume;
}
void Persoana::afiseaza()
{
	cout << "Nume=" << nume << " varsta=" << varsta << endl;
}
Persoana Persoana::schimbaVarsta(int n)
{
	Persoana temp (nume, varsta);
	varsta = n;
	return temp;
}

int main ()
{
	Persoana p1 ("Pavel",22);
	p1.afiseaza();
	Persoana p2 ("Ana",23);
	Persoana p3 = p2.schimbaVarsta(25);
	p2.afiseaza();
	p3.afiseaza();
	return 0;
}
```

Ieșire:

```
Nume=Pavel varsta=22
Nume=Ana varsta=25
Nume=ŢŢŢŢŢŢŢŢEtł2ŢŢ varsta=23
```



Se afișează mesajul de eroare pentru că în funcția schimbaVarsta, la apelul instrucțiunii `return temp;` compilatorul apelează constructorul de copiere implicit pentru a crea o copie a obiectului temp și pentru a o plasa pe stivă, pentru a o putea returna. Acest constructor, după cum am menționat anterior, copiază bit cu bit obiectul sursa în obiectul destinație. Din cauză că în clasa `Persoana` există membrul `Nume` de tip pointer la char, la copierea bit cu bit a obiectului temp se va face o copie a cărui membru `Nume` va indica spre aceeași adresă de memorie ca și membrul `Nume` al obiectului `temp`. În momentul când obiectul `temp` nu mai este folosit (după instrucțiunea return), se apelează destructorul pentru acest obiect.

Destructorul va dealoca zona de memorie spre care indică membrul `Nume` al obiectului `temp`. Deoarece și membrul `Nume` aparținând copiei obiectului de pe stivă indică spre aceeași zonă de memorie, după ștergerea acesteia va indica spre o zona de memorie nealocată. În momentul când se va încerca afișarea obiectului returnat, va apărea mesajul de eroare.

Pentru a putea remedia aceasta problemă, trebuie adăugat clasei un constructor de copiere care să trateze în mod explicit problema alocării și copierii membrului Nume al obiectului destinație. Pentru acestea, trebuie adăugat clasei constructorul de copiere:

```
Persoana (Persoana & p);
```

Cu definiția:

```
Persoana::Persoana (Persoana & p)
{
	nume = new char [strlen (p.nume) + 1];
	strcpy_s (nume, strlen (p.nume) + 1, p.nume);
	varsta = p.varsta;
}
```

După adăugarea acestui constructor, programul va funcționa în modul așteptat, fără a mai furniza nicio eroare.

2. Liste de initializare

Prin utilizarea listelor de inițializare se apelează numai constructorii de copiere ai claselor membrilor pe care îi inițializează. Utilizarea constructorului de copiere este mai eficace din punctul de vedere al timpului

de execuție în comparație cu apelarea constructorului fără argumente (sau al celui implicit) urmată de o atribuire.

Listele de inițializare sunt folosite în principal pentru a controla ce constructori se apelează în cadrul claselor derivate.

Consideram următorul constructor al clasei Data:

```
Data::Data(unsigned char z, unsigned char l, int a)
{
	cout << "S-a apelat constr. cu lista de argumente" << endl;
	zi = z;
	luna = l;
	an = a;
}
```

Pentru a putea reduce timpii de execuție necesari apelării operatorului de atribuire implicit și înlocuirea acestora cu timpii necesari apelării constructorilor de copiere, se folosesc listele de inițializare:

```
Data::Data(unsigned char z, unsigned char l, int a) : zi(z), luna(l), an(a)
{
	cout << "S-a apelat constr. cu lista de argumente" << endl;
}
```

3. Membri statici ai claselor

Folosind cuvântul cheie **`static`** unii membri ai clasei pot fi declarați de tip static.

Membrii de tip dată declarați statici au următoarele proprietăți:

- există într-un singur exemplar indiferent de numărul de instanțieri ale clasei; ca o consecință, dimensiunea datelor membre statice nu participă la dimensiunea nici unei instanțieri a clasei;
- pot fi referiți prin numele clasei urmat de operatorul de rezoluție `::` și de numele membrului dată static;

```
class A
{
public:
	static int x;
};
int A::x = 10;
int main ()
{
	A a1,a2;
	cout << "a1.x=" << a1.x << endl;
	a1.x = 99;
	cout << "a2.x=" << a2.x << endl;
	cout << "A::x=" << A::x << endl;
}
```

Ieșire

