TP Semaines 15 et 16
Implémentation de types abstraits de données : les Listes

Nous allons aujourd'hui manipuler une classe nommée `Liste<T>`, dont l'interface correspond au type abstrait Liste. Dans les premiers exercices, la liste sera implémentée sous la forme d'une liste chaînée utilisant des pointeurs (allocation non contiguë).

Exercice 1

Recopiez chez vous le répertoire `/net/Bibliotheque/AP2/TP_par_Semaine/Semaine15_16`
Vous disposerez alors d'une classe `Liste`, qui implémente une liste par allocation non contiguë. Le Makefile vous est fourni, ainsi qu'un fichier de tests (comportant main) nommé `main.cc`.

Exercice 2

Complétez le fichier `Liste.cxx` en écrivant les fonctions membres :
adressePremier, adresseSuivant, valeurElement et modifierValeur.
Ecrivez aussi une fonction membre d'affichage afficher. Ecrivez maintenant le fichier `main.cc` ; écrivez une fonction menu() proposant :

- insérer en tête
- supprimer en tête
- afficher
- quitter

La fonction `main()` déclare une liste (pour simplifier choisissez `Liste<int>` ou `Liste<char>` par exemple), appelle menu(), puis, dans une boucle, saisit le choix et l'exécute jusqu'à ce que l'utilisateur choisisse de quitter.

A la fin des exercices 3, 4, 5 et 7, vous modifierez la fonction menu() pour qu'elle permette à chaque fois de vérifier les fonctions nouvellement implémentées.

Exercice 3

Ecrivez la fonction insererApres de `Liste.cxx`. Ecrivez aussi une fonction membre `placer(const T& elem)` qui range l'élément elem à sa place (ordre croissant) dans la liste. Evidemment, le type T doit offrir l'opérateur $<$ (ce qui est le cas avec int ou char :-), si la liste n'est pas triée, l'élément sera placé avant le premier plus grand que lui, ou à la fin selon le cas. Testez ces nouvelles méthodes dans `main.cc`.

Exercice 4

Après avoir écrit supprimerApres, écrivez aussi une fonction membre detruire(int rang) qui supprime le k^{eme} élément d'une liste. Testez cette méthode dans `main.cc`.

Exercice 5

Implémentez maintenant le constructeur de copie puis surchargez l'opérateur d'affectation.

Exercice 6

Au même niveau que `ListePtr`, créez un répertoire `ListeTab`.
Copiez-y `/net/Bibliotheque/AP2/TP_par_Semaine/Semaine15_16/source/ListeTab/*`.
Copiez-y également les fichiers `main.cc` et `Makefile` que vous venez d'écrire et de tester.

Ce répertoire ressemble à celui dans lequel vous venez de travailler. Mais en lisant le fichier `Liste.h`, vous constatez que la liste est maintenant implémentée à l'aide d'un tableau de structures alloué dynamiquement et nommé `m_bloc` (allocation contiguë à cellules non contiguës) ; chaque cellule du tableau comporte deux champs : le champ `info` contient un élément de la liste, le champ `suivant` contient l'indice dans le tableau de la cellule contenant l'élément suivant ; on "entre" dans la liste grâce à un attribut entier nommé `m_premier`, qui est simplement l'indice de la cellule contenant le premier élément. On pourra remarquer sans toutefois s'en préoccuper que les cellules libres sont chaînées de la même façon, grâce à un deuxième point d'entrée : `m_premierLibre`.

Ecrivez les seules fonctions manquantes adressePremier, adresseSuivant, valeurElement et modifierValeur. Quand c'est fait, vous pouvez compiler et vérifier ainsi que deux implémentations distinctes de Liste possédant la même interface se comportent exactement de la même façon pour l'utilisateur.

Exercice 7

Ecrivez dans `Liste.cxx` une fonction membre trier(). Vérifiez que le résultat est correct avec chacune des deux implémentations.