REVISIONS

1 Objectifs du TD

1) Implémenter une liste chainée.
2) Réviser les notions vue dans les TPs précédents.
3) Utiliser new et delete.
4) Comprendre l'utilité de dynamic cast.

Le but de ce TP est d'implémenter une structure de liste chainée d'éléments comparables qui trie automatiquement la liste lorsque les éléments sont insérés. Vous appliquerez cette liste à des entiers et des réels.

EXERCICE 1 – L'interface Comparable.

1) Déclarez une interface `Comparable` contenant 2 méthodes virtuelles pures :
   – une méthode `more` prenant en paramètre un pointeur sur un `Comparable` et renvoyant *True* si l'objet de la classe est supérieur à l'élément pointé par le paramètre,
   – une méthode qui affiche un objet `Comparable`.

EXERCICE 2 – La cellule, élément de base d'une liste.

1) Déclarez la classe `Cell` qui contient les données publiques suivantes : un pointeur `object` sur un `Comparable` et deux pointeurs `right` et `left` sur des cellules. Ajoutez un constructeur par défaut, un constructeur à paramètre `Comparable *`, un constructeur de copie et le destructeur.
2) Définissez les méthodes en initialisant les pointeurs à `NULL` par défaut dans la liste d'initialisation.

EXERCICE 3 – La liste doublement chaînée.

1) Déclarez la classe `Liste` avec deux pointeurs `first_elt` et `last_elt` sur des cellules, un constructeur par défaut, le destructeur et 2 méthodes `add` permettant d'ajouter un élément `Comparable` à la liste et `printlist` permettant d'afficher la liste.
2) Définissez le constructeur par défaut en initialisant les pointeurs.
3) Définissez la méthode `add`. Celle-ci doit insérer l'objet dans une liste supposée triée de facon à ce que cette liste reste triée (on utilisera bien sûr la méthode `more`). Notez que l'ajout d'un élément à la suite suppose l'allocation manuelle d'un nouveau bloc mémoire et donc l'utilisation de `new`.
4) Définissez le destructeur en prenant garde de bien libérer la mémoire.
5) Définissez la méthode `printlist`.

EXERCICE 4 – La classe Integer.

1) Déclarez la classe `Integer` qui a un champ entier privé, hérite de `Comparable` et possède un constructeur par défaut, à paramètre et un destructeur en plus.
2) Définissez toutes les méthodes à l'exception de `more`.
3) La méthode
   tt more s'applique à un pointeur sur un objet de la classe `Comparable` mais nécessite une comparaison avec un paramètre de la classe `Integer` qui en hérite. Il faut donc forcer cet objet à être de type `Integer`. On utilise pour cela `dynamic_cast <Integer &> (obj)`, où *obj* est l'objet à transtyper. Définissez la méthode `more`.
4) Essayer de créer une liste chaînée d'entiers et de l'afficher.

EXERCICE 5 – La classe Real. Mêmes questions que l'exercice précédent pour la classe `Real` avec un champ `float` au lieu de `int`.

EXERCICE 6 – Que se passe-t-il à votre avis si vous ajoutez malencontreusement un élément de `Real` à une liste d'entiers ? Pour prévenir ce type d'erreurs, on utilise une propriété de `dynamic_cast`. Lorsqu'on essaye de faire un transtypage impossible vers un pointeur, `dynamic_cast` renvoie le pointeur `NULL`. Ajoutez dans les deux définitions de `more` un test permettant d'afficher un message d'erreur puis de quitter l'exécution (`exit(-1)`).