Itérateur pour Liste doublement chaînée en dynamique

1 Contexte Général

Quand on cherche à parcourir une suite d'éléments, on utilise la plus part du temps des *itérateurs*. Il s'agit de données pour lesquelles on peut donner :
– une valeur de début,
– une valeur de fin,
– une manière de passer à la valeur suivante.
Cela se voit sans difficulté sur le code très simple suivant :

```
for (i = 0 ; i<N; i++)
{
...
}
```

ou bien :

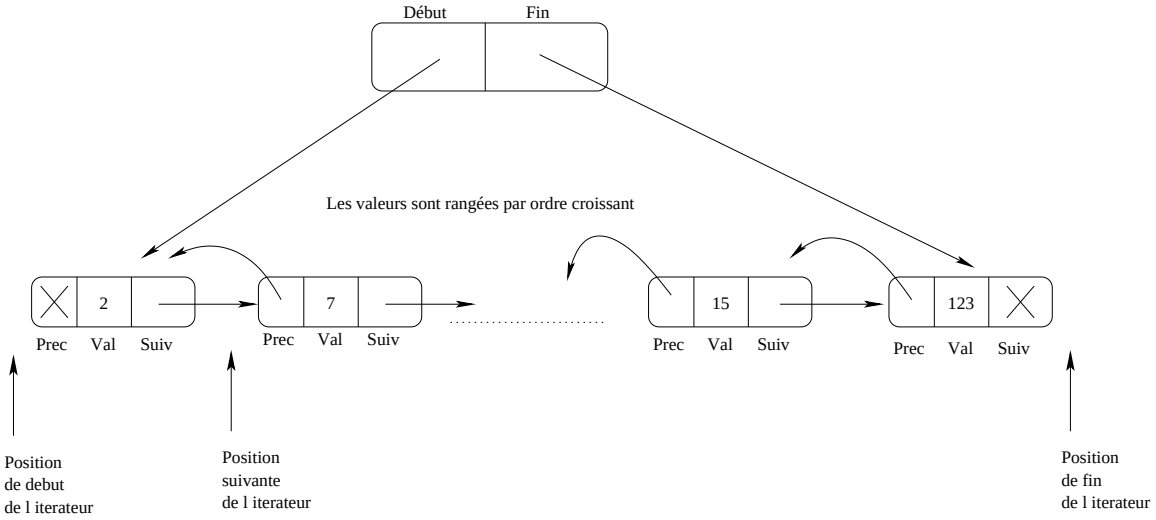
```
for (i = N-1 ; i>=0; i--)
{
...
}
```

A chaque fois, la variable i sert d'itérateur avec :
– comme valeur de début soit 0, soit $N-1$,
– comme valeur de fin soit $N-1$, soit 0,
– comme passage au suivant soit une incrémentation, soit une décrémentation.
Ce mécanisme peut se généraliser de la manière suivante (en pseudo-code objet) :

```
i = creerIterateur(CROISSANT) ;
for (i.debut(i) ; ! i.fin(i); i.suivant(i))
{
...
}
```

2 Utilisation sur une LDCD

Il s'agit de définir une structure de données ITERATEUR permettant le parcours des LDCD d'entiers définies au TD précédent. On aura donc le schéma suivant :



Un itérateur sur une LDCD peut donc être vu comme un *curseur* qui se déplace dans un sens ou dans l'autre le long de la liste. On considère en général que le curseur est "entre" deux éléments (voir schéma ci-dessus). Au départ, ce curseur est situé juste avant le premier élément de la liste de façon à ce que l'appel de la fonction `next` permette de récupérer ce premier élément. De la même façon, si le curseur est en fin de liste, il est situé après le dernier élément et l'appel de la fonction `previous` permet de récupérer ce dernier élément. Cette structure de données permet ainsi de parcourir soit par ordre croissant, soit par ordre décroissant n'importe quelle LDCD.

3 Questions à résoudre

1. Proposer en langage C un type de données dynamique pour un ITERATEUR sur une LDCD d'entiers.
2. Écrire une fonction qui crée un itérateur pour une ldcd donnée sans préciser pour l'instant le sens de parcours.
3. Écrire les fonctions `next`, `previous`, `hasNext` , `hasPrevious` qui permettent respectivement de :
   - récupérer l'élément suivant dans la liste et de déplacer l'itérateur après cet élément,
   - récupérer l'élément précédent dans la liste et de déplacer l'itérateur avant cet élément,
   - savoir s'il existe un élément suivant,
   - savoir s'il existe un élément précédent.
4. Si on veut imposer un sens de parcours lors de la création de l'itérateur que faut-il modifier ?
5. Si on veut utiliser l'itérateur pour insérer de nouvelles données dans la ldcd que faut-il ajouter comme fonctions ? Donner le code de ces fonctions.

Vous devrez utiliser le mécanisme de la compilation séparée.