C++ 2A
TD/TP 4 – Généricité, STL

1 Généricité

1.1 Fonctions

1. Créer une fonction `maxi` retournant le plus grand de ses 2 arguments flottants.
2. Transformer cette fonction en une fonction générique `maxi<T>`.
3. Tester votre code sur des types primitifs.
4. Tester sur le type `std::string`.
5. Comment modifier ce comportement pour que la fonction retourne la plus ongue chaîne de caractères ?
6. Proposer 2 solutions pour pouvoir appliquer cette fonction à 2 vecteurs de type `vec3f`, exercice 3.1 de la feuille de TDTP2, et retourner celui dont la norme est la plus grande.

1.2 Types

1. Reprendre la classe `vec3f`, exercice 3.1 de la feuille de TDTP2 et la transformer en une classe générique `vec3<T>`.
2. Tester cette classe avec les types primitifs : **`int`** et **`float`**.
3. Tester cette classe avec le type `complex`, exercice 3.2 de la feuille TDTP2.
4. Surcharger l'opérateur << afin de pouvoir afficher le contenu d'un vecteur `vec3<T>`.

2 STL

2.1 Conteneurs

2.1.1 Vector, Deque, List

1. Que réalise le code suivant ? :

```
vector<int> v1(10);
v1[0] = 5;
for(unsigned int i = 0 ; i < 10 ; ++i) {
  v1.push_back(rand()%100);
}
cout << "Taille␣=␣" << v1.size() << endl << "␣␣";
for(unsigned int i = 0 ; i < v1.size() ; ++i) {
  cout << v1[i] << '␣';
}
cout << endl;
```

2. Tester les différents conteneurs `vectors`, `deques` et `lists` ainsi que leurs méthodes (cf `http://en.cppreference.com/w/cpp/container`).

2.1.2 Map et Multimap

1. Utiliser un conteneur `map` pour stocker le nombre de jours pour chaque mois.
2. Que se passe-t-il pour le mois de février si on souhaite stocker 2 valeurs ? :

```
map<string, int> mois;
...
mois["fevrier"] = 28;
mois["fevrier"] = 29;
...
cout << mois["fevrier"] << endl;
```

3. Procéder de même avec un conteneur `multimap`.

2.1.3 Set et Multiset

1. Utiliser un conteneur `set` pour stocker 50 caractères (remplissage aléatoire).
2. Quelle est la taille du `set` ?
3. Procéder de même avec un conteneur `multiset`.
4. Compter le nombre d'occurences de chaque caractère dans le `multiset`.

2.2 Iterateurs

2.2.1 Affichage

1. Utiliser les itérateurs pour afficher le contenu d'un `vector` et surcharger l'opérateur `<<`.
2. Utiliser le `reverse_iterator` pour afficher dans l'ordre inverse.

2.2.2 Opérations diverses

1. Reprendre l'exercice "Set et Multiset" sur le conteneur `multiset` et supprimer toutes les voyelles.
2. Effectuer l'union des deux `set`.

2.3 Algorithmes

2.3.1 Vecteur, algorithme et foncteur

Définir une fonction `length()` calculant la norme d'un vecteur `vector<float>` à partir de l'algorithme `for_each` et d'un foncteur utilisé comme accumulateur. Essayer également d'utiliser une lambda avec capture de variables externes.

2.3.2 Affichage

1. Utiliser l'algorithme `copy` et l'itérateur `ostream_iterator` pour afficher le contenu d'un conteneur en une ligne de code !
2. Tester sur les différents conteneurs utilisés précédemment.

2.3.3 Tri

1. Utiliser l'algorithme `sort` pour trier un conteneur de type `vector` d'entiers, dans l'ordre croissant, puis décroissant.
2. Trier un conteneur `vector` de vecteurs à composantes flottantes en fonction de leurs normes.

2.3.4 `Vector` de vecteurs

1. Utiliser un algorithme pour remplir un `vector< vector<float> >` avec des vecteurs de même taille contenant des valeurs flottantes aléatoires pour chaque composante, comprises entre `-1.0f` et `+1.0f`.
2. Normaliser chaque vecteur du `vector` en utilisant une fonction, un foncteur, puis une lambda.
3. Calculer la somme de tous les vecteurs du `vector`.