Universitatea Tehnică „Gheorghe Asachi" din Iași
Facultatea de Automatică și Calculatoare

PROGRAMARE II

Curs 5

Supraîncărcarea operatorilor. Membri statici. Functii inline. Compoziția și agregarea.

Operatori impliciți

- Compilatorul are definite variante implicite ale lui *operator*= (atribuire) și *operator*& (adresa lui)
- Pentru folosirea acestor operatori în scopul lor clasic nu trebuiesc supraîncărcați operatorii = și &

```
Data *d4 = &d2;
d4->afisare();
```

```
Data calendaristica este1-2-2008
```

- Se pot supraîncărca acești operatori dar se recomandă să nu se schimbe semnificația lor

Supraîncărcarea operatorului ++

- Forma pre-fixată
  - ++a
- Forma post-fixată
  - a++
- Funcțiile trebuie să asigure:
  - Modificarea obiectului pentru care s-a făcut apelarea
  - Returnarea unui obiect cu membrii modificați corespunzător, pentru a asigura funcționarea apelurilor de genul
    - b = a++

obiectul apelat

tipul de return

Forma pre-fixată

- În clasă se adaugă:

```
public:
        Data operator++();
```

- Funcția de supraîncărcare:

```
Data Data::operator++()
{
        zi++;
        //conditii pentru transporturi
        return *this;
}
```

Forma post-fixată a operatorului ++

- În clasă se adaugă:

```
public:
        Data operator++(int);
```

- Funcția de supraîncărcare:

```
Data Data::operator++(int)
{
        Data temp = *this;
        //conditii pentru transporturi
        zi++;
        return temp;
}
```

Operatorii ++ pentru numere intregi

```
int aa = 1, xx = 1;
int bb =  aa++;
int yy = ++xx;

cout << bb << endl;
cout << xx << endl;

int cc = ++bb + aa++;

cout << cc << endl;
```

- 1
- 2
- 4

Forma pre-fixată a operatorului ++

```
cout<<"supraincarcarea operatorului ++
          pre-fixat"<<endl;
Data d5 (3,4,2000);
++d5;
d5.afisare();
```

```
supraincarcarea operatorului ++ pre-fixat
Data calendaristica este 4-4-2000
```

```
cout<<"supraincarcarea operatorului ++ post-
fixat"<<endl;
Data d6 = ++d1;
d6.afisare();

cout<<"d7:"<<endl;
Data d7 = d2++; // Data d2 (1,2,2008);
d7.afisare();
d2.afisare();
```

```
supraincarcarea operatorului ++ post-fixat
Data calendaristica este 4-4-2000
d7:
Data calendaristica este 1-2-2008
Data calendaristica este 2-2-2008
```

Cuvantul cheie static - C

- Variabilele declarate static
  - Nu-şi pierd valoarea intre apelurile funcţiei in care au fost declarate
  - Variabilele statice globale nu sunt vizibile in afara fişierului in care au fost declarate
- Funcţiile declarate static
  - Nu sunt vizibile decat din fişierul de unde au fost declarate

Cuvantul cheie static – C++

- Variabile statice membre
- Funcţii statice membre

Cuvantul cheie static – C++

- Variabile statice membre
  - Toate obiectele de un anumit tip împart accesul la variabilele statice ale clasei
  - Pot fi declarate în clasă, nu şi definite
  - Trebuie definite (iniţializate) în afara clasei în scop global
  - Pentru că este o variabilă unică pentru toate obiectele de tipul clasei, **poate fi accesată direct ca un membru al clasei**, fără a avea un obiect instanţiat de tipul clasei din care fac parte

Variabile membre statice

```
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;
}
```

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

Funcții statice membre

- 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

Funcții statice membre

- In clasă se adaugă

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

- Se adaugă funcția

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

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

- In main, se adaugă:

