Containeurs & Itérateurs

Nouveaux paradigmes de manipulation des données offerts par les langages à objets

Séparer Structure de Données et Algorithmes

Idée : un algorithme de recherche d'un plus grand élément

```
template<class T>
T find_greatest_element(T theArray[], size_t theNumberOfElements)
{
    if (theNumberOfElements == 0)
        return T();
    T greatestElement = theArray[0];
    for (int index = 1; index < theNummberOfElements; index++)
    {
        if (theArray[index] > greatestElement)
            greatestElement = theArray[index];
    }
    return greatestElement;
}
```

Comment faire fonctionner cet algorithme sur d'autres structures de données que des tableaux ?

Comment accéder aux éléments dans un ensemble de données

- L'approche séquentielle (uniquement en avant)
  - Accès au premier élément de l'ensemble de données
  - Accès à l'élément suivant
  - Ce modèle correspond au modèle de flux (données venant d'un réseau, etc.) sans mémoire.

Comment accéder aux éléments dans un ensemble de données

- L'approche séquentielle (en avant et en arrière)
  - Accès au premier élément de l'ensemble de données
  - Accès à l'élément suivant
  - Accès à l'élément précédent
  - Ce modèle correspond au modèle de données stockées sur une bande magnétique par exemple.

Comment accéder aux éléments dans un ensemble de données

- L'approche à accès indexé
  - Accès au premier élément de l'ensemble de données
  - Accès au dernier élément de l'ensemble
  - Accès à n'importe quel élément à partir de sa position.
  - Ce modèle correspond au modèle de données stockées dans un tableau par exemple.

Idée : Rendre les structures de données conforme à un modèle d'accès

- Une structure de données ou toute structure modélisant un ensemble de données doit :
  - Etre conforme à un modèle d'accès donné,
    - Ie. fournir soit un accès séquentiel allant du premier au dernier élément.
    - Fournir un accès séquentiel bidirectionnel
    - Fournir un accès indexé aux données dans la structure.

Comment fournir une référence aux données internes de la structure ?

- Un itérateur est une classe qui :
  - Identifie de manière unique une donnée stockée dans la structure de donnée
  - Offre une méthode ou un opérateur qui accède à la donnée référencée par l'itérateur)
  - Offre une méthode ou un opérateur qui permet de référencer la donnée suivante (parfois la donnée précédente).

Un « Input »Iterator en C++

- L'itérateur doit fournir au moins le comportement suivant :

Propriétés Imposées par le contrat	**Expressions valides**
Il supporte la recopie, l'affectation et la duplication. Les deux itérateurs après recopie font référence à la même donnée.	X b(a); b = a;
Si deux itérateurs sont égaux, cela signifie qu'ils font reference à la même donnée.	a == b a != b
L'operation de déréférencement permet d'accéder à la donnée stockée dans la structure de donnée.	*a a->m
Il peut être incrémentée. Dans ce cas, soit l'itérateur fait référence à la donnée stockée après la donnée actuellement référencée, soit au contraire, il	++a (void)a++ *a++
Deux itérateurs peuvent-être échangés, ie. le premier opérateur fait après l'échange référence à la donnée référencée par le second itérateur et le second itérateur fait après l'échange référence à la donnée référencée par le premier itérateur.	swap(a,b)

Une structure de données « énumérable » en C++

- Doit fournir au moins deux fonctions

begin

Retourne un itérateur faisant référence à la première donnée présente dans la structure ou à l'itérateur désignant la fin des données.

end

Retourne l'itérateur désignant la fin des données de la structure.

Transformer l'algorithme pour fonctionner sur des itérateurs.

```
template<class inputIterator, class T>
inputIterator find_greatest_element(inputIterator first,
    inputIterator end, T& value)
{
    if(first == end)
       return end;
    inputIterator greatest = first; value = *first;
    first ++;
    for (; first != end; first ++) {
        if (*first > value) {
            greatest = first;  value = *first;
        }
    }
    return greatest;
}
```

Généralisation de l'usage

```
int test()
{
    int myints[] = { 10,20,30,30,20,10,10,20 };
    std::vector<int> v(myints, myints + 8);
    int maxValue;
    std::vector<int>::iterator up =
      find_greatest_element(v.begin(), v.end(),
      maxValue);
    std::cout << "greatest value at position "
      << (up - v.begin()) << '\n';
    return 0;
}
```

TD --- Utilisation d’itérateur

- Créer un algorithme manipulant des structures itératives.
- Tester cet algorithme.
- Créer une classe itérateur pour manipuler des intervalles d’entier.