Université Lille1 Sciences et Technologies

ASD

2012-2013
Licence STS - Semestre 4

TP5 : listes avec itérateurs

Vous avez vu en API2 comment construire une liste définie récursivement comme une tête et un reste. Ce type de liste est dit *simplement chaînée* et une représentation graphique de celle-ci est donnée Figure 1. L'inconvénient de la

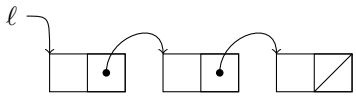

FIGURE 1 – Représentation d'une liste simplement chaînée. Chaque *cellule* contient la valeur de l'élément et une référence vers le reste (la cellule suivante).

définition vue en API2 :

```
type 'a liste = Vide | Cons of 'a * 'a liste
```

est que les éléments de la liste ne peuvent être modifiés. Une définition alternative consiste donc à définir un type `cellule` qui contient l'élément et une référence vers le reste de la liste. On aboutit à la définition suivante :

```
type 'a liste =
  | Vide
  | Cons of 'a cellule
and 'a cellule = {
  valeur : 'a;
  mutable suivant : 'a liste;
}
```

On peut allez encore plus loin en créant ce qu'on appelle des listes *doublement chaînées* dont une représentation est donnée Figure 2. En OCAML on peux réaliser cela grâce à la définition suivante :

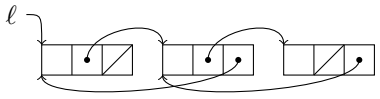

FIGURE 2 – Représentation d'une liste doublement chaînée. Chaque cellule contient la valeur de l'élément stocké et deux références : l'une vers la cellule suivante et l'autre vers la cellule précédente.

```
type 'a liste =
  | Vide
  | Cons of 'a cellule
and 'a cellule = {
  valeur : 'a;
  mutable suivant   : 'a liste;
  mutable precedent : 'a liste;
}
```

L'avantage de telles listes est de pouvoir créer une structure supplémentaire qui permet de « se déplacer »sur la liste, c'est ce qu'on appelle un *itérateur* (voir Figure 3). La manipulation des listes avec un itérateur est plus facile : on peut

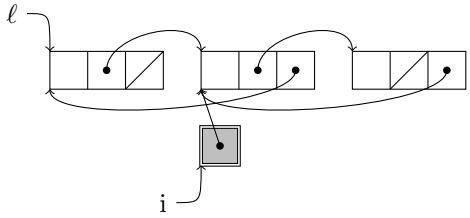

FIGURE 3 – Représentation d'une liste doublement chaînée et d'un itérateur. L'itérateur a une référence vers une cellule courante c.

se positionner au début de la liste, avancer, reculer, insérer avant, insérer après.

Le but du TP est l'implémentation de listes doublement chaînées avec itérateurs. On supposera dans notre implémentation qu'un itérateur est toujours défini, c'est-à-dire qu'il est toujours positionné sur un élément de la liste.

Récupérez auparavant l'archive disponible sur le portail. Vous trouverez dans un répertoire `tp5` : les fichiers `listeit.mli` et `listeit.ml` où vous implanterez les itérateurs de liste, et un programme `testtp5.ml` avec les entêtes des fonctions à écrire et le code pour des tests. Attention, le code ne compile pas tant que les premières questions ne sont pas traitées.

1 La liste doublement chaînée

Commencez par observer le code livré et comprendre les fonctions déjà implantées.

Q 1 Pourquoi y a-t-il une definition de `liste_interne` dans la définition de la structure de liste doublement chaînée ?

Q 2 A quoi sert la fonction `la_cellule` ?

2 Création des itérateurs

Q 3 Déclarer le type `iterateur` dans le module `Listeit`. Faites valider par votre enseignant.

Q 4 Compléter le code des fonctions : `iterateur_en_debut`, `iterateur_en_fin`, `est_en_fin`, `est_en_debut`, `avancer`, `reculer`, `valeur`. Bien regarder la déclaration de ces fonctions dans le fichier `listeit.mli` pour comprendre le comportement de celles-ci.

Q 5 Implanter dans le programme `testtp5.ml` les deux fonctions d'affichage de la liste, à l'endroit et à l'envers, qui utilisent les primitives sur les itérateurs.

Q 6 Réaliser le test 1 (commenter/décommenter le code dans `testtp5.ml`).

3 Insertion avec des itérateurs

Q 7 Compléter le code des fonctions : `inserer_avant` et `inserer_apres`. Pour ces deux procédures, une fois l'insertion réalisée, l'itérateur sera placé sur le nouvel élément.

Q 8 Tester les nouvelles fonctionnalités : tests 2 et 3.

Q 9 Réaliser maintenant les tests 4 et 5. Que constatez-vous ? Remédiez au problème.

Q 10 Compléter, dans le programme de test, la procédure `inserer_trie` qui ajoute un élément à la liste en conservant la liste dans un ordre croissant.

Q 11 Ecrire le code du test 6 permettant de vérifier le bon fonctionnement de votre insertion triée.

4 Suppression avec des itérateurs

Q 12 Pensez-vous que l'implantation réalisée ici soit compatible avec des opérations de suppression ?

En fait, la suppression d'un élément ne pose pas de problème particulier mis à part que l'itérateur n'est plus positