TP Semaines 12 et 13
Manipulation de Listes (suite)

Comme pour le TP précédent, nous allons manipuler des listes d'Etudiants. Créez donc chez vous un répertoire de travail pour ce TP, puis copiez tous les fichiers du répertoire `/net/Bibliotheque/AP2/TP_par_Semaine/Semaine12_13` dans ce répertoire.

Ouvrir/Enregistrer

Exercice 39 : Création d'une liste à partir d'un fichier

1. Ajoutez dans la classe `Etudiant` une méthode `lireFlux` comportant un paramètre `fstream`.

2. Ajoutez maintenant dans les fichiers `gestion` (`.h` et `.cc`) une fonction `fic2Liste` qui construit une liste d'étudiants à partir d'un fichier. Vous disposez d'un fichier `liste1` pour tester votre fonction. Votre `main` peut ressembler à ceci :

```
int main(){
  Liste<Etudiant> l;
  fic2Liste(``liste1'', l);
  afficherListe(l);
}
```

Exercice 40 : Sauvegarde d'une liste

1. Ajoutez dans la classe `Etudiant` une méthode `ecrireFlux`.

2. Ecrivez une fonction `liste2Fic` qui sauvegarde une liste dans un fichier.

Tri de listes

Nous allons maintenant implémenter le *tri fusion*. Le tri se fera sur les noms des étudiants, par ordre croissant. Le principe du tri fusion est simple : on partage la liste en deux sous-listes consécutives, on trie ces sous-listes, et on les fusionne. Dans toute la suite, il sera question de sous-listes ; nous adopterons la convention suivante : si `l` est une liste, `a` et `b` deux TAdresse dans cette liste, la sous liste `(l, a, b)` est l'intervalle *ouvert* `]a, b[`.

L'algorithme s'écrit naturellement de façon récursive :

```
tri(ES l : Liste, E  a, b : TAdresse)
var m : TAdresse
debut
  Si (l, a, b) possede au moins deux elements
  Alors debut
        m <-  milieu(l, a, b)
        tri(l,a,adresseSuivant(l,m))
        m <-  milieu(l, a, b) // le tri de la premieere moitie
                              // a pu changer l'adresse de l'objet median
        tri(l, m,b)
        fusion(l,m,a,b)
        fin
fin
```

Exercice 41 : recherche du milieu

1. Créez deux nouveaux fichiers `tri.h` et `tri.cc`, et modifiez le `Makefile` en conséquence.

2. Ecrivez une fonction qui, étant donnée une sous liste `(l, a, b)`, retourne l'adresse de son élément médian ; on conviendra que l'élément médian d'une sous-liste ayant $2k$ éléments est le k^e, tandis que c'est le $(k+1)^e$ dans le cas d'une sous-liste de longueur $2k+1$. De plus, cette fonction ne sera appelée que sur des sous-listes ayant au moins deux éléments. Enfin, la première borne définissant la sous-liste ne sera jamais NULL ; si l'on souhaite connaître le milieu d'une liste entière, on insèrera d'abord en tête un élément fictif, puis on calculera le milieu sur
   `(l, adressePremier(l), NULL)`, et on n'oubliera pas ensuite de supprimer en tête.

   Exemples avec une liste d'entiers :

   `l= (1, 7, 5, 14, 2, 3, 8)`

   supposons que les adresses soient : `a, b, c, d, e, f, g, NULL` ; donc
   `adressePremier(l)` vaut `a`, `valeurElement(l, f)` vaut `3`, etc... dans ce cas, l'adresse de l'élément médian de `(l, b, g)` est `d` ; de même, l'adresse de l'élément médian de `(l, d, NULL)` est `f`. Insérons 0 en tête, supposons que `adressePremier(l)` vaille maintenant `x` ; alors `milieu(l, x, NULL)` vaut `d`.

   Pour programmer cette fonction, vous ne devez pas compter les éléments de la sous-liste, mais utiliser deux TAdresse dont l'un progressera deux fois plus que l'autre.

   Testez votre fonction, en faisant afficher l'élément médian de quelques sous-listes.

Exercice 42 : fusion Ecrivez une fonction qui, étant données deux sous-listes consécutives déjà triées `(l, a, adresseSuivant(m))` et `(l, m, b)` fusionne ces deux sous-listes. A la fin de la fonction,
`(l, a, b)` est triée. **Vous ne devez pas utiliser de structure intermédiaire (liste ou autre).**

Testez votre fonction à l'aide du fichier `liste2` contenant une liste dont les deux "moitiés" sont déjà triées.

Exercice 43 : tri

1. Vous pouvez maintenant écrire la fonction `triFusion` qui trie récursivement un intervalle ouvert. Testez votre fonction sur les listes disponibles.
2. Ecrivez une fonction `tri` qui trie une liste en utilisant la fonction `triFusion`. Testez.

Tri indirect

A la fin du module AP1, vous avez entendu parler du tri indirect. Il s'agissait de trier des indices au lieu des éléments eux-mêmes ; dans une liste, il n'y a pas d'indices, mais des `TAdresse`. Nous allons donc trier des `TAdresse`, bien sûr toujours en fonction des valeurs des éléments auxquels ils renvoient. L'intérêt de cette démarche est double :

- on n'effectue les échanges que sur les `TAdresse`, ce qui optimise le tri lorsque les éléments de la liste sont des objets de taille importante.
- on peut, sur une même liste, effectuer plusieurs indexations.

Exercice 44 : création d'une liste de TAdresse Ecrivez une fonction `creerIndex` qui crée la liste des `TAdresse` d'une liste donnée. Les adresses de cette liste seront du type `TAdresse2` (défini dans `tri.h`), et les valeurs seront donc du type `TAdresse`.

Exercice 45 : affichage indirect Ecrivez une fonction qui affiche les éléments d'une liste en utilisant une liste d'index (liste de `TAdresse`).

Exercice 46 : tri indirect selon les notes

1. Ecrivez une fonction `milieuIndex` analogue à `milieu` (cf. ci-dessus).
2. Ecrivez `fusionIndirectNotes`
3. De même, écrivez `triFusionIndirectNotes` et enfin `triIndirectNotes`

Exercice 47 : tri fusion - encore !

Une des bases du tri fusion est le découpage de la liste (ou tableau) en deux sous-listes de même taille (ou presque). Cette opération peut être réaliser de la manière suivante :

```
Action decoupage (E L: TListe, S laurel : Tliste, S hardy : TListe)
Var flipflop : booleen
Debut
  flipflop <- vrai
  adr <- AdressePremier(L)

  Tant Que adr <> NULL
```

```
  Faire Debut
      Si flipflop = VRAI Alors
          InsererEnTeete(laurel,valeurEleement(L,adr))
      Sinon
          InsererEnTete(hardy,valeurElement(L,adr))
      flipflop <- non flipflop
      adr <- AdresseSuivant(L,adr)
  Fin
Fin
```

Implémenter le tri fusion en vous basant sur cette idée.

Une autre idée d'utilisation de liste

Reprenez la correction du TP sur les formes géométriques. Pour stocker les formes, cette version utilise un tableau statique de taille maximum.

Supprimez cette limitation en remplaÃ§ant ce tableau statique par une liste.