```
a1.x=10
a2.x=99
A::x=99
```

Funcțiile membre statice:

- Pot fi apelate fără a folosi un obiect instanțiat de tipul clasei
- Nu au acces la pointerul this
- Se comportă precum o funcție globală, dar au acces către membrii privați ai clasei din are fac parte
- Nu pot accesa membrii non-statici ai clasei decât prin intermediul unui obiect

Adăugam la clasa A de mai sus următoarele:

```
public:
	static int x;
	static void f1 ();
```

```
void A::f1()
{
	x++;
}
```

In main adăugam:

```
a1.f1();
cout << "A::x=" << A::x << endl;
A::f1();
cout << "A::x=" << A::x << endl;
```

Ieșire

```
A::x=100
A::x=101
```

4. Pointerul `this`

Pointerul `this` este o variabilă predefinită în C++ accesibilă în corpul oricărei metode non-statice din cadrul unei clase. Valoarea pointerului este dată de adresa obiectului pentru care s-a apelat o anume metodă non-statică din clasă.

Este folosit:

- Pentru a înlătura ambiguitățile dintre un parametru al unei funcției și o variabilă membră
- În cazurile când este necesar un pointer către obiectul pentru care s-a apelat o anumită metodă

Considerăm o clasă X și o instanțiere x a acestei clase. Există două posibilități de utilizare a pointerului this. Acestea sunt exemplificate prin `func1()` și `func2()` în exemplul următor:

- când se face apelul metodei `x.func1()`, lui this i se dă valoarea adresei lui x (&x) și poate fi folosit în corpul funcției `func1`, după care se poate returna ca și pointer.
- când se face apelul metodei `x.func2()`,se returnează conținutul pointerului `this`.

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

class X
{
public:
	X* func1()
	{
		cout<<"Test1 pentru this."<<endl;
		return this;
	}

	X& func2()
	{
		cout<<"Test2 pentru this."<<endl;
		return *this;
	}
};
```

```
int main()
{
	X  x;
	X *x1 = x.func1();
	X& x2 = x.func2();
	x1->func1();
	x2.func2();
	return 0;
}
```

Ieșire

```
Test1 pentru this.
Test2 pentru this.
Test1 pentru this.
Test2 pentru this
```

Orice metodă non-statică a unei clase are ca prim parametru variabila `this` transmisă implicit. Altfel spus, compilatorul C++ convertește apelul funcției non-statice apelate și pune ca prim parametru pointerul `this`. Acest lucru este exemplificat in exemplul de mai jos:

Funcția:

```
void Data::afisare()
{
	cout << "Data este" << (int)zi << "-" << (int)luna
<< "-" << an << endl;
}
```

Este transformata de compilator in functia:

```
void Data::afisare(Data * const this)
{
	cout << "Data este" << (int)this->zi << "-" <<
(int)this->luna << "-" << this->an << endl;
}
```

5. Exerciţii

1. Sa se implementeze o clasa Autoturism cu următorii membri:
   a. culoare de tip char *
   b. an_fabricatie de tip unsigned int;

Pentru aceasta clasa, sa se implementeze constructorii necesari (utilizând liste de inițializare acolo unde este posibil) care sa permită următoarele apeluri în funcția main:

```
Autoturism a1;
Autoturism a2(„Rosie”,1999);
Autoturism a3=a2;
Autoturism * a4 = &a3;
```

De asemenea, să se implementeze următoarele:

- o variabila statică `nr_autoturisme` care să retină câte obiecte de tip Autoturism au fost create.
- O metodă `afisare` care afișează datele corespunzătoare unui obiect de tip Autoturism.
- Un destructor pentru dealocarea zonelor de memorie alocate dinamic.
- O metodă care să permită schimbarea culorii unui autoturism.

- O metodă care să compare două autoturisme din punctul de vedere al anului fabricației.

2. Creați un vector care sa conțină un număr n citit de la tastatură de obiecte de tip Autoturism. Vectorul va fi salvat cu ajutorul unui pointer către un obiect Autoturism.
   Se cer:
   - Să se citească vectorul de autoturisme de la tastatură. (Pentru fiecare autoturism se va citi culoarea și anul fabricației).
   - Să se afișeze toate datele obiectelor de tip Autoturism citite.
   - Să se sorteze obiectele de tip Autoturism în funcție de anul fabricației.