TP « introduction »

Ce TP de « prise en main » a pour objectif de vous familiariser avec une chaine de compilation du langage C++ ; d'ajouter une fonctionnalité simple à un programme existant et d'utiliser quelques « *containers* » et fonctionnalités de la STL.

1 - Vérification de la chaine de compilation GCC

A partir de l'archive qui vous sera fournie, assurez-vous que vous avez une chaine de compilation opérationnelle sur votre machine avec GCC. Quelle version de GCC avez-vous ? Utilisez `g++` en ligne de commande, puis utilisez le `Makefile` pour la compilation du programme source `mean_and_median.cpp`. Exécutez le programme et vérifiez que la sortie sur la console est conforme au résultat présenté en cours.

2 - Les `string` et les `vector<>`

Copiez le fichier `mean_and_median.cpp` dans un fichier `histogram.cpp`
Modifiez ce fichier pour afficher en plus de la moyenne et du médian, un histogramme des valeurs comme en page suivante, après lecture du fichier `data_100000.txt`.
N'oubliez pas de modifier aussi le fichier `Makefile` pour la compilation. Assurez-vous que votre programme source est sans erreur en utilisant `cpplint.py` (à installer en utilisant le lien suivant : https://github.com/google/styleguide/raw/gh-pages/cpplint/cpplint.py).

Vous prendrez les hypothèses suivantes :
Toutes les valeurs sont positives comprises entre 0 et 7999.99. Vous ignorerez toutes les valeurs en dehors de cet intervalle.
Vous utiliserez une résolution de 100 en 100 pour le comptage.
Vous afficherez sur 3 colonnes : le « bin » (b), le nombre de valeurs trouvées dans ce « bin », notez qu'une valeur v entre dans le bin b si : $b \leq v < b+100$. Vous afficherez 60 « * » pour la catégorie maximum, le nombre de « * » pour les autres catégories devra être ajusté en proportion.

Essayez votre programme sur les fichiers de données suivants :

- `data_10.txt`
- `data_100.txt`
- `data_1000.txt`
- `data_10000.txt`
- `data_100000.txt`

Voici ce que vous devez obtenir avec le fichier `data_100000.txt`

```
shell> histogram.exe data/data_100000.txt
number of elements = 99957, median = 1715.81, mean = 2124.64
    0       0
  100       0
  200     334 ****
  300     857 **********
  400    1706 **********************
  500    2392 ******************************
  600    3130 ****************************************
  700    3684 ***********************************************
  800    3948 **************************************************
  900    4270 ******************************************************
 1000    4466 ************************************************************
 1100    4432 ********************************************************
 1200    4327 *******************************************************
 1300    4175 *****************************************************
 1400    4031 ***************************************************
 1500    3897 *************************************************
 1600    3739 ***********************************************
 1700    3388 *******************************************
 1800    3334 ******************************************
 1900    3140 ****************************************
 2000    2856 ************************************
 2100    2656 **********************************
 2200    2471 *******************************
 2300    2340 ******************************
 2400    2155 ***************************
 2500    1958 *************************
 2600    1894 ************************
 2700    1693 **********************
...
```

60 étoiles

Pour cet exercice, vous pouvez avoir besoin d'une recherche d'un maximum dans un `vector<>`, vous noterez le code ci-dessous (et ses variantes possibles) ne passe pas une « *code review* ». A vous de proposer autre chose.

```
int  max = 0;
for (int i = 0; i < v.size(); ++i) {
  if (v[i] > max) {
    max = v[i];
  }
}
```

Indice : http://en.cppreference.com/w/cpp/algorithm

3 - Les Tableaux Associatifs

Vous pouvez lire la page suivante : https://en.wikipedia.org/wiki/Hash_table

3.1 « *Unordered* »

Les fichiers `data/full_10*.txt` ont une colonne supplémentaire : la première colonne est un identifiant unique, la seconde colonne est une valeur associée à l'identifiant :

Exemple de fichier :

```
shell> cat data/full_10.txt
fea0536b7a94    2615.93
84fd1c80659c     863.93
8a5b74971f70    1990.52
2ffeabe25cb0    2815.77
44e5db4dbe09    1181.31
179d95de9a47    1321.13
2b3ace2e711d     455.36
911708687b4d     812.47
8789d2ce5b3b    2638.90
bbd9c8b1b456   17301.72
```

On cherche maintenant à écrire un programme qui lit l'un de ces fichiers, puis un « *prompt* » spécifique permet à l'utilisateur de saisir un identifiant et d'afficher immédiatement la valeur associée. Le programme doit afficher un message d'information si l'identifiant n'est pas dans la liste. Le programme termine lorsque l'utilisateur saisie le mot clé END.

