CR TME 3

Conteneurs & Itérateurs

Toutes les questions de ce TME demandent de parcourir un tableau de type char** afin de récupérer les différentes chaînes de caractères s'y trouvant et de les insérer dans un Vecteur, une Liste ou une Map.
Le tableau donné se terminant par NULL on peut le parcourir simplement à l'aide d'une boucle while

```
14  #include "GPL_2_text.h"
15  int i = 0;
16  while(GPL_2_text[i] != NULL){
17      //conversion de la chaine de char en string
18      string s(GPL_2_text[i]);
19
20      //Traitement demandé...
21
22
23      i++;
24  }
```

Le conteneur vector<>

backInsert() :

On ajoute les string récupérés à la fin du vecteur à l'aide de la fonction vector::push_back().
On parcourt le vecteur à l'aide d'un itérateur afin d'afficher les éléments.

```
50          //Parcours du vecteur avec un iterateur
51          for(vector<string>::iterator it=vgpl.begin(); it!=vgpl.end(); ++it)
52              cout << *it << ' ';
```

La fonction s'exécute en 0.1 secondes

```
Vector - backInsert: 0.1 secondes ecoulees
```

frontInsert() :

Il n'existe pas de fonction vector::push_front(). Pour insérer simplement un élément en tête de vecteur on utilise les fonctions vector::insert() et vector::begin().

```
60          //Parcours du tableau de char et ajout au début du vecteur
61          while(GPL_2_text[i] != NULL){
62              string s(GPL_2_text[i]);
63              //on insere la string au début du vecteur
64              vgpl.insert(vgpl.begin(),s);
65              i++;
66          }
```

Cette méthode prend cependant plus de temps à exécuter. Il est donc préférable d'utiliser les vecteurs seulement pour insérer de nouveaux éléments à la fin.

```
Vector - frontInsert: 0.37 secondes ecoulees
```

Dans le cas de ce TME, il est cependant possible d'obtenir le même résultat qu'une insertion en tête de vecteur, avant le tri de celui-ci, dans un temps équivalent à une insertion en fin de vecteur.

```
15      while(GPL_2_text[i] != NULL){
16          i++;
17      }
18      for(int j = i-1; j >= 0 ; j--){
19          string s(GPL_2_text[j]);
20          vgpl.push_back(s);
21      }
```

On parcourt simplement le tableau de char depuis la fin et on ajoute les éléments à la fin du vecteur.

```
Vector - frontInsert: 0.1 secondes ecoulees
```

sortEachInsert()

Cette fonction est identique à backInsert() à la différence qu'on ajoute la fonction de tri dans la boucle d'insertion. Le temps d'exécution est beaucoup plus important. Il est donc préférable de trier un vecteur une fois les insertions dans celui ci terminées.

```
Vector - sortEachInsert: 2.843 secondes ecoulees
```

Le conteneur List<>

backInsert() :

Même principe que pour les vecteurs. La classe list<> dispose cependant directement d'une méthode std::sort() plus optimisée. Le temps d'exécution est cependant légèrement plus élevé que pour vector_bench::backInsert()

```
102         //insertion en fin de liste
103         while(GPL_2_text[i] != NULL){
104             string s(GPL_2_text[i]);
105             lgpl.push_back(s);
106             i++;
107         }
```

```
List - backInsert: 0.2 secondes ecoulees
```

frontInsert() :

La classe list<> dispose directement d'une méthode push_front() que l'on utilise de la même manière que push_back(). On obtient le même temps d'exécution que l'insertion en fin de liste.

```
List - frontInsert: 0.2 secondes ecoulees
```

sortEachInsert() :

Même principe que pour vector_bench::sortEachInsert() en utilisant la méthode sort() de la classe list<>. Le temps d'exécution est moins élevé que la méthode équivalente utilisant les vecteurs. Mais il reste préférable de trier sa liste une fois les insertions dans celle ci terminées.

```
List - sortEachInsert: 1.94 secondes ecoulees
```

Conclusions List<> vs Vector<>

Il est plus efficace d'utiliser les vecteurs si l'on veut faire des insertions successives à la fin d'un conteneur

Il est plus efficace d'utiliser les listes si l'on doit insérer des éléments en fin ET en tête de conteneur ou même à une position précise dans celui ci.

Dans les deux cas, il est plus efficace de trier son vecteur ou sa liste une fois les insertions terminées. Cependant s'il est nécessaire de trier notre conteneur après une insertion, il est plus efficace d'utiliser les listes.

Il est aussi possible d'utiliser les deque<> si l'on veut uniquement faire des insertions en tête ET fin de conteneur, ils seront plus efficace que les listes dans ce cas là. Ils seront cependant moins performants que les listes pour des insertions à une position précise dans le conteneur autre que début ou fin.

Le conteneur map<>

Pour effectuer les traitements demandés, on parcourt le tableau de char à l'aide d'une boucle while comme précédemment. Pour chacun des mots on test s'il existe déjà dans la map lors de l'insertion.

```
189         while(GPL_2_text[i] != NULL){
190             string s(GPL_2_text[i]);
191
192             //on test l'existence du mot dans la map
193             //s'il existe on incrémente son compteur de 1
194             //s'il n'existe pas il est inséré dans la map
195             if(mapOfGplWords.insert(make_pair(s,1)).second == false)
196                 mapOfGplWords[s]+=1;
197
198             i++;
199         }
```

S'il existe déjà dans la map, on incrémente la valeur du mot existant, sinon il est inséré.

Pour la suite des traitements on parcourt simplement la map à l'aide d'un itérateur. Pour chacune des entrées, on affiche le mot et le nombre de fois ou il apparaît tout en incrémentant un compteur avec cette dernière valeur afin d'obtenir le nombre de mots du texte à la fin du parcours.

```
202         map<string, int>::iterator it = mapOfGplWords.begin();
203         while(it != mapOfGplWords.end()){
204             cout<< it->first << ':' << it->second << ' ';
205             numberOfWords+=it->second;
206             it++;
207         }
```