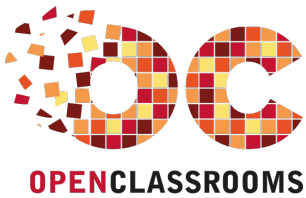
Foncteurs et itérateurs

Par Ice_Keese



www.openclassrooms.com

Dernière mise à jour le 5/02/2011

Sommaire

Foncteurs et itérateurs

Par  Ice_Keese

Mise à jour : 05/02/2011

En C++, comme dans tout langage orienté objet, on retrouve la notion de foncteur (contraction de *Function Object*). Un foncteur est un objet qui se comporte comme une fonction, et cela peut avoir une grande utilité dans bien des cas. Pour comprendre la matière de ce tuto, vous devrez vous y connaitre un peu en pointeurs.
Sommaire du tutoriel :

Itérateurs et for_each

Avant de voir les foncteurs, nous verrons d'abord une fonction très pratique: for_each. Cette fonction effectue une action quelconque sur les éléments d'une séquence. Mais pour comprendre for_each, il faut d'abord comprendre la notion d'itérateur.

Un itérateur est en fait un pointeur. On le nomme "itérateur" car sa fonction est d'"itérer", c'est à dire parcourir une séquence.

Une séquence est une suite de données, comme un vecteur, une chaine de caractères ou une liste. Elle est identifiée par un début et une fin. Par convention, on considère que la fin ne fait pas partie de la séquence. C'est également pourquoi, dans un tableau de 5 cases, la case 5 est exclue du tableau.

Imaginons le code suivant:

Code : C++

```
#include <iostream>
int main()
{
    const int NB_ELEMENTS = 5; // Taille du tableau
    int elems[NB_ELEMENTS] = {1,4,7,12,20}; // Tableau
    for(int * p = elems; p != elems + NB_ELEMENTS; ++p)
    {
        std::cout << *p << " "; // On affiche chaque élément
    }
}
```

Ce code peut vous paraitre étrange. Le nom d'un tableau correspond à l'adresse de son premier élément. Ainsi, ces deux bouts de code sont équivalents:

Code : C++

```
*(elems + 1);
```

Code : C++

```
elems[1];
```

Pas de magie ici. Donc, on déclare un pointeur sur un entier en le faisant initialement pointer vers le début du tableau. Puis, tant que l'on n'a pas atteint la fin du tableau (c'est-à-dire le début plus sa taille), on affiche la valeur pointée et on incrémente le pointeur (donc, on avance d'une position en mémoire).

Remarquez l'usage du signe != au lieu de <. En effet, on ne sait pas où l'on se trouve en mémoire, nous ne sommes pas nécessairement avant l'élément! L'utilisation d'un signe < pourrait ne pas donner l'effet escompté!

Pourquoi ne pas simplement utiliser une boucle à compteur?

Avec un tableau, il n'y a pas d'avantage, puisque les données peuvent être accédées aléatoirement (vous pouvez donc demander n'importe laquelle à n'importe quel moment). C'est la même chose avec un vecteur. Or, vous ne pouvez pas faire cela avec une file ou une pile (à moins d'avoir l'adresse de l'élément, ce qui revient à utiliser un itérateur). Les itérateurs constituent donc une approche relativement standardisée de parcourir une séquence, peu importe comment elle est construite et ce qu'elle contient, en ce sens qu'une fois qu'on sait s'en servir, on peut les utiliser à peu près partout sans trop de modifications.

Mais bon, vous venez de coder un itérateur! Bravo!

Les types standards comme vector, deque ou string possèdent leurs propres itérateurs internes. Imaginons une fonction pour transformer tous les caractères d'une std::string en majuscules.

Code : C++

```
#include <string>
#include <locale>

std::string majuscules(const std::string &chaineMinuscules)
{
    std::string chaineMajuscules = chaineMinuscules; // En faisant
une copie, on s'assure que la chaine originale ne sera pas altérée
    for(std::string::iterator it =  chaineMajuscules.begin(); it !=
chaineMajuscules.end(); ++it)
    {
        *it = std::toupper(*it,std::locale(""));
    }
    return chaineMajuscules;
}
```

string::iterator est le type d'un itérateur sur un élément d'une string. Théoriquement, c'est un char*, mais ça peut-être autre chose d'un compilateur à l'autre, alors ne prenez pas pour acquis que c'est un pointeur sur un char! Les méthodes begin() et end() (présentes dans la plupart des conteneurs standards, comme vector ou deque) retournent respectivement un itérateur sur le début et la fin de la chaine (ou du vecteur, ou de la file, etc.). toupper() retourne la version en masjuscules du caractère passé en paramètre. Le second paramètre de la fonction, locale(""), permet de prendre en charge les caractères spéciaux comme les accents. Aucun paramètre signifie l'anglais. Une chaîne vide correspondra à la localité utilisée sur la machine, et vous pouvez forcer l'utilisation d'une localité en entrant le nom de celle-ci.

