Programmation orientée objet

Tuples

Motivation

- Souvent, on est face à une situation où l'on veut regrouper plusieurs éléments
- L'exemple le plus commun est lorsqu'une fonction doit retourner plusieurs valeurs
- On peut également vouloir affecter et comparer plusieurs valeurs simultanément
- Si les éléments sont tous du même type, on utilise un conteneur (vector par exemple)
- Mais si les éléments sont hétérogènes ?

Classe

- Une méthode est de regrouper tous les types sous une classe
- Il faut alors implémenter, entre autres, les constructeurs, les opérateurs de comparaison et les méthodes get/set
- Cette méthode devient rapidement très lourde, surtout s'il faut modifier le nombre d'éléments ou leurs types
- De plus, pour chaque combinaison nécessaire, il faut créer une nouvelle classe
- Cette méthode est donc à éviter

std::pair

- La librairie STL offre une classe utilitaire générique qui permet de regrouper deux éléments hétérogènes
- Elle est extrêmement pratique si le nombre d'éléments requis est petit
- Pour construire une paire, on peut utiliser le constructeur ou bien la fonction globale make_pair :

```
pair < int , string > objet(4, "bonjour");
objet = pair < int , string > (5, "salut");
objet = make_pair(6, "bye");
```

- La paire de STL nous donne un accès direct aux attributs first et second, qui représentent les deux valeurs contenues (elle viole donc le principe d'encapsulation)

std::pair

- Si le nombre d'éléments dépasse deux, il faut alors imbriquer les paires. Par exemple, pour quatre éléments, on obtient:

```
pair< pair< T1 , T2 > , pair < T3 , T4 > > quad;
```

- De plus, pour aller chercher les valeurs dans ce cas, il faut faire appel deux fois à l'opérateur .
- Par exemple, pour chercher la troisième valeur, on utilise:

```
quad.second.first
```

- On remarque donc que l'instruction n'est pas intuitive, elle ne donne pas directement l'information sur l'élément auquel on accède
- Les paires sont généralement à éviter lorsque le nombre d'éléments dépasse deux

std::tuple

- L'idéal serait donc d'avoir une classe qui permet de regrouper *n* éléments hétérogènes
- C'est exactement l'objectif de la bibliothèque <tuple>
- Elle offre la classe générique *tuple* qui permet donc de regrouper *n* éléments hétérogènes :

```
tuple< T1 , T2 > paire;
tuple< T1 , T2 , T3 > triplet;
tuple< T1 , T2 , T3, T4 > quadruple;
```

- Avant C++11, un tuple ne pouvait contenir que jusqu'à 10 éléments, depuis, c'est virtuellement illimité!

tuple

- Comme avec pair, les tuples peuvent être créés à l'aide du constructeur ou à l'aide la fonction globale make_tuple :

```
tuple < int , char , string > objet(4, 'f', "abc");
objet = tuple < int , char , string > (5, 'c', "def");
objet = make_tuple(6, 'z', "xyz");
```

- Pour accéder à un élément du tuple, on utilise la fonction globale du même nom, qui nous retourne une référence vers l'élément :

```
cout << get<2>(objet);  // Imprime le 3ème élément
get<0>(objet) = 10; // Modifie la valeur du 1er élément
```

tuple

- Les tuples implémentent certaines surcharges d'opérateurs :
  - Opérateur ==, qui retourne vrai si toutes les paires d'éléments sont égales (2 à 2)
  - Opérateur !=, qui retourne vrai s'il existe une paire d'éléments non égale (2 à 2)
  - Opérateurs <, <=, > et >=, qui implémentent des comparaisons lexicographiques
- Pour que l'utilisation de ces opérateurs soit valide, il faut que les tuples comparés soient du même type et que tous les types contenus dans ces tuples implémentent l'opérateur en question