TP 1
CONSTRUCTEUR DE CLASSE D'ÉQUIVALENCE

Introduction

Pour le premier devoir, vous allez construire un programme qui construit une représentation de classe d'équivalence. Cette représentation sera efficace pour la construction des classes et pour leurs consultations. Votre logiciel permettra :

- Construction :
  - L'ajout d'élément.
  - L'ajout d'élément dans une classe d'équivalence.
  - L'union de deux classes existantes.
- De répondre à des requêtes :
  - Vérifier que deux éléments sont présentement dans la même classe d'équivalence.
  - Donner le représentant courant d'une classe d'équivalence.
  - Donner le représentant minimal commun de deux éléments.

Le logiciel va recevoir ses entrées au clavier et placer les sorties à l'écran.

Description des entrées

Au démarrage, le logiciel doit consulter sa liste d'argument afin de vérifier si ''-h'' est un argument. Si c'est le cas, alors le logiciel devra utiliser une d'optimisation pour la représentation des classes d'équivalence. Cette optimisation est présentée dans la description de la structure de donnée plus bas.

Le logiciel va lire les entrées au clavier (`cin`) et traiter l'information. Une commande peut contenir de l'information à ajouter dans la base de données ou une requête. Le logiciel termine lorsque le caractère <ctrl-d> est entré (fin de fichier en Linux). Voici les différentes commandes possibles. Il est à remarquer qu'une commande peut être écrite sur plusieurs lignes.

- $id.$
  Cette commande indique que l'identificateur id est une classe d'équivalence. Cette information doit être ajoutée dans la base de connaissances. La commande se termine par un point.
- id_0 rep $id_1, \ldots, id_n.$

Cette commande indique que les identificateurs id_1 , … , id_n sont représentés par la classe d'équivalence de l'identificateur id_0 . Cette information doit être ajoutée dans la base de connaissances. Il est possible que la liste ne contienne qu'un seul identificateur. La commande se termine par un point. S'il y a plus d'un identificateur, alors ils sont séparés par des virgules.

- id_0 ce id_1.
  Cette commande indique que les deux identificateurs ont la même classe d'équivalence. Alors, les deux classes doivent être jointes. Cette commande se termine par un point.
- id_0 ce id_1?
  Cette commande est une requête, car elle se termine par un point d'interrogation. Vous devez vérifier que les deux identificateurs (id_0 et id_1) sont dans la même classe d'équivalence.
- rep id_1?
  Cette commande est une requête. Elle demande qui est le représentant courant de la classe d'équivalence de l'identificateur id_1. Cette commande se termine par un point d'interrogation.
- rep id_1, id_2 ?
  Cette commande est une requête. Elle demande qui est le représentant courant le plus profond (le plus près) des identificateurs id_1 et id_2. Cette commande se termine par un point d'interrogation.

Un identificateur est une suite de caractères alphabétiques. Ils peuvent contenir des lettres minuscules et majuscules. Les identificateurs et les mots 'ce' et 'rep' doivent être séparés par au moins un espace, mais peuvent en contenir plus. Ils peuvent aussi être séparés par des fins de lignes. Les caractères de ponctuation (le point '.', la virgule ',' et le point d'interrogation '?') peuvent être précédés ou suivit d'espaces, mais ce n'est pas obligatoire. Les mots 'ce' et 'rep' ne peuvent être utilisés comme identificateur. Après un point ou un point d'interrogation, il ne peut avoir seulement des espaces et une fin de ligne. Donc, une commande peut être sur plus d'une ligne, mais il n'y a pas plus d'une commande par ligne. Si une erreur d'entrée se produit, votre logiciel termine avec le message d'erreur « Entrees non conforme. » sur le canal `cerr` et utilise `exit(-1)` pour quitter. Les commandes sont exécutées à mesure qu'elles sont entrées.

Traitement

Vous devez conserver l'information entrée en mémoire. Pour cela vous allez construire deux structures. Une première structure interne va contenir l'information indiquant à quelle classe d'équivalence appartient un identificateur (Structure pour les classes d'équivalence). La deuxième structure va lier chaque chaîne de caractère à la position de l'identificateur qu'elle représente dans la structure interne (Structure de lien, ou d'accès). Cette structure va permettre de retrouver rapidement l'instance d'un identificateur en mémoire.

Structure pour les classes d'équivalence

Cette structure est construite à l'aide d'un élément de base (classe `Element`). Ces éléments représentent un identificateur et un lien vers le représentant supérieur (`suivant`), qui est aussi un `Element`. Cette structure est similaire au Maillon (Chainon) qui est utilisé pour la construction de liste chaînée. Elle doit aussi contenir un champ indiquant le rang d'un identificateur. À la création, un `Element` a un rang de 0 et le pointeur `suivant` pointe sur `nullptr`.

```
class Element {
protected:
    string identificateur;
    int rang;
    Element * suivant;
};
```

Pour les trois premières commandes (commande qui ajoute de l'information à la base de données), tout identificateur qui n'existe pas doit être construit. Il y aura donc un Element pour chaque identificateur. Chaque construction d'identificateur impliquera un ajout dans la structure de lien (voir plus loin).

Vous allez ajouter à cette classe une méthode `Element * representant()` qui retourne le représentant de la classe d'équivalence à laquelle appartient un élément. Pour cela, vous suivez les pointeurs suivant jusqu'à ce que vous trouviez `nullptr`. Un pointeur sur le dernier `Element` de la chaîne (celui contenant `nullptr`) est retourné par la méthode.

La première commande ($id.$) ne fais que construire un `Element`.

Pour la deuxième commande (id_0 rep id_1, … , id_n.), les identificateurs qui sont listés à droite du mot 'rep' devront être reliés à l'identificateur à gauche du mot 'rep'. Pour indiquer que l'identificateur id_0 représente l'identificateur id_1, il suffit de trouver l'instance d'`Element` qui représente l'identificateur id_1 et de faire pointer son champ `representant` vers l'`Element` représentant l'identificateur id_0. Si le rang de id_0 est plus petit ou égal au rang de id_1, alors le rang de id_0 est placé égal au rang de id_1 + 1. Refaire ces opérations pour tous les identificateurs à droite de 'rep' dans la commande.

Pour la troisième commande (id_0 ce id_1.), vous devez commencer par trouver les représentants que nous dénoterons E_0 et E_1 (méthode `representant`). Il faut trouver lequel des éléments (E_0 ou E_1) a le rang le plus petit. Ensuite, le représentant ayant le plus petit rang devra référer au représentant ayant le rang le plus grand. Ainsi, les deux classes seront unies en une seule classe d'équivalence. Si les deux représentants ont le même rang, alors E_1 utilisera E_0 comme représentant et le rang de E_0 recevra un incrément de 1.

Structure de lien

Cette structure va permettre de retrouver rapidement l'`Element` qui contient l'instance pour un identificateur à partir d'une chaîne de caractères. Pour cela, nous allons utiliser une classe de la `stl` : `map`.

Il suffit de construire une instance de `map<string, Element *>`. Lors de la construction d'un nouvel `Element`, vous devrez insérer l'élément et la chaîne de caractères le représentant dans la structure (opérateur `[]`). Et lorsque vous voulez retrouver l'élément lié à une chaîne de caractères, vous pouvez utiliser la méthode `find`. Je vous laisse consulter la documentation sur internet (ou autre source).

Commandes de requête

Pour les trois commandes de requête, il suffit de consulter la base de connaissance (les deux structures) pour trouver la réponse.

Pour la première requête (id_0 ce id_1?), vous devez trouver les représentants des classes d'équivalence des identificateurs id_0 et id_1 (méthode `representant`). Si les deux représentants sont différents, alors vous affichez 'non' sur 'cout', sinon, vous affichez 'oui' sur 'cout'. Cette requête vérifie donc si deux identificateurs sont dans la même classe d'équivalence.

Pour la deuxième requête (rep id?), vous devez simplement trouver le représentant de l'identificateur id. Ensuite vous affichez l'identificateur du représentant sur 'cout'.

Pour la dernière requête (rep id_1, id_2 ?), vous devez suivre les chaînes 'suivant' des deux éléments id_1 et id_2. Il faut trouver où les deux chaînes se croisent et retourner le premier élément commun trouvé. Je vous laisse le soin de construire un algorithme performant pour cette tâche.

Pour les trois commandes de requête, si un des identificateurs n'existe pas dans la structure de lien, alors le message « Identificateur inexistant. » est affiché sur '`cerr`'. Le programme continue de lire les commandes au clavier.

Optimisation (argument -h)

Lorsque l'argument -h est sur la ligne de commande, votre logiciel devra utiliser une optimisation pour la méthode `representant`. Cette optimisation est très simple, lorsque la méthode parcourt la liste des `suivants` pour trouver le dernier élément de la liste, elle doit ensuite faire pointer les champs `suivants` de tous les éléments rencontrés vers le dernier élément de la liste. Ainsi, à la prochaine demande d'un représentant, l'élément sera retrouvé en un seul tour de boucle.

Exemple d'exécution

cin (entrées)	cout (sorties)
Scolopacidae. Scolopacidae rep Actitis. Actitis rep Hypoleucos, Macularia. ce Macularia ? Calidris. Calidris rep Alba. Hypoleucos ce Macularia ? Hypoleucos ce Alba ? rep Hypoleucos, Macularia ? Scolopacidae rep Calidris, Arenaria. Hypoleucos ce Alba ? Alpina. Alba ce Alpina ? Alba ce Alpina. Alba ce Alpina ?	 Scolopacidae oui non Actitis oui non oui

Directives

1. Le TP est à faire seul ou en équipe de deux.
2. Commentaire :
    a. Commentez l'entête de chaque fonction. Ces commentaires doivent contenir la description de la fonction et le rôle de ces paramètres.
    b. Une ligne contient soit un commentaire, soit du code, pas les deux.
    c. Utilisez des noms d'identificateur significatifs.
    d. Utilisez le français.
3. Code :
    a. Pas de `goto`.
    b. Un seul `return` par méthode.
4. Indentez votre code. Assurez-vous que l'indentation est faite avec des espaces.

Remise

Remettre le TP par l'entremise de Moodle. Remettez votre fichier `'nomEtudiantTP1.cpp'`, nous utiliserons un seul fichier de code pour le premier TP. Le TP est à remettre avant le 15 février 23:55.

Évaluation

- Fonctionnalité (10 pts) : Votre TP doit compiler sans erreur (il peut y avoir des warnings). J'utilise la ligne de compilation suivante : `g++ nomTp.cpp -lm -std=c++11` sur les serveurs Linux de l'université.
    - 1 point de la fonctionnalité sera attribué à la performance de la sixième commande. Les équipes ayant la meilleure performance auront 1 pt, les autres auront des dixièmes de point dépendant de leur performance relative. Un minima de 0.1 pt est attribué aussitôt que la commande donne la bonne réponse.
- Structure (2 pt) : Il faut avoir **plusieurs** fonctions. Construisez un code bien structuré.
- Lisibilité (3 pts) : commentaire, indentation et noms d'identificateur significatif.