Feuille de travaux dirigés n°6
Structures de données

Exercice 6.1 — Listes Simplement chaînées

On supposera que l'on dispose des primitives suivantes :

- `donnée(elt)` : renvoie la donnée associée à l'élément `elt`.
- `suivant(elt)` : renvoie l'élément suivant l'élément `elt`.
- `premier(L)` : renvoie le premier élément de la liste `L`, renvoie nil si la liste est vide.

Pour alléger l'écriture, on considérera qu'on peut utiliser une primitive

- pour connaître un résultat (utilisation en Rvalue ou Right-value) : `y <- suivant(x)` affecte dans `y` la valeur suivante de `x`.
- pour affecter une donnée interne (utilisation en Lvalue ou Left-value) : `suivant(x) <- y` affecte la valeur suivante de `x` à `y`.

Écrivez les fonctions suivantes en mettant éventuellement à jour les primitives précédentes :

1. `estVide(L)` renvoie vrai si la liste est vide et faux sinon.
2. `supprimerPremier(L)` supprime le premier élément de `L`.
3. `vider(L)` supprime tous les éléments de `L`.
4. `ajouterEnTête(x,L)` ajoute `x` au début de `L`.
5. `ajouterAprès(x,y,L)` ajoute `x` dans `L` après `y`.
6. `supprimerSuivant(x,L)` supprime le suivant de `x` dans `L`.
7. `numÉlément(L)` renvoie le nombre d'éléments de `L`. Quelle est la complexité de cette fonction ?

Exercice 6.2 — Comptage des éléments

On veut améliorer la complexité de la fonction `numÉlément(L)`. Pour ce faire, on introduit une nouvelle primitive `num(L)` qui contient le nombre d'éléments de `L`.

1. Réécrivez toutes les fonctions précédentes en tenant compte de cette primitive.
2. De quelle primitive devrait-on disposer pour pouvoir ajouter un élément à la fin de la liste ? Quelles sont les modifications à apporter aux fonctions précédentes dans ce cas ? Réécrivez les fonctions avec cette nouvelle information.
3. Donnez un algorithme qui insère un élément dans une liste triée. Essayez de trouver une solution qui n'a pas besoin de mémoriser dans une variable temporaire l'élément précédent (aide : la donnée associée à un élément peut être modifiée).
4. On suppose que l'on a deux listes `L1` et `L2` triées par ordre croissant et que l'on dispose de la fonction booléenne `donnéePlusGrande(x,y)` qui est vraie si la donnée associée à `x` est plus grande que la donnée associée à `y`. Le but est d'arriver à fusionner les deux listes de façon à obtenir une seule liste triée.
   - Considérez les deux listes dont les données sont les lettres suivantes : `L1` : a, c, g, j, k et `L2` : b, h, m, n. Représentez graphiquement les listes.
   - Proposez une méthode pour insérer les éléments de `L2` dans `L1` pour que `L1` soit triée à la fin. Donnez deux versions, une qui vide la liste `L2` et une autre qui garde intacte la liste `L2`.

Exercice 6.3 — Listes Doublement chaînées

Une liste doublement chaînée est une liste qui, en plus de permettre l'accès au suivant d'un élément, permet l'accès au précédent d'un élément. Pour représenter cette nouvelle liste, on introduit la primitive `précédent(x)` qui renvoie l'élément précédant l'élément `x`.

— Quel est l'intérêt de ce type de liste par rapport aux listes simplement chaînées ?

— Écrivez les fonctions suivantes en mettant éventuellement à jour les primitives précédentes :

1. `supprimerPremier(L)` supprime le premier élément de `L`.
2. `vider(L)` supprime tous les éléments de `L`.
3. `ajouterAprès(x,y,L)` ajoute `x` dans `L` après `y`.
4. `supprimerSuivant(x,L)` supprime l'élément suivant `x` dans `L`.

On veut maintenant gérer le dernier élément de la liste. On pourrait introduire une primitive contenant le dernier élément, mais comme vous l'avez montré cela complique les fonctions.
Pour résoudre ce problème, on peut travailler avec des listes circulaires : on connaît le premier et le précédent du premier est le dernier élément de la liste.
Afin de résoudre les problèmes de l'existence d'une donnée dans la liste, on introduit un élément fictif que l'on appelle sentinelle. Le premier élément réel de la liste devient donc le suivant de la sentinelle et le dernier élément de la liste est le précédent de la sentinelle. On peut donc supprimer la primitive `premier`. On la remplace par la primitive `sentinelle`.

— Faites un dessin qui montre cela.

— Écrivez les fonctions suivantes en mettant éventuellement à jour les primitives précédentes :

1. `premier(L)` renvoie le premier élément de `L`.
2. `dernier(L)` renvoie le dernier élément de `L`.
3. `estVide(L)` renvoie vrai si la liste est vide et faux sinon.
4. `supprimerPremier(L)` supprime le premier élément de `L`.
5. `vider(L)` supprime tous les éléments de `L`.
6. `ajouterAprès(x,y,L)` ajoute `x` dans `L` après `y`.
7. `supprimer(x,L)` supprime `x` dans `L`.

Exercice 6.4 — Opérations sur les listes

On considère que `L1` et `L2` sont deux listes doublement chaînées circulaires avec sentinelle.

1. Faites un dessin qui montre comment vous allez ajouter `L2` à la fin de `L1`.
2. Écrivez la fonction `ajouterEnFin(L1,L2)`.
3. Faites la même chose pour ajouter `L2` au début de `L1`. On nommera la fonction `ajouterAuDébut(L1,L2)`.
4. Faites une fonction qui supprime de `L1` tous les éléments dont la donnée associée est impaire (on utilisera la fonction `donnéeEstPaire(x)`) et qui ajoute tous ceux de `L2` dont la donnée associée est paire.