

Compilation

TP 0.1 : Mise à niveau en C++

C. ALIAS & G. IOOSS

Catastrophe! le compilateur est en C++ et même pas en Caml. Mais quelle idée? En fait, on verra que beaucoup de traitements sont impératifs, et que C++ est plutôt bien adapté. En attendant voilà une rapide introduction qui suffira la plupart du temps.

Exercice 1. *Objet?*

Quand on programme, on a tendance à regrouper dans un fichier toutes les fonctions qui manipulent une structure de donnée. Voilà un exemple de fichier `list.h`:

```
typedef struct {...} list_t;

list_t* empty_list();
list_t* add(list_t*, int);

int get_nth(list_t, int);
int length(list_t*);
```

Le paradigme objet fournit un cadre syntaxique pour expliciter ce regroupement. Les fonctions (méthodes) sont syntaxiquement rattachées à une variable (objet) d'un certain type (classe). Au lieu d'écrire `add(list,2)`, on écrit `list.add(2)`...

Le type (la classe) d'un objet `list` se déclare de la façon suivante:

Fichier `List.h`:

```
class List
{
public:
    boolean is_empty;
    int head;
    List* tail;

    // Declaration de la liste
    List(); // Constructeurs
    List(List*,int);

    void add(int e); // Méthodes
    int get_nth(int);
    int length();
};
```

Dans `List.cc`, on implémente les méthodes:

```
#include "List.h"

List::List()
{ this->is_empty = true; } // this: pointeur sur l'objet en construction

List::List(List* t, int h)
{ is_empty = false; tail = t; head = h; }
```

Ces 2 méthodes sont des *constructeurs*. Elles permettent de construire un objet de type (classe) `List` en mémoire. Par exemple:

```
List* l1 = new List(); //[], appel au premier constructeur
List* l2 = new List(new List(),1); //[1]
```

Noter le 'this' qui est un pointeur sur l'objet en cours de construction (retourné par le constructeur, en fait). Il peut rester implicite, comme dans le deuxième constructeur. Dans les méthodes, this est un pointeur sur l'objet manipulé. Dans l1->add(1), le 'this' de add vaudra l1.

Manip.

- Recopier le code ci-dessus dans les fichiers `List.h` et `List.cc` respectivement. Finir d'implémenter les méthodes.
- (Pour aller plus loin) Implementer une classe d'arbre binaire `BinTree`, admettant:
 - Deux constructeurs `BinTree()` et `BinTree(BinTree*,BinTree*,int)`
 - Une méthode pour calculer la profondeur maximale de l'arbre `height`
 - Une méthode pour additionner tous les éléments de l'arbre (`int` stocké au niveau des noeuds)

Exercice 2. Entrées/Sorties

Recopier, compiler, tester:

```
#include <iostream>

using namespace std;

int main(void)
{
    cout << "Hello" << " world" << endl;
}
```

On devrait plutôt écrire `((cout << "Hello") << " world") << endl; :`

- `(cout << "Hello")` ajoute la chaîne "Hello" au flux de sortie standard (écran) et retourne le flux résultant.
- `(...) << "world"` ajoute "world" au flux résultant. Etc, etc...

En C++, on peut définir les opérateurs (+, -, *, «, etc) sur des objets. Écrivons un « sur nos List. Ajouter:

```
à List.h:
ostream& operator<< (ostream& sout, List& l);
```

```
à List.cc:
ostream& operator<< (ostream& sout, List& l)
{
    if(l.is_empty) return sout;
    sout << l.head << " " << *(l.tail);
    return sout;
}
```

On peut alors écrire `cout << *ma_liste << endl;...` Mais on peut tout aussi bien remplacer `cout` par un flux fichier (voir la classe `fstream`), et écrire la liste dans un fichier.

Manip.

- En vous aidant de votre moteur de recherche favori, modifier/compléter le programme pour écrire une liste dans un fichier.

Exercice 3. STL

La STL (Standart Template Library) est une bibliothèque qui implémente une fois pour toutes les listes et les ensembles. Autrement, ça ne ferait pas sérieux. Personnellement, j'utilise beaucoup `vector` (des tableaux de tout) et `map` (listes d'association à la caml).

Voilà quelques idiomes à connaître (sans forcément comprendre le détail):

```
#include <iostream> // Pour le "cout"
#include <vector>
#include <map>

using namespace std;

int main(void)
{
    // Vector
    vector<int> vec; //table
    vec.push_back(7); // [7]
    vec.push_back(5); // [7,5]
    cout << vec.size << endl //2
         << vec[0] << endl //7
         << vec[1] << endl; //5

    // Map
    map<string,int> age;
    age["toto"] = 25;
    age["titi"] = 8;
    if(age.find("toto") != age.end())
        //toto existe?
        cout << "toto a " << age["toto"] << "ans" << endl;

    for(map<string,int>::const_iterator it = age.begin();
        it != age.end(); ++it)
        //Pour chaque couple (name,age) dans le map...
        cout << (*it).first << " -> " << (*it).second << endl;

    age.begin() retourne un pointeur sur le premier élément (typé et appelé "itérateur"). age.end()
    pointe après le dernier élément. D'où le if..
```

Manip.

- Recopier le code et expérimentez.
- (Pour aller plus loin) De même, expérimentez avec la structure de `pair<T1,T2>`. Quel est la relation entre la structure `map` et la structure `pair`?