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

Funcții statice membre

```
class A
{
private:
       int y;
public:
       static int x;
       static void f1 ();
};
```

```
void A::f1()
{
       x++;
       y=0;
}
```

```
void A::f1()
{
       x++;
       A tempClass;
       tempClass.y=0;
}
```

- error C2597: illegal reference to non-static member 'A::y'

Funcții membre constante

- Fie clasa:

```
class Data
{
	unsigned char zi, luna;
	int an;
public:
	Data():zi(0),luna(0),an(0){};
	int getZi() const
		{ return zi; }
	int getLuna() const
		{ return luna; }
	int getAn() const;
		{ return an; }

	int addAn() const;
};
```

Funcții membre constante

- Prin prezența cuvântului cheie `const` se specifică compilatorului că funcțiile respective nu pot modifica conținutul obiectului *Data*.

```
int Data::addAn() const
{
	an++;
	return an;
}
```

```
error C2166: l-value specifies const object
```

- O funcție membră constantă definită în afara clasei necesită sufixul `const`.

Funcții membre constante

- O funcție membră poate fi apelată atât de obiecte constante cât și de obiecte non-constante.
- Se modifică în clasă: `int addAn();`

```
int Data::addAn()
{
	an++;
	return an;
}
void f(Data d, const Data cd)
{
	int i = d.getAn();		//ok
	d.addAn();			//ok
	int j = cd.getAn();		//ok
	cd.addAn();
	//error C2662: 'Data::addAn' : cannot convert 'this'
	//pointer from 'const Data' to 'Data &'
}
```

Funcții inline

- Funcțiile `inline` sunt funcții care forțează compilatorul să le expandeze inline
- Cu alte cuvinte, compilatorul va insera tot corpul funcției acolo unde este apelată
- Se renunță astfel la instrucțiunile de salt, care implică un timp mare de execuție
- Crește dimensiunea programului
- Unele compilatoare ignoră funcțiile inline definite de utilizator și decid singure ce funcții vor fi inline și care nu
- Funcțiile de dimensiuni mici și care se apelează foarte des trebuie să fie `inline`
- Majoritatea compilatoarelor nu vor face `inline` o funcție recursivă

Funcții inline C++

- Funcțiile declarate și definite în clase sunt implicit funcții inline
- Se preferă folosirea cuvantului cheie `inline` la definirea funcției declarate în clasă
- Există diferite metode de a forța o anumită funcție să fie `inline` (`pragma inline_recursion(on)`, `__forceinline`, `etc`)

Compoziția și agregarea

- În lumea reală, obiectele mari sunt de obicei compuse din mai multe obiecte mici (o mașină are roți, motor, etc)
- Procesul de construire a obiectelor complexe din mai multe sub-obiecte poartă numele de compoziție
- C++ permite utilizarea claselor ca obiecte membre ale altor clase

Compoziția

- Compoziția
  - Toti membrii au aceeași durată de viață ca și clasa în care au fost incluși
  - Obiectul parinte "are" (deține) (*eng. "has a"*) obiectele de tipul claselor folosite

Compoziția

```
class Adresa
{
	char * strada;
	int nr;
	//...
};
```

```
class Persoana
{
	Adresa *m_adresa;
public:
	Persoana ()
	{
		m_adresa = new Adresa;
	}
	~Persoana()
	{
		delete m_adresa;
	}
};
```

- Adresa îi aparține persoanei
- Alt exemplu:
  - Relația casă - cameră

Agregarea

- Este un tip special de compoziție
- Obiectele membre pot avea durate de viață diferite față de clasa părinte
- Compoziția este un caz special de agregare

Agregarea

```
class Profesor
{
      char * nume;
      int vechime;
};
```

```
class Departament
{
      Profesor * profesor;
public:
      Departament(Profesor * prof)
            :profesor(prof){};
      ~Departament();
};
```

- Profesorii nu sunt detinuți de departamentul in care se află
- Alt exemplu:
  - Relația companie - angajat