Rope - Part 1.

Julien Allali : allali@univ-mlv.fr

1 Introduction.

La première partie du sujet consiste en l'écriture d'une classe *Rope* modélisant une chaîne de caractères (comme la classe *string*). La section suivante donne une description de cette classe. La deuxième partie du sujet vous sera remise plus tard.

2 La classe *Rope*.

La classe *Rope* sert à représenter des chaînes de caractères. Son codage doit être effectué comme suit: la chaîne interne à *Rope* est une liste chaînée de tableaux+. Un tableau+ est un tableau de caractères dont le premier indice contient un nombre représentant le nombre de pointeurs pointant sur lui. Le code suivant montre comment on utilise un tableau+:

```
char *tableauPlus = new char [101];
tableauPlus[0]=1;

char *tableauPlus2 = tableauPlus;
tableauPlus2[0]++;

....
ici on utilise tableauPlus+1 et tableauPlus2+1 pour faire des
opérations sur la chaîne.
....
if (tableauPlus[0]==1) // Si il ne reste plus qu'un seul pointeur
   delete tableauPlus;
else
   {
     tableauPlus[0]--; // Sinon, on décrémente le compteur de références.
     tableauPlus=0; // le pointeur n'utilise plus la chaîne.
   }
```

En fait le premier caractère d'un tableau+ est un compteur de références.

Maintenant on va implémenter des chaînes de caractères utilisant ces tableaux+ : la classe *Rope*.

Un exemple de *Rope* est donné sur la figure 1. Sur cette figure, on a une *Rope* qui code pour la chaîne *Bonjour*. Puis on a concaténé la chaîne *monde*. Plutôt que d'allouer un nouveau tableau pour y mettre la chaîne *Bonjour monde*, on utilise les listes chaînées. Ainsi, on a ajouté à la fin de la liste chaînée une nouvelle cellule pointant sur un tableau contenant la chaîne *monde*.

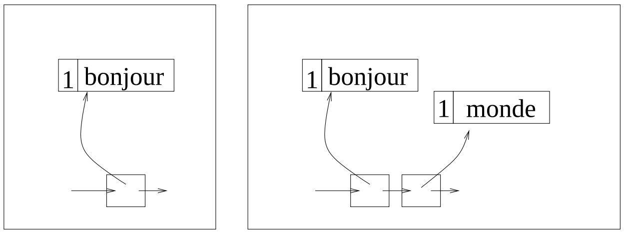

Figure 1: Exemple de la classe *Rope*.

Maintenant supposons que l'on veuille insérer la chaîne *onsieur B* à la 8ème position de la *Rope*. On obtient alors la *Rope* de la figure 2. Si on parcourt la *Rope*, on lit : la première cellule : *bonjour*, la deuxième cellule *m*, puis *onsieur B* et enfin *onde* (soit *bonjour monsieur Bonde*). Au passage, on voit que l'indice du tableau+ *monde* est passé à 2 : en effet, lors de la création de la 4ème cellule, on a placé un nouveau pointeur sur ce tableau et donc on a incrémenté son compteur de références.

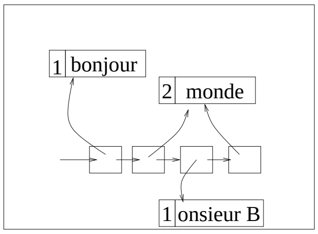

Figure 2: Exemple d'insertion dans la classe *Rope*.

De la même manière, supprimer du texte se fait par des décalages de pointeur et/ou des suppressions de cellules. Sur la figure 3, on a supprimer l'intervalle $[3;8[$. Lors de cette suppression, on a detruit la deuxième cellule. Durant cette destruction, on a décrémenté le compteur de référence du tableau *monde*. Si celui-ci était tombé à 0, alors on aurait détruit le tableau (plus aucune cellule ne pointe dessus).

Ainsi, une cellule contient au moins 4 informations : l'adresse du tableau, un décalage, une taille et un pointeur sur la cellule suivante.

