GÉNÉRICITÉ : PATRONS DE CLASSE (TEMPLATES)

Patrons : modèle à partir duquel des classes ou des fonctions pourront être générées automatiquement par le compilateur en fonction d'une série de paramètres. Permet de définir en particulier :

- des classes dites "containers" (listes, tableaux...), avec vérification statique et sans perte d'efficacité.
  Exemple : ajouter un element à un tableau, supprimer un element, parcourir le tableau, etc. pour tous les types de tableaux.
- des traitements (fonctions) génériques pour un ensemble de types.
  Exemple : tri, recherche d'un élément, etc.

• Patron de fonction

syntaxe

```
template <class T> type_retour patron_fonction (liste)
    { code};
```

- Un patron de fonction est paramétré par une ou plusieurs classes.
- Ce patron génére une version spécifique à une classe, lorsque le code source le nécessite. Il n'y a pas d'action spécifique à réaliser en dehors du patron pour créer ces fonctions.
- La surcharge des fonctions patron est possible par des fonctions générées par le même patron ou des fonctions de nom identique.
- La résolution de la surcharge est faite dans l'ordre suivant :
  - correspondance exacte avec une fonction
  - recherche d'un patron de fonction pouvant générer la fonction exacte
  - résolution de la surcharge ordinaire

```
template<class T> T max ( T a, T b) { return (a>b ? a : b);
main() { int a,b,m1;  char c,d,m2
  m1=max(a,b); // OK max (int , int)
  m2=max(c,d); // OK max (char, char)
  m3=max(a,c); // ERREUR pas de patron pour max (int, char)
}
```

Le troisième appel ne peut être résolu que si la fonction
`int max( int, char)` existe (règle 3 de la résolution).

- Chaque argument spécifié dans la déclaration du patron doit être utilisé dans la déclaration des arguments de la fonction.

```
template<class T> T*valeur() {•••} ;        // ERREUR
template<class T> void f() { T inter; •••};  // ERREUR
```

GÉNÉRICITÉ : PATRONS DE CLASSE (TEMPLATES)
(2)

- **Patron de classes**

Syntaxe du patron

```
template <class T> class nom_patron
     { déclaration attributs ;
       déclaration méthodes ;
     };
```

Définition d'un objet

```
nom_patron<type_ou_classe> nom_objet(paramètres du
                                     constructeur);
```

- Il est inutile de créer explicitement les classes à partir du patron
- Ces classes sont créées lorsqu'on utilise des objets.

- **Exemple**

```
template <class T> class vecteur {
    int nb;
    T *v; // Tableau d'éléments de type T
    public :
     vecteur (int n) {nb = n ; v=new T[n]}
      // Constructeur
    T& operator[] (int i) { return v[i]; }
    T somme () { T s ;
      for (int i=0 ;i<nb ; i++) s = s+v[i] ;
      return s ; }
      // Suppose que + soit défini pour T
    •••
};
main() {
    vecteur<int> t1(10);          // Tableau de 10 entiers
    vecteur<complexe> t2(50);     // Tableau de 50 complexes
}
```

Partout où un nom de classe est nécessaire, on peut utiliser `vecteur<type>`, en particulier, lors de l'héritage :

```
class FENETRE { ••• };
class MENU : public vecteur<FENETRE> { ••• };
```

INTRODUCTION À LA STANDARD TEMPLATE LIBRARY

Objectif : fournir des outils pour créer données et algorithmes standards

Remarque de base : **pas de liaison dynamique**, tout est compilé 🖧 rapidité

- ## Organisation : 4 types d'éléments

1. conteneurs
2. itérateurs,
3. méthodes
4. allocateurs

- ## Les conteneurs

- structures de données classiques contenant des éléments : le type d'éléments utilisés paramètre ces conteneurs (généricité)
- obligatoirement associé à un itérateur

