TDM n°1 Les itérateurs sur des listes

1- Préliminaire sur le codage et les tests

- Le codage se fera avec une séparation de l'interface et du programme. L'interface sera définie dans un .h (le type LISTE et les prototypes des fonctions) et les autres types de données nécessaires dans le .c avec le reste du code.
- On réalise des tests unitaires sur chaque sous programme de base (ceux qui n'appellent pas d'autres sous programmes) avant de tester les assemblages (interaction de sous programmes ou sous programmes appelant des sous programmes de base).
- Les tests doivent, autant que possible, être exhaustifs. Cela suppose qu'une réelle réflexion aura été mise en œuvre pour définir les tests utiles.
- On garde les fichiers de test et le résultat des tests unitaires aussi bien que les tests d'intégration, afin de pouvoir éventuellement refaire le point sur ce qui a été testé et ce qui n'a pas été testé en cas de bug dans une partie déjà intégrée.

2- Reprise de contact avec les pointeurs

1. Implémenter une liste doublement chaînée d'éléments. Les éléments seront des structures avec des champs : valeur (entier signé) et nom (chaîne de caractères), le tout correspondant à un type ELEMENT. Les éléments de la liste sont ordonnés par ordre croissant de la valeur.
2. Implémenter les sous programmes : initialiser_liste, vide_liste, est_premier, est_dernier, ajoute_élément (à sa place), imprimer_liste.
3. Faire un programme *main* de test permettant de créer une liste, tester si elle est vide, ajouter un élément, imprimer les éléments de la liste.

Remarque : reprenez et copier/coller autant que possible les CTD et les TDM du semestre précédent : notamment le CTD sur la liste doublement chaînée d'entiers et le TDM sur les ensembles.

3- Les itérateurs

1. Implémenter un itérateur de liste doublement chaînée avec un sens de parcours comme vu en CTD ce semestre. Plus précisément, créer un itérateur sur une liste passée en paramètre avec les opérateurs suivants :
   - *hasNext* : renvoie *true* si l'élément courant a un suivant, *false* sinon ;
   - *hasPrevious* : renvoie *true* si l'élément courant a un prédécesseur, *false* sinon ;
   - *next* : renvoie l'élément courant et passe à l'élément suivant ;
   - *previous :* renvoie l'élément courant et passe à l'élément précédent ;
   - *begin* : place le curseur au début de la liste ;
   - *end :* place le curseur à la fin de la liste.
2. Tester précisément la définition du sens de parcours, le passage à l'élément suivant, précédent sur plusieurs variables de type liste avant de détecter d'éventuels effets de bord.
3. Imprimer les données de liste dans le sens *début->fin* puis *fin->début* en employant une structure de contrôle de type *for*.
4. Fonction *map*
   - Programmer la fonction *map* avec en paramètre une liste et un pointeur de fonction et en utilisant l'itérateur sur la liste. *map* applique la fonction à tous les éléments de la liste, elle renvoie en sortie la liste des résultats de l'application.
   - Tester sur la fonction « mettre la première lettre du champ nom en majuscule ».
   - Utiliser la fonction *map* pour récolter l'ensemble des valeurs négatives de la liste.