TP Semaine 11
Manipulation de Listes

Le but de ce TP est de vous faire manipuler les listes. Vous ne serez que des utilisateurs de cette classe. Nous verrons plus tard comment elle peut être implémentée. Notez que cette pratique est rendue possible grâce à la compilation séparée.

Pour utiliser la classe liste, vous devez donc dans un premier temps récupérer son fichier entête et son fichier source. Créez chez vous un répertoire de travail pour ce TP, puis copiez tous les fichiers du répertoire `/net/Bibliotheque/AP2/TP_par_Semaine/Semaine11` dans ce répertoire.

Les listes en C++

Nous souhaitons développer un module de gestion des notes des élèves. Ainsi, une classe `Etudiant` vous est proposée (fichiers `Etudiant.cc` et `Etudiant.h`). Chaque `Etudiant` est formé d'un nom (`string`) et d'une note (`float`). Pour la suite du TP, vous pourrez éventuellement rajouter des méthodes à cette classe.

On vous fournit une *classe générique* `Liste` vous permettant de manipuler des listes d'objets de type quelconque. Le prototype de la classe `Liste` est dans `Liste.h`. Son implémentation est dans `Liste.cxx`.

On remarque qu'on utilise la notation objet au lieu de la notation fonctionnelle (voir le schéma suivant).

ASD	C++
var l : liste d'etudiants	Liste<Etudiant> l;
adr ← AdressePremier(l)	adr = l.adressePremier()
adr2 ← AdresseSuivant(l, adr)	adr2 = l.adresseSuivant(adr)
val ← ValeurElément(l, adr)	val = l.valeurElement(adr)
ModifieValeur(l, adr, val)	l.modifieValeur(adr, val)
InsérerEnTete(l, elem)	l.insererEnTete(elem)
InsérerAprès(l, elem, adr)	l.insererApres(elem, adr)
SupprimerEnTete(l)	l.supprimerEnTete()
SupprimerAprès(l, adr)	l.supprimerApres(adr)
NULL	l.null()

L'exemple complet suivant montre comment inverser une liste d'étudiants en C++ :

```
// Inclusion de l'en-tete definissant le type abstrait.
#include "Liste.h"
```

```
// Raccourci pour TAdresse.
typedef Liste<Etudiant>::TIterator TAdresse;

// Inversion de liste.
void inverserListe( Liste<Etudiant> l_entree,      // E : passage par valeur
                    Liste<Etudiant> & l_inverse )  // ES : passage par reference
{
  // l_inverse est supposee vide.
  TAdresse adr = l_entree.adressePremier();
  while ( adr != l_entree.null() )
    {
      l_inverse.insererEnTete( l_entree.valeurElement( adr ) );
      adr = l_entree.adresseSuivant( adr );
    }
}

// Programme principal
int main()
{
  Liste<Etudiant> L1;
  Liste<Etudiant> L2;
  for ( int i = 0; i < 10; i++ )
    {
      Etudiant e(``sans nom'', i);
      L1.insererEnTete( e );
    }
  // en ne considerant que les notes:
  // L1 vaut 9 8 7 6 5 4 3 2 1 0
  inverserListe( L1, L2 );
  // en ne considerant que les notes:
  // L2 vaut 0 1 2 3 4 5 6 7 8 9
}
```

On remarque qu'il n'y a pas besoin d'initialiser la liste : le constructeur par défaut est invoqué automatiquement à la déclaration des variables. Comme le C++ est fortement typé, on est obligé de définir un type `TAdresse` pour chaque type de liste. Dans l'exemple ci-dessus, on a défini le type `TAdresse` pour les listes d'étudiants.

Dans ce TP, vous allez écrire dans un fichier `gestion.cc` toutes les fonctions demandées ci-dessous. Vous écrirez le programme principal permettant de les utiliser (et de les valider) dans le fichier `main.cc`. Les prototypes des fonctions de `gestion.cc` seront placés dans le fichier entête `gestion.h`.

Exercice 35 : Fichiers gestion.* et Makefile
Créez (vides pour l'instant) les deux fichiers `gestion.cc` et `gestion.h` et modifiez le `Makefile` en conséquence.
Dans quel fichier doit maintenant se trouver la définition suivante ?
`typedef Liste<Etudiant>::TIterator TAdresse;`

Exercice 36 : Liste non triée

1. Écrivez une action qui permette de saisir une liste d'étudiants (noms et notes).
2. Écrivez une action qui permette d'afficher une liste d'étudiants (noms et notes). Vérifiez que votre fonction de saisie fonctionne correctement.
3. Écrivez une fonction qui étant donné un nom `n` renvoie son rang dans une liste (1, 2, etc.) ou retourne `0` s'il est absent.
4. Écrivez une fonction qui retourne la moyenne des notes de la liste des étudiants.

Exercice 37 : Liste triée

Nous considérons à présent la liste d'étudiants triée en fonction des notes (ordre croissant).

1. Écrivez une action qui permette de rajouter un étudiant à la bonne place dans une liste triée.
2. Écrivez une fonction qui vérifie si une liste est triée (retourne un booléen).
3. Écrivez une action qui fusionne deux listes triées en une liste triée.

Exercice 38 : Tri

Supposons que le module serve à gérer la notation d'un qcm (questionnaire à choix multiples). Les étudiants ne peuvent donc avoir que répondu faux (note 0) ou juste (réponse 1). Écrivez une action qui place tous les élèves ayant répondu faux en début de liste et tous les élèves ayant répondu juste en fin de liste.