TP7: STL

Exercice 1 : Vector

L'objectif de cet exercice est de créer un programme C++ nommé **vecteur.cpp** qui permet de stocker des **std::string** et manipuler ceux-ci de manière simple.

1. Créer un vecteur nommé **monVecteur** stockant des **std::string**.
2. Ajoutez cinq **std::string** dans le vecteur qui vaudront respectivement **"bonjour", "comment", "allez", "vous", " ?".**
3. Affichez **la taille** de votre vecteur. Afficher sa capacité (**capacity**). Quel est la différence avec sa taille ?
4. Afficher le contenu du vecteur en utilisant la notation indexée de tableau en accédant à la valeur avec une syntaxe du type : **monVecteur[k].**
5. Afficher le contenu du vecteur en utilisant les itérateurs (**iterator**) sur votre vecteur.
6. Réaliser un échange entre le contenu de la case d'indice 1 et le contenu de la case d'indice 3 de votre vecteur (vérifiez votre résultat en affichant le vecteur). Notez l'existence de **std::swap.**
7. Trier le vecteur en utilisant un algorithme de la STL (inclure l'en-tête **algorithm**). L'ordre de tri par défaut est celui de la comparaison sur des **std::string**. Afficher le résultat obtenu.
8. Créer une fonction **affiche** qui affiche le contenu du vecteur passé en paramètre. Chaque élément sera espacé d'un '**espace**' à l'affichage. Notez qu'ici, on passera le vecteur sous forme de **référence constante** car il n'a pas à être modifié, ni copié.
9. Créer une fonction **concatene** qui concatène l'ensemble des éléments du vecteur dans une seule variable de type **std::string**. Chaque élément sera espacé d'un 'espace' dans la **std::string**. Réfléchir au prototype de votre fonction, sous quelle forme vous passez le paramètre d'entrée, sous quelle forme retournez-vous le **std::string** ?
10. Insérer la valeur "à tous" après le premier élément dans votre vecteur. Vérifier votre résultat.

Exercice 2 : List

Créer un fichier source **liste.cpp** qui permettra de :

1. Créer une liste de 30 entiers (les 30 premiers entiers impairs).
2. Afficher votre liste.
3. Supprimer le troisième élément.
4. Afficher à nouveau votre liste.
5. Insérer les 20 entiers impairs suivants.
6. Afficher à nouveau votre liste.

Exercice 3 : Map

Ecrire un fichier **mymap.cpp** qui permettra de construire une **std::map** dont la ***clé*** *est une* ***chaîne de caractère****, et la* ***valeur*** *est un* ***nombre flottant.***

1. Remplir cette map avec un exemple de type "**facture de restaurant**" (nom du plat et prix du plat). Afficher le contenu de la map.
2. Afficher le contenu de la map.
3. Calculer et afficher le prix total des plats présents dans la map.

Exercice 4 : Pile

1. Ecrire une fonction **stack_copy** recevant une pile « **pile** » comme argument et renvoyant une copie « **pile_copie** » de « **pile** ».
2. Ecrire une fonction **stack_reverse** recevant une pile « **pile** » comme argument et renvoyant une copie inversée « **pile_inversee** » de « **pile** ».

Exercice 5 : Pile

Les expressions arithmétiques sont habituellement écrites de manière infixe, c'est à dire que l'opérateur est placé entre ses deux opérandes. Il existe aussi une notation postfixée, que l'on appelle également notation polonaise car introduite par le polonais Lukasiewicz : l'opérateur est placé après ses opérandes.

Exemples :

La notation infixée comme (3 + 5) * 2 => La notation postfixée équivalente est 3 5 + 2 *

La notation infixée comme 3 + (5 * 2) => La notation postfixée équivalente est 3 5 2 * +

- Ecrire un programme c++ qui permet de calculer le résultat d'une expression postfixée saisie par un utilisateur.

Résultats attendus :