- taille en général variable dynamiquement, même pour les vecteurs
  - conteneurs linéaires
    - tableaux (**vector** , **deque**, **bitvector**)
    - listes chainées simples (**slist**), double (**list**)
  - ensembles
    - valeur : ensembles simples (**set**), multiples (**multiset**)
    - valeur : ensembles simples à accès calculé via table de hashage (**hast_set**), multiples (**hash_multiset**)
  - conteneurs associatifs
    - paire clé-valeur : conteneurs associatifs simples (**map**), multiples et doublons autorisés (**multimap**)
  - strings
  - Conteneurs spécifiques
    - piles (**stack**),
    - files (**queue**)
    - files de priorité (**priority_queue**)

COMPLÉMENTS SUR LE C++

INTRODUCTION À LA STANDARD TEMPLATE LIBRARY (2)

• Les itérateurs

- Extension de la notion de parcours par pointeurs en C

- permettent de séparer le code des méthodes (exemple : reverse) des données elles-mêmes : inverser un vecteur et une liste est fondementalement le même algorithme
  - ⇒ une seule fonction à écrire
- **rôle : donner la valeur d'une donnée et passage à la suivante (itération).**

- types :
  - itérateurs pour algorithmes en une seule passe : leur valeur est changée à chaque itération par incrémentation
    - Input Iterator : lecture des valeurs uniquement
    - Output Iterator : écriture des valeurs uniquement

  - itérateurs pour algorithmes en multi passe : valeur constante du pointeur (type constant) ou modifié à chaque itération (type mutable)
    - Forward Iterator : itération == incrémentation
    - Bidirectional Iterator : itération == incrémentation ou décrémentation
    -
- Random Access Iterator.

- Trivial Iterator

• Les allocateurs

- Ils permettent de choisir entre plusieurs possibilités d'allocation de mémoire (afin de contenir les données).
- Tous les conteneurs prennent en dernier paramètre générique un type d'allocateur.

Introduction à la Standard Template Library (2)

• Les méthodes ou fonctions

- algorithmes de bases des types abstraits
- utilisent les itérateurs pour accéder aux données (code réutilisable)

- Types d'algorithmes
  - appliquer une fonction à des données : foreach
  - recherche d'une valeur : find, find_if, find_end, search, search_n
  - comptage : count, count_if
  - copie : copy, copy_n, copy_backward, unique_copy, reverse, reverse_copy, rotate, rotate_copy
  - échange : swap, iter_swap, swap_ranges
  - remplacement : replace, replace_if, replace_copy, replace_copy_if
  - suppression : remove,remove_if, remove_copy, remove_copy_if
  - remplissage : fill, fill_n, generate, generate_n, random_sample
  - tris : sort, stable_sort, partial_sort, partial_sort_copy, is_sorted
  - recherche binaire, fusion : lower_bound, upper_bound, equal_range, binary_search, merge
  - opérations ensemblistes : includes, set_union, set_intersection, set_difference, set_symmetric_difference
  - extremas : min, max, min_element, max_element
  - opérations sur les tas : push_heap, pop_heap, make_heap, sort_heap, is_heap
  - Algorithmes numériques de base : accumulate, inner_product, partial_sum, adjacent_difference, power

STL : EXEMPLE SIMPLE

```
#include <iostream>
#include <vector>
#include <list>
using namespace std;


main() {
    // Declare a vector of 3 elements.
    vector<int> v(3);

    v[0] = 7;
    v[1] = v[0] + 3;
    v[2] = v[0] + v[1]; // v[0] == 7, v[1] == 10, v[2] ==
17

    // inverse le vecteur
    reverse(v.begin(), v.end()); // v[0]==17, v[1]==10,
v[2]==7

    // trouve si la valeur 7 est dans le vecteur
    vector<int>::iterator result = find(v.begin(),
v.end(), 7);

    // compte le nmbre de 7 dans le vecteur
    cout << "Nombres de 7:"<<count(v.begin(),v.end(),0)<<
endl;

    // Creation d'une liste d'entier de la même taille
que v
    list<int> L(v.size());

    // Copy des entiers de v dans la liste L
    copy(v.begin(), v.end(), L.begin());
}
```

STL : VECTOR

- éléments adjacents en mémoire,
- réallouer automatiquement par bloc en cas de besoin lors d'un push_back
  - Erreur en cas d'accès par []
- Complexité
  - Accès O(1)
  - Insertion : O(n) en début de vector (pop_back), O(1) en fin ou milieu de vector (push_back).