Copiez le fichier `mean_and_median.cpp` dans un fichier `map1.cpp`
Modifiez ce fichier pour réaliser le programme demandé.

Résultat attendu :

```
shell> map1 data/full_10.txt
query> 8789d2ce5b3b
value[8789d2ce5b3b]= 2638.9
query> 8a5b74971f70
value[8a5b74971f70]= 1990.52
query> 8a5b74972f70
This ID does not exists
query> END
Bye...
```

Le « *prompt* » spécifique est le mot : `query>`. J'ai mis en italique les identifiants saisis par l'utilisateur (copier/coller) et le chiffre en rouge vous permet de voir clairement la différence entre 2 identifiants : l'un est valide, l'autre pas.

Indices :

1. Pour la lecture de plusieurs champs sur une ligne, inspirez-vous de l'exemple donné http://en.cppreference.com/w/cpp/io/basic_istream/operator_gtgt

2. Vous avez besoin d'un « *container* » décrit dans http://en.cppreference.com/w/cpp/container. A vous de choisir le bon.

3. Pour vous aider, je trouve dans mon programme les lignes suivantes :

```
    ...
    string qin;
    for (;;) {
      std::cout << "query> ";
      std::cin >> qin;
      ...
    }
    std::cout << "Bye..." << std::endl;
}
```

4. Un programme simple doit avoir moins de 50 lignes de code. Hors commentaires et lignes vides. J'utilise la commande suivante pour calculer le nombre de lignes de code :

```
shell> perl -n -e 'if(!/^\s*\/\/*/ && /\S/) {print}' map.cpp | wc -l
41
```

3.2 « *Ordered* »

Amélioriez le programme précèdent pour permettre aussi une recherche inverse : l'utilisateur peut aussi saisir une valeur (v), et le programme doit retourner tous les identifiants dont la valeur associé est comprise entre $v \pm 1\%$. Vous nommerez votre fichier source `map2.cpp`. Faire bien attention au choix de votre nouveau « *container* »: plusieurs identifiants peuvent avoir la même valeur, par exemple on trouve deux fois la valeur 615.25 dans le fichier `data/full_1000.txt`. Pour distinguer facilement les identifiants des valeurs, on impose un signe + devant une valeur.

Exemple d'exécution :

```
shell> map2 data/full_1000.txt
query> 44e2d4b8d7aa
value[44e2d4b8d7aa]= 1358.56
query> +5000
value[375df8b1ac86]= 5022.42
query> +616
value[f6a5f1e9f733]= 612.69
value[7860f4b10a57]= 615.25
value[1201267a89a7]= 615.25
query> END
Bye...
```

Indices :

1. Vous n'avez pas plus de 15 lignes de code supplémentaire à écrire.

```
shell> perl -n -e 'if(!/^\s*\/\/*/ && /\S/) {print}' map2.cpp | wc -l
54
```

2. En cherchant bien à partir du site http://www.cplusplus.com/reference/stl, vous pouvez trouver la solution (presque) complète dans l'un des exemples données en référence ☺.

4 - Livrables pour ce TP

1) Les codes sources `histogram.cpp`, `map1.cpp` et `map2.cpp`

2) Un rapport avec des captures d'écran permettant de vérifier que vos programmes fonctionnent correctement.

3) Une explication sur les « *containers* » utilisés en Section 3.1 & 3.2.

4) Quelle est la complexité d'une « *query* » en fonction de *n*, le nombre de lignes dans le fichier : constante, logarithmique, linéaire, polynomial ? Pourquoi ?

5 - Optionnel

En utilisant les générateurs de nombres pseudo aléatoires de la STL, faire un programme qui génère un fichier avec *n* données dont l'histogramme est similaire à celui des fichiers `data_100000.txt`. Vous pouvez vous aider de http://en.cppreference.com/w/cpp/numeric/random
Quelle est la distribution utilisée ?