Pour finir, on va se pencher sur le problème de la copie et de l'accès à un élément. Munit de nos tableaux+, la copie devient très facile: on copie juste

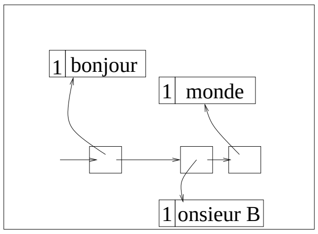

Figure 3: Exemple de deletion dans la classe *Rope*.

la liste chaînée de la *Rope* source et on incrémente les compteurs de références (voir figure 4).

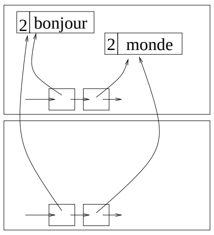

Figure 4: Exemple de copie de deux *Rope*.

Maintenant, que se passe-t-il si l'on souhaite modifier un caractère de la copie. Lors de l'accès (via l'opérateur `[]`), on regarde si le compteur de référence est à 1. Si il est à 1 alors aucune autre cellule ne pointe sur le tableau et on peut le modifier sans problème. Sinon, on créer une copie du tableau (mais uniquement du sous-tableau pointé par la cellule), on décrémente le compteur de référence de l'ancien tableau et enfin on utilise le nouveau tableau (dont le compteur est à 1).

La classe *Rope* doit posséder les méthodes suivantes (à vous de bien placer les **const** et les **références**):

- *Rope* Rope() : construit une *Rope* vide.
- *Rope* Rope(char *str) : construit une *Rope* à partir de str.
- *Rope* Rope(string str) : construit une *Rope* à partir de str.
- *Rope* insert(int position,*Rope* rope) : Insert la rope *rope* avant le caractère d'indice *position* de la *Rope* (0 pour le début). Si $position \geq length()$ l'insertion se fait à la fin. Retourne *this*.
- int length() : Renvoie la longueur courante de la *Rope*.
- double fragmentationValue() : Renvoie la valeur $\frac{tailledelaRope}{nombredecellules}$

- *Rope* remove(int start,int end) : Supprime l'intervalle $[start; end[$ de la *Rope*. Retourne *this*.
- *Rope* replace(*Rope* word,*Rope* value) : Remplace toutes les occurrences de *word* par le mot *value*. Retourne *this*.
- void compact(void) : Compacte la *Rope* : Concatène tous les morceaux dans un seul tableau et libère les sous tableaux. Après un appel à cette méthode la *Rope* contient au plus une cellule et donc $fragmentationValue() = length()$.
- *Rope* getCompactCopie(void) : Retourne une nouvelle *Rope* qui est la version compactée de *this*. (ne modifie pas $this$).
- *Rope* subRope(int start,int end) : Renvoie la sous chaîne $[start; end[$.
- int indexOf(*Rope* motif,int from=0, int to=length()) : Renvoie l'indice de la première occurrence de $motif$ dans la sous chaîne $[from; to[$. Renvoie -1 si aucune occurrence n'est trouvée.

De plus les opérateurs suivant devront être implémenté:

- `[]` : permet d'accéder a un caractère particulier.
- `<<` : écriture sur un flux de type ostream.
- `+` : créer une nouvelle *Rope* qui est la concaténation de 2 *Ropes*.
- `==` : teste l'égalité de deux *Rope*.
- `!=` : teste la différence de deux *Rope*.
- `<`: compare deux *Rope* selon l'ordre lexicographique.
- `<=`: compare deux *Rope* selon l'ordre lexicographique.
- `>`: compare deux *Rope* selon l'ordre lexicographique.
- `>=`: compare deux *Rope* selon l'ordre lexicographique.
- L'opérateur de conversion en *string*.

Bien évidemment, les fonctions classiques seront implémentées (pour la copie par exemple).

3 Ce qu'il faut faire

Pour bien commencer il faut : **Lire le sujet**. Ce projet est à faire par groupe de **2**. Ensuite il faut bien réfléchir au problème d'allocation (c'est là la difficulté du sujet). N'oublier pas de bien replacer les **références** et les **const** pour chaque méthode.

La classe *Rope* devra être implémentée dans deux fichiers (Rope.hpp et Rope.cpp), et devra faire partie de l'espace de nom *umlv*.

Un teste pour votre implémentation est disponible sur ma page Web.

Pour s'assurer que votre programme ne fait pas de *bêtise*, tester le avec *valgrind* (c.f. man).

Suite au prochain numéro...