Le code précédent est donc relativement simple. Un itérateur parcourt la chaîne, et à chaque tour de boucle (à chaque itération), le caractère qui se trouve à la position de l'itérateur est mis en majuscules.

Mais nous pourrions améliorer cette fonction, à l'aide de notre ami for_each. for_each est une fonction présente dans le fichier d'entête standard algorithm.

algorithm est le meilleur ami du programmeur C++, apprenez à vous en servir!

Sa syntaxe est:

Code : C++

```
std::for_each(debut, fin, fonction)
```

Il applique une fonction aux éléments d'une séquence. Cette fonction ne doit posséder qu'un seul paramètre, et le type de ce paramètre n'est pas important, tant qu'il est identique aux type des éléments de la séquence parcourue. Avec des templates, vous pouvez éliminer cette contrainte.

Code : C++

```
#include <string>
#include <locale>
#include <algorithm>

void _majuscules(char & c)
{
   c = std::toupper(c,std::locale(""));
}

std::string majuscules(const std::string &chaineMinuscules)
{
    std::string chaineMajuscules = chaineMinuscules;
    std::for_each(chaineMajuscules.begin(), chaineMajuscules.end(),
_majuscules);
    return chaineMajuscules;
}
```

Ouch. La première version de majuscules, celle qui prend un char et retourne un char, agit comme intermédiaire entre for_each et le reste du programme. En effet, toupper() prend deux paramètres, et for_each ne fonctionne qu'avec des fonctions ne prenant qu'un seul paramètre!
Mais je vous vois sourciller à la vue de la fonction for_each. Les deux premiers paramètres sont facilement identifiables: il s'agit du début et de la fin la chaine, respectivement. Mais quel est le troisième? C'est le même nom que la fonction qui prend un char et retourne un char...

Eh bien, c'est la fonction.

En C++, le nom d'une fonction (sans les parenthèses), c'est un peu comme un pointeur sur celle-ci.

for_each, pour chaque élément entre begin() et end(), appellera la fonction passée en paramètre en lui passant comme paramètre l'élément. Ceci est donc équivalent à:

Code : C++

```
for(std::string::iterator it = chaineMajuscules.begin(); it !=
chaineMajuscules.end(); ++it)
{
    _majuscules(*it);
}
```

Remarquez qu'avec for_each, la valeur de retour de la fonction est ignorée. Vous devez donc créer une fonction qui reçoit une référence si vous voulez modifier la séquence.
L'avantage? Vous réduisez le nombre de lignes de code et le risque d'erreurs potentielles. Notez qu'un for_each n'est pas approprié dans toutes les situations, pour la simple raison qu'il repose sur une fonction. Si vous construisez des objets dans cette fonction, ils seront détruits à la fin, si bien qu'entre chaque itération du for_each, il y a aura constructions et destructions! Il y a deux solutions à ce problème: la première est de construire ces objets avant la boucle, et d'employer un for classique, et la seconde est d'utiliser un foncteur.

Avant de finir, un peu d'information sur ce qu'on peut faire avec des itérateurs et la STL. On retrouve, dans la STL, beaucoup d'algorithmes qui fonctionnent sur une séquence d'éléments, comme reverse(), qui inverse une séquence, random_shuffle(), qui mélange les éléments d'une séquence, ou copy(), qui copie les éléments d'une séquence dans une autre:

Code : C++

```
#include <vector>
#include <algorithm>

int main()
{
    std::vector v1,v2,v3,v4;
    // Admettons qu'on les remplisse ici...
    std::reverse(v1.begin(), v1.end()); // Inverse les éléments de
```

```
v1
    std::random_shuffle(v2.begin(), v2.end()); // Mélange les
éléments de v2, attention de bien appeller srand() avant!
    std::copy(v3.begin(), v3.end(), v4.begin()); // Copie les
éléments de v3 dans v4. Attention, plante si v4 est plus petit que
v3...
}
```

On emploie ici des vecteurs, mais si v1, v2, v3 et v4 étaient des deque, il n'y aura absolument aucun changement au code qui s'en sert (sauf peut-être pour les remplir).

On retrouve aussi, avec les conteneurs standards, des "constructeurs de séquences", qui prennent un itérateur sur le début et sur la fin d'une séquence. On peut donc construire une std::string à l'aide d'un tableau de char:

Code : C++

```
#include <string>

std::string tabCharToString(char tab[], int longueur)
{
    return std::string(tab, tab + longueur);
}
```

Une chaine est construite avec les données contenues entre le début et la fin de la séquence, puis retournée.
Ce ne sont que des exemples. Les itérateurs sont des outils très puissants!

