Programmation orientée objet

Le constructeur de copie & opérateur =

Samuel Kadoury & Claudy Picard

Motivation

```
void f(Employee p) {}

int main() {
    Employee e1;//Constructeur par défaut
    Employee e2("Marc", 15000);//Constructeur par
    paramètres
    Employee e3(e2); //???
    Employee e4 = e2; //???
    f(e4); //???
    e3 = e1; //???
}
```

Construction d'un **nouvel objet** en utilisant **le constructeur de copie**

Copie d'un objet dans un autre **objet existant** en utilisant **l'opérateur d'affectation (=)**

Constructeur de copie

- Lorsqu'on fait une **copie** d'un objet, il faut créer un **nouvel objet** qui sera utilisé dans la fonction
- On utilisera donc un constructeur lors de la création de ce nouvel objet
- **Ce constructeur recevra comme paramètre un autre objet de la même classe**, soit celui qu'on doit copier

Constructeur de copie (suite)

- Si on ne définit pas ce constructeur de copie, C++ utilisera un constructeur de copie par défaut, qui copie tout simplement les attributs (“**shallow copy**”) . Ce constructeur est correcte lorsque l’objet englobant est un **agrégat** ou un **composite par valeur**.
- Par contre, lorsque l’objet englobant est un **composite par pointeurs**, une **copie en profondeur** est nécessaire. Il faut alors **définir un constructeur de copie** qui fera la copie en profondeur.

Constructeur de copie & Agrégation

```
class Company {
public:
   Company(const Company& c):
         employees_(c.employees_) ,
         president_(c.president_) {}

private:
   vector<shared_ptr<Employee>> employees_;
   Employee& president_;
};
```

Remarque: des fonctions de la même classe peuvent toujours accéder aux attributs privés des autres objets de cette classe!

Constructeur de copie – Composition par valeurs

```
class Company {
public:
   Company(const Company& c):
         president_(c.president_) {}
private:
   Employee president_;
};
```

Constructeur de copie – Composition par pointeurs

```
class Company {
public:
    Company(const Company& c) :
        employees_(c.employees_) {
    president_ = make_unique<Employee>(*c.president_);
}

private:
    vector<shared_ptr<Employee>> employees_;
    unique_ptr<Employee> president_;
};
```

Opérateur =

- Ce que nous venons de dire pour la copie d'un objet vaut aussi pour l'opérateur =:

```
int main() {
   Company c1;
   Company c2;

   c1 = c2;
}
```

Ici aussi une copie attribut par attribut sera effectuée, à moins qu'on ne redéfinisse l'opérateur =, ce qui, évidemment, doit être fait pour la classe Company, comme on a dû le faire pour le constructeur de copie.

Définition et d'implémentation de l'opérateur =

```
class Company {
public:
    Company& operator=(const Company& company) {
        if (this != &company) {
            employees_ = company.employees_;
            president_ = make_unique<Employee>(*company.president_);
        }
        return *this;
    }
private:
    vector<shared_ptr<Employee>> employees_;
    unique_ptr<Employee> president_;
};
```

Appel en cascade

Pour éviter l'auto-affectation

références

On retourne une référence à l'objet parce que l'opérateur = peut être appelé en cascade:
c1 = c2 = c3 (qui est équivalent à **c1 = (c2 = c3)**)

Résumé

- Lorsqu'on définit une classe en C++, il faut toujours penser à définir les items suivants si leur définition par défaut n'est pas adéquate:
  - Le constructeur par défaut
  - Le constructeur de copie
  - L'opérateur =
  - Le destructeur