Programmation C++ Avancée

Joel Falcou Guillaume Melquiond

19 décembre 2017

L'objectif de ce TP est de réaliser un compresseur (un peu naïf) de texte. Chaque mot sera remplacé par une séquence de caractères et celle-ci sera d'autant plus courte que le mot apparaît souvent dans le texte. Ainsi les mots les plus courants n'occuperont plus qu'un seul caractère.

L'une des structures de données utilisée sera `std::map<key,value>`. C'est une table associative qui, à chaque élément de type `key` associe un élément de type `value`. Parmi les fonctions membres intéressantes, il y a

- `insert(std::pair<key,value> const &)`,
- `erase(iterator)`,
- `begin()`,
- `size()`,
- `operator[](key const &)`.

L'opérateur `[]` offre le même genre d'accès à la table que s'il s'agissait d'un tableau C indexé par des entiers. La fonction `begin` renvoie un itérateur qui, une fois déréférencé, donne une paire `<key,value>`. Cette paire est celle de la table qui a le membre de gauche le plus petit (pour un certain ordre qui dépend du type `key`). L'itérateur renvoyé par `begin` peut aussi être passé à la fonction `erase` pour supprimer la paire en question de la table. La fonction `size` renvoie le nombre de paires présentes dans la table.

Le type `std::multimap<key,value>` propose le même genre d'interface, si ce n'est que plusieurs paires de la table peuvent maintenant posséder le même membre de gauche. L'opérateur `[]` n'a donc plus aucun intérêt mais les autres fonctions restent utilisables.

Aussi bien pour `map` que pour `multimap`, il est possible d'utiliser la syntaxe suivante de boucle pour itérer sur toutes les paires `e` contenues dans la table `m`.

```
for(std::pair<key,value> const &e : m) { ... }
```

1 Lecture de fichier

Récupérez le fichier à l'adresse suivante : `https://www.lri.fr/~melquion/divers/macbeth.txt`

Notez que ce fichier a été nettoyé de toute sa ponctuation et qu'il contient un seul mot par ligne afin de simplifier le travail.

Définissez une fonction `load` qui lit ce fichier et renvoie un vecteur de chaînes de caractères contenant un mot par case. Les fonctions et méthodes suivantes pourront être utiles pour lire le fichier :

- `std::ifstream(char const *)`,
- `std::ios::eof()`,
- `std::getline(std::istream &, std::string &)`.

Testez votre fonction en affichant la longueur du vecteur (le nombre de mots lus) ainsi que le dernier mot du texte.

2 Comptage des occurrences

Le type suivant définit une table qui associe à chaque mot son nombre d'occurrences :

```
typedef std::map<std::string, int> occs;
```

Définissez une fonction `count` qui prend en argument un vecteur de mots et renvoie une table associative qui associe à chaque mot son nombre d'occurrences dans le vecteur.

Testez votre fonction en affichant le nombre d'occurrences des mots "the", "macbeth" et "chestnuts" dans le vecteur renvoyé par la fonction `load`. Affichez aussi le nombre de mots différents présents dans le texte.

3 Création des codes

On associera à chaque mot un code qui est, dans un premier temps, une chaîne de '0' et de '1'. Le type suivant décrit une telle association :

```
typedef std::map<std::string, std::string> codes;
```

Définissez une fonction `merge` qui prend deux arguments de type `codes` et en renvoie un. La table renvoyée doit associer à chaque mot du premier argument son code préfixé par '0' et à chaque mot du second argument son code préfixé par '1'. Par exemple, si les arguments sont les tables $\langle \textit{"baz"} \mapsto \textit{"x"} \rangle$ et $\langle \textit{"bar"} \mapsto \textit{"y"}; \textit{"foo"} \mapsto \textit{"z"} \rangle$, alors la table renvoyée doit être $\langle \textit{"bar"} \mapsto \textit{"1y"}; \textit{"baz"} \mapsto \textit{"0x"}; \textit{"foo"} \mapsto \textit{"1z"} \rangle$.

Le type suivant stocke de façon ordonnée des paires contenant un entier et une table de type `codes` :

```
typedef std::multimap<int, codes> partial_codes;
```

Définissez une fonction `reduce` qui prend un argument de type `partial_codes` et le modifie de la façon suivante. La fonction supprime les deux plus petits éléments et ajoute un nouvel élément à leur place. Ce nouvel élément a pour membre de gauche la somme des membres de gauche des éléments supprimés et il a pour membre de droite le résultat de `merge` appliquée aux membres de droite des éléments supprimés.

Définissez une fonction `create` qui prend un argument de type `occs` et renvoie une table de type `codes` obtenue de la façon suivante. On part d'une table de type `partial_codes` créée de telle sorte que, si l'argument passé à la fonction contient une paire ("foo", 42), alors la table créée contient une paire $(42, \langle \text{"foo"} \mapsto \text{""} \rangle)$. Puis on applique la fonction `reduce` à cette table jusqu'à ce qu'elle ne contienne plus qu'une seule paire. Le membre de droite de cette paire est le résultat de la fonction `create`.

Testez vos fonctions en affichant les codes associés aux mots "the", "macbeth" et "chestnuts".

4 Compression de texte

Définissez une fonction `compress` qui prend un vecteur de mots et une table de type `codes` en argument et crée un nouveau fichier qui contient les mots du vecteur remplacés par leur code. Testez la fonction sur le contenu du fichier initial. Quelle est la taille du fichier obtenu ?

Définissez une fonction `shorten` qui prend une table de type `codes` et en renvoie une autre dont les membres de droite peuvent utiliser d'autres caractères que '0' et '1'. Testez à nouveau sur le fichier initial. Quelle est la taille du fichier obtenu ?

Expliquez comment vous feriez pour décompresser le fichier obtenu.