Foncteurs

Ah, enfin! Nous voici enfin aux foncteurs. Si vous avez compris la partie sur les itérateurs, ça devrait bien aller.

Un for_each ne fait qu'appliquer des parenthèses à son troisième paramètre. Ainsi, tout ce qui peut prendre des parenthèses peut passer dans un for_each.

Le saviez-vous? Parmi les opérateurs surchargeables en C++, on retrouve les parenthèses...

Les foncteurs sont une façon élégante de solutionner le problème de paramètre seul dans le for_each. Car, actuellement, imaginons que nous ayons un vecteur de string et que nous voudrions écrire son contenu dans un fichier. Nous voudrions naturellement pouvoir spécifier le fichier où l'écrire, mais malheureusement, le seul paramètre de la fonction sera occupé par une chaine de caractères... comment procéder? Ou vous abandonnez et codez un bon vieux for, qui sera moins performant (vous avez beau essayer, vous ne pouvez pas battre la STL), ou vous utilisez un foncteur.

Un foncteur, en fin de compte, c'est une classe qui surcharge l'opérateur (). Rien de bien sorcier. Voici une classe bien générale qui, grâce aux templates, peut écrire n'importe quelle valeur dans un flux standard:

Code : C++

```
// ecrireDansFlux.h
#ifndef ECRIRE_DANS_FLUX_H
#define ECRIRE_DANS_FLUX_H

#include <iostream>

class ecrireDansFlux
{
   std::ostream & flux_;

public:
   ecrireDansFlux(std::ostream & flux)
   : flux_(flux)
   {
   }
```

```
    template <class T>
    void operator() (const T &val)
    {
        flux_ << val;
    }
};

#endif
```

Les flux standards ne peuvent être copiés (vous pouvez bien essayer, ça ne compilera pas), alors il est impératif de passer une référence au constructeur. Cette classe fonctionnera avec tout type qui supporte l'opérateur <<.
Et voici le code qui s'en sert:

Code : C++

```
#include <vector>
#include <string>
#include <algorithm>
#include <fstream>

#include "ecrireDansFlux.h"

void ecrireVecteurDansFichier(const std::string &filename, const
std::vector<std::string> &v)
{
    std::ofstream flux(filename.c_str()); // Flux d'écriture dans le
fichier.
    std::for_each(v.begin(), v.end(), ecrireDansFlux(flux));
}
```

Voilà.

C'est tout?

Yep. Le flux est ouvert et pointe vers le fichier dont le nom est contenu dans la std::string. Si le fichier n'existe pas, il est créé automatiquement. De plus, il sera fermé à la fin de sa portée (donc, la fin de la fonction). Ensuite, ce flux est passé au constructeur d'un objet de classe ecrireDansFlux, lequel sera ensuite acheminé vers le for_each, qui le traitera comme une fonction puisqu'il implémente l'opérateur (). Le for_each appellera donc cet opérateur, en lui passant une à une les chaines du vecteur. Simple et efficace. Puisque std::string a surchargé l'opérateur << pour les flux standards, le tout compile.

En fait, on pourrait même faire ça:

Code : C++

```
void ecrireVecteurALecran(const std::vector<std::string> &v)
{
    std::for_each(v.begin(), v.end(), ecrireDansFlux(std::cout));
}
```

Lequel affichera le contenu du vecteur à l'écran, puisque std::cout est un flux de sortie valide.

Je le répète: un foncteur est une classe qui surcharge l'opérateur (). Une fois ce critère rempli, vous pouvez faire ce que vous voulez avec la classe. Mais attention avec votre nouvel outil: un proverbe anglophone dit que pour celui qui a un marteau doré, tout ressemble à un clou...
Je vous laisse méditer là-dessus.

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Qu'est-ce qu'un itérateur?

- ◯ Une fonction
- ◯ Un pointeur
- ◯ Une classe
- ◯ Un opérateur

Sachant que elems est un tableau d'entiers, que fait le code suivant?

Code : C++

```
std::cout << elems + 1;
```

- ◯ Il affiche le contenu de la deuxième case du tableau
- ◯ Il affiche la valeur de la première case, plus 1
- ◯ Il affiche l'adresse de la deuxième case du tableau

Que fait std::for_each?

- ◯ Il applique une fonction sur une séquence
- ◯ Il affiche les éléments d'une séquence
- ◯ Il modifie les éléments d'une séquence

Qu'est-ce qu'un foncteur?

- ◯ Une fonction
- ◯ Un pointeur
- ◯ Une classe
- ◯ Un opérateur

Correction !