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    // Declare a vector of 3 elements.
    vector<int> v(3);

    v[0] = 7;
    v[1] = v[0] + 3;
    v[2] = v[0] + v[1]; // v[0] == 7, v[1] == 10, v[2] ==
17
    // v[4]=4. v[5]=2,
  v.push_back(4);
  v.push_back(2);
  v.push_back(5);

  // on peut utiliser les iterators ou les index
  for(int i=0;i<v.size();++i) cout << v[i] << ' ';
  cout<<endl ;
  vector<int>::const_iterator it(v.begin()) ;
  for(;it!=v.end();++it) cout << *it << ' ';
return 0 ;
}
```

STL : LIST

- Liste doublement chainée ,
- Complexité
  - Insertion, suppression : O(1) en début/fin de liste
  - Acces, recherche : O(n)

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    // Declare a list.
    list<int> v;

  v.push_back(4);
  v.push_back(2);
  v.push_back(5);

  // on utilise les iterators
  list<int>::const_iterator it(v.begin()) ;
  for(;it!=v.end();++it) cout << *it << ' ';
  cout<<endl ;
  return 0 ;
}
```

STL : SET

- Ensemble ordonné sans doublons
- Imlanté avec un arbre rouge-noir
- Complexité
  - Insertion : O(log n)
  - Acces, recherche : O(log n)
- Attention : ne pas modifier un set dans une boucle avec des itérateurs
  - La modification invalide les itérateurs

```
#include <iostream>
#include <set>
using namespace std;

int main() {
    // Declare a set.
  set<int> v;

  v.insert(4);
  v.insert(2);
  v.insert(5);

  // on utilise les iterators
  set<int>::const_iterator it(v.begin()) ;
  for(;it!=v.end();++it) cout << *it << ' ';
  cout<<endl ;

  v.insert(2);
  for(it=v.begin();it!=v.end();++it) cout << *it << ' ';

  return 0 ;
}
```

STL : MAP

- Table associative clé-valeur
- Imlanté avec un arbre rouge-noir
- Complexité
  - Insertion : O(log n)
  - Acces, recherche : O(log n)

- l'opérateur [ ] crée et insère la clé avec la donnée T() dans la map

- un itérateur retourne une paire (pair) avec 2 membres ; first pour la clé, second pour la valeur

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main() {
    // Declare a map.
  map<string,int> v;

  v["janvier"]=1;
  v["aout"]=8;
  v["juin"]=6;

  // on utilise les iterators qui retourne des paires
  map<string,int>::const_iterator it(v.begin()) ;
  for(;it!=v.end();++it)
     cout << it->first << ' '<< it->second << ' ';
  cout<<endl ;

  return 0 ;
}
```

RÉFÉRENCES

- ## Guide de reference C++

Stroustrup: *The C++ Programming Language*, Fourth Edition, Addison Wesley. Reading Mass. USA. May 2013. ISBN 0-321-56384-0. 1360 pages

C++ reference documentation. En ligne : http://www.cplusplus.com/reference/

- ## Recueil de conseils C++ / FAQ technique

FAQ Lite C++ de Marshall Cline.
En ligne : http://www.parashift.com/c++-faq/
Livre : Cline, Lomow, and Girou, C++ FAQs, Second Edition, 587 pgs, Addison-Wesley, 1999, ISBN 0-201-30983-1.

- ## Ouvrages pédagogiques C++

Tutoriel C++ du *C++ resource network.*
En ligne : http://www.cplusplus.com/doc/tutorial/

- ## UML

Pierre-Alain MULLER, Nathalie GAERTNER : *Modélisation objet avec UML*. Best of Eyrolles ed. 2003.