TP3 Pointeurs et listes chainées

Exercice 1 – Tri

On souhaite disposer d'un ensemble de données sur des individus (nom, prénom, date de naissance, …) et de pouvoir les trier suivant différents critères sans dupliquer l'ensemble des informations, ni effectuer plusieurs fois les mêmes tris. Un ensemble d'individus sera représenté par une structure composée d'un tableau d'individus et d'un entier (nombre d'individus). Un ensemble de références sera une structure similaire, avec un tableau de pointeurs vers des individus à la place d'un tableau d'individus.

1. Définir les types permettant de manipuler des individus, des ensembles d'individus, et des ensembles de références.
2. Écrire un sous-programme qui construit un ensemble de références à partir d'un ensemble d'individus, selon l'ordre initial des données.
3. Écrire un sous-programme qui réalise le même traitement en sélectionnant uniquement le sous-ensemble des individus dont l'année de naissance correspond à celle donnée en paramètre.
4. Écrire un sous-programme qui trie des personnes (en modifiant uniquement un ensemble de références) selon l'ordre lexicographique des noms de l'ensemble d'individus pointés par l'ensemble de références.
5. Même question pour les dates de naissance.
6. Compléter le programme (procédures de saisie et d'affichage, programme principal) afin d'effectuer quelques tests.

Exercice 2 – Répertoire

On veut écrire un programme permettant de manipuler un répertoire de personnes en mémorisant, pour chacune des personnes, le nom, le prénom et le numéro de téléphone. Nous représenterons en mémoire un répertoire par une liste chaînée, et pour cela, on pourra utiliser les structures de données suivantes :

```
struct personne {
   std::string nom, prenom, tel;
};

struct element {
    personne val;
    element *suiv;
};
using repertoire = element *;
```

Écrire les sous-programmes suivants :

1. void initialiserRepertoire (repertoire & R)
2. void ajouterEnTete (std::string nom, std::string prenom, std::string tel, repertoire & R)
3. void ajouterEnQueue (std::string nom, std::string prenom, std::string tel, repertoire & R)
4. void afficherPersonne (personne P) : affichage des informations d'une personne sous la forme [nom, prénom, téléphone]
5. void afficherRepertoire (repertoire R)
6. std::string telephone (std::string nom, std::string prenom, repertoire R) : recherche la première occurrence d'une personne selon son nom et son prénom, puis retourne son numéro de téléphone (ou la chaine vide si cette personne n'est pas dans le répertoire)
7. int rechercherPosition (std::string nom, std::string prenom, repertoire R) : recherche la première occurrence d'une personne selon son nom et son prénom, puis retourne sa position dans le répertoire (la 1ère personne a pour position 1 ; 0 sera retourné si cette personne n'est pas dans le répertoire).
8. void ajouter (int position, std::string nom, std::string prenom, std::string tel, repertoire & R) : ajoute une personne à une position donnée dans le répertoire (si la position est strictement supérieure à taille du répertoire, alors la personne est ajoutée en fin de répertoire).
9. void supprimer (int position, repertoire & R) : supprime la personne à la position donnée (si la position est strictement supérieure à taille du répertoire, alors on ne fait rien).
10. void supprimer (std::string nom, repertoire & R) : supprime du repertoire toutes les personnes portant ce nom.
11. void trier (repertoire & R) : trie un répertoire selon le classement alphabétique des noms.

Exercice 3 – Dominos

On veut écrire des fonctions et procédures permettant de représenter un jeu de dominos et de faire jouer la machine automatiquement. Un domino est composé de deux parties, chacune de ces deux parties contient un nombre de points (de 0 à 6 points). Un domino sera donc représenté par un couple d'entiers. Une main est constituée d'un certain nombre de dominos, et représente l'ensemble des dominos d'un joueur.

1. Définir le type (structure) domino permettant de représenter un domino.
2. Définir le type liste, permettant de représenter une main comme une liste chaînée de dominos.
3. Écrire une procédure de saisie d'un domino. La procédure demandera à l'utilisateur de saisir les nombres de points des deux parties du domino.
4. Écrire une procédure permettant d'initialiser une liste de dominos (à une liste ne contenant aucun domino).
5. Écrire une procédure qui ajoute à gauche (au début) le domino d dans la liste (main) m.
6. Écrire une procédure qui ajoute à droite (à la fin) le domino d dans la liste (main) m.
7. À l'aide des deux procédures précédentes, écrire une procédure de saisie d'une main (demander en premier le nombre de dominos à saisir).

8. Écrire un sous-programme d'affichage d'un domino (ex : 2:3).
9. Écrire un sous-programme d'affichage d'une main. Voici un exemple d'affichage à obtenir :
   4:0 – 1:6 – 2:2 – 2:3
10. Écrire la procédure qui supprime le premier domino d'une main m.
11. Écrire une fonction qui retourne le nombre de domino contenus dans une main.

On veut pouvoir gérer la pioche. La pioche sera en fait représentée par une autre liste.

12. Écrire une procédure qui construit la pioche initiale avec l'ensemble des 28 dominos possibles.
13. Écrire une procédure qui retire le n-ième domino de la pioche, c'est-à-dire qui le supprime après avoir mémorisé ses valeurs dans un paramètre.
14. Écrire une procédure "piocher", qui choisit au hasard un domino dans la pioche et l'ajoute dans une main.
15. Écrire une procédure qui génère une main m avec un nombre n de dominos de la pioche p (tirés aléatoirement).
16. Écrire un sous-programme qui calcule le nombre de points contenus dans une main.
17. Écrire une procédure "retourner" qui échange les valeurs d'un domino (pour le retourner).

Le jeu (domino joués sur la table), sera également représenté par une liste.

18. Écrire les fonctions "aGauche" et "aDroite" retournant la valeur du domino situé à l'extrémité gauche (respectivement droite) d'une liste de dominos, donc au début (respectivement à la fin).

Pour pouvoir être joué, un domino doit avoir une face qui correspond à l'extrémité gauche ou droite des dominos déjà joués. Par exemple, si on a déjà joué « 2:3 – 3:6 – 6:6 – 6:4 », on pourra jouer un domino contenant un 2 ou un 4.

19. À l'aide des sous-programmes précédents, écrire une procédure jouer (liste& m, liste& j, bool& bloque) qui joue le premier domino de la main m jouable (en fonction du jeu j), lorsque cela est possible. La variable bloque vaudra vrai uniquement si aucun coup jouable n'a été trouvé.
20. Écrire un programme principal qui initialise le jeu, demande le nombre de dominos à piocher au départ, puis joue des dominos lorsque cela est possible, et pioche lorsque le joueur est bloqué. Le jeu s'arrête si l'on est bloqué et que la pioche est vide (on affiche alors le nombre de points restants en main), ou bien si le joueur arrive à vider sa main (il a alors gagné).

Question supplémentaire

Modifier le programme afin de faire jouer 1 puis 2 joueurs au jeu de dominos.