École Centrale de Nantes

Algorithmique et Programmation

TP4
Listes et Arbres

FICHIERS À RENDRE DANS L'ARCHIVE :

- `type_def.h`
- `TP4.cpp`
- `utilitaires.cpp` et `utilitaires.h`
- `repertoire.cpp` et `repertoire.h`
- `rapport_TP4.pdf`
- `repertoire.txt`

1 Introduction

Les structures linéaires et arborescentes sont essentielles à l'organisation de l'information. Ce TP est l'occasion de créer et de manipuler concrètement ces structures de données et de comparer les temps d'exécution des opérations de base, à savoir l'ajout, la recherche, la suppression ou l'affichage d'éléments. Par ailleurs, vous serez aussi amenés à utiliser le principe de la récursivité pour parcourir, modifier ou afficher vos structures de données.
Le thème du TP est l'organisation d'informations relatives à des personnes à travers la gestion d'un répertoire contenant des éléments de type `personne`, lui même contenant les informations relatives à une personne. Ce répertoire est représenté avec trois structures de données différentes : tableau trié, liste triée, arbre binaire de recherche. Toutes ces structures de données sont ordonnées d'abord selon le nom de la personne, puis son prénom et enfin son numéro de sécurité sociale.

2 Structures de données

Définissez les structures suivantes et implémentez-les dans un fichier `type_def.h` :

Question 1. `personne` contenant le nom (chaine de caractère), le prénom (chaine de caractère), le numéro de sécurité sociale (13 chiffres dans une chaine de caractère sans espaces).

Question 2. `vectPersonne` tableau statique de 10000 `personnes`.

Question 3. `elementListe` un enregistrement contenant une `personne` et les pointeurs permettant de créer une liste doublement chainée (suivant, précédent).

Question 4. `nœud` un enregistrement contenant une `personne` et les pointeurs permettant de créer un arbre binaire (filsDroit, filsGauche).

3 Utilitaires

Définissez les fonctions suivantes et implémentez-les dans le fichier `utilitaires.cpp`. Vous testerez vos fonctions avec un programme principal (`main`) dans un fichier `TP4.cpp`.

Question 5. `initTabNomPrenom` : Lecture des fichiers `Prenoms.txt` et `Noms.txt` et stockage dans des tableaux passés en paramètres.

Question 6. `genererRepertoire` : Génération aléatoire d'un répertoire de 10000 personnes. Cette fonction utilise les tableaux contenant les noms et les prénoms, ainsi que la fonction `genererPersonne` pour créer une personne aléatoirement, puis écrit cette personne dans le fichier `repertoire.txt`.

Question 7. `egalitePersonne` : Détermine si deux personnes ont les mêmes informations, nom, prénom et numéro de sécurité sociale. Cette fonction retourne un booléen : vrai en cas d'égalité, faux dans l'autre cas.

Question 8. `comparerPersonne` : Définition d'un ordre sur les personnes (qui sera utilisé pour ordonner les structures de données). Cette comparaison se fait d'abord selon le nom de la personne, puis son prénom et enfin son numéro de sécurité sociale. On considère que les deux personnes ne sont pas identiques en entrée de la fonction. Cette fonction retourne un booléen : vrai si la première personne est avant l'autre, faux dans l'autre cas.
Indice pour les Q7 et Q8 : Les `string` peuvent se comparer avec les opérateurs de comparaison standard ($==, !=, <, <=, >, >=$)

4 Création et utilisation d'un répertoire

Définissez les fonctions suivantes et implémentez-les dans un fichier `repertoire.cpp` pour les trois structures de données : **tableau trié, liste triée et arbre binaire de recherche**. Pour les listes et les arbres, vous devez faire des fonctions récursives, et utiliser les pointeurs (en paramètre et/ou en résultat) dans les spécifications de certaines fonctions.

Question 9. `afficher` : Affichage de la structure de données dans l'ordre.

Question 10. `ajouter` : Ajout d'une personne au bon endroit dans la structure de données en utilisant la fonction de comparaison. Si on trouve une personne identique à une autre déjà présente dans la structure de données, la personne n'est pas ajoutée. Pour les listes et les arbres, ces fonctions renvoient un pointeur vers la tête de la nouvelle liste ou vers la racine du nouvel arbre, respectivement.

Question 11. `rechercher` : Recherche d'une personne dans la structure de données à partir du nom, prénom et numéro de sécurité sociale. Ces fonctions renvoient vrai ou faux.

Question 12. `supprimer` : Suppression d'une personne dans la structure de donnée à partir du nom, prénom et numéro de sécurité sociale. Pour les listes et les arbres, ces fonctions renvoient un pointeur vers la tête de la nouvelle liste ou vers la racine du nouvel arbre, respectivement.

Question 13. `lectureRepertoire` : Lecture des informations du fichier `repertoire.txt` pour créer la structure de donnée triée.

5 Tests et rapport

— Comparez les temps d'exécution des trois fonctions `lectureRepertoire`.
— Comparez les temps d'exécution de l'affichage des trois structures.
— Comparez les temps d'exécution de la recherche de 1000 personnes dans les trois structures.
— Comparez les temps d'exécution de la suppression de 1000 personnes dans les trois structures.

Les temps d'exécution peuvent être comparé avec la librairie `ctime`, en particulier avec la fonction `clock` : `http://www.cplusplus.com/reference/ctime/clock/`

Vous mettrez dans le rapport toutes les structures de données, les **spécifications** des fonctions des questions 5, 6, 7 et 8, ainsi que des fonctions des questions 9, 10, 11, 12, 13 pour les tableaux et les listes, **et les algorithmes des fonctions des questions 9, 10, 11, 12, 13 pour les arbres binaires de recherche**. Vous ne devez pas mettre de code C++ dans le rapport, uniquement du pseudo-code.
Vous indiquerez les résultats des tests et commenterez le comportement de vos fonctions dans les cas normaux et les cas limites.