Travaux pratiques sur STL (Standard Template Library)

Travaux dirigés et pratiques - PO2 (UE ULIN606)

Compléments sur les conteneurs L'interface des classes conteneurs contient un ensemble de définitions de types publiques, telles que (pour un conteneur paramétré par le type `T`) :

- `typedef T value_type`, type des éléments,
- `typedef ... iterator`, type des iterateurs sur le conteneur (la partie en pointillé dépend de l'implémentation choisie),
- `typedef ... const_iterator`, type des iterateurs constants sur le conteneur (la partie en pointillé dépend de l'implémentation choisie).

Ces définitions de types permettent d'écrire des fonctions très générales, s'appliquant à tous les conteneurs. Pour les utiliser, il faut les faire précéder du mot-clef `typename`. Comme le dit B. Stroustrup lui-même « Having to add typename before the names of the members types of a template is a nuisance » qu'il semble justifier par la difficulté des compilateurs à reconnaître ces types. Admettons, le résultat est que dans les nouveaux compilateurs, cette nuisance s'applique comme vous le montre le code suivant que nous vous invitons à tester pour faire la somme des éléments d'un vecteur d'entiers et d'un set de float.

```
template<typename C>
// le parametre doit etre un conteneur
// C::value_type doit etre un type muni de l'addition
// et 0 son élément neutre
typename C::value_type somme(const C&c)
{
typename C::value_type s=0;
typename C::const_iterator p = c.begin();
while (p != c.end()){s+=*p; p++;}
return s;
}
```

Comptage de mots Tout comme vous l'avez fait lors du TP sur le dictionnaire associatif, utilisez son équivalent en STL, la classe `map`, pour compter le nombre de mots d'un texte et afficher le résultat du comptage.

Effacement conditionnel Nous vous proposons d'examiner l'algorithme d'effacement conditionnel `remove_if`. Sa sémantique est très particulière : la fonction ne supprime pas d'éléments en mémoire, elle les écrase, et retourne la fin valide du conteneur. Remarquez au passage que les tableaux se manipulent comme les conteneurs, et où se trouve l'itérateur (la position) de fin du conteneur ! Essayez les mêmes manipulations avec un vecteur d'entiers. Notez le type de la fonction passée en paramètre à `remove_if`.

```
bool odd (int a)
{ return a % 2;}

int numbers[6] = { 0, 0, 1, 1, 2, 2 };

int main ()
{
  int* ite =
  remove_if (numbers, numbers + 6, odd);
  for (int i = 0; i < 6; i++)
    cout << "adresse element " << i << " = " << (int)(&numbers[i])
         << " valeur = " << numbers[i] << endl;
  cout << endl;
  cout << "adresse retournee par remove_if =" << (int)ite;
  cout << endl;
  return 0;
}
```

Un effacement conditionnel plus satisfaisant Ecrivez une fonction `MonRemove_if`, paramétrée par un conteneur et un prédicat (une fonction booléenne, ...inspirez-vous de l'entête de `remove_if` dans le fichier `algorithm`), qui supprime effectivement les éléments du conteneur vérifiant le prédicat. Utilisez `remove_if` et `erase`.

Classes fonctions Vous trouverez ci-dessous une classe dont les instances sont des fonctions testant la divisibilité.

```
class divisiblePar {
private:
  int diviseur;     // donnée membre : le diviseur
public:
  divisiblePar(int div) : diviseur(div) {}; // constructeur
  bool operator()(const int& dividende) { //
     return (dividende%diviseur)==0 ;
  }
};
```

Testez-la avec la fonction suivante, et analysez-la. Comparez cette classe et ses instances à une fonction classique "divisiblePar" à deux paramètres.

```
void TesteDivisiblePar()
{
 divisiblePar f(2);
 divisiblePar g(3);

 cout << "f(4) = " << f(4) << endl;
 cout << "f(8) = " << f(8) << endl;
 cout << "f(5) = " << f(5) << endl;
}
```

Que vaudrait `g(9)` ?

Affichage Ecrivez un modèle de fonction d'affichage, paramétrée par un type `C`, supposé être un type de conteneur, prenant comme paramètre un conteneur (passé par référence), et affichant sur le flot de sortie standard `cout` tous les éléments du conteneur. Vous utiliserez `C`::`value`, qui représente le type des éléments du conteneur, pour paramétrer l'`ostream_iterator`.

Crible d'Erathostènes On souhaite connaître (en les affichant, par exemple) tous les nombres premiers jusqu'à une certaine borne `B`. Utilisez une liste (`list` en STL pour stocker tous les entiers de l'intervalle [2, B]. Puis supprimez successivement tous les multiples des nombres premiers rencontrés, en vous servant de `MonRemove_if`. Vous pouvez extraire ou non les nombres premiers de la liste quand vous les trouvez.

Ensemble ordonné Ne faites cet exercice que si vous avez terminé tous les autres travaux pratiques. Il est volontairement plus libre et vous demande un peu de conception préalable. Ecrivez la classe `EnsembleOrdonne` vue en cours. Les paramètres de généricité sont le type des éléments stockés et une fonction de comparaison de ces éléments. On doit pouvoir ajouter des éléments, en retirer, retourner les plus petits ou les plus grands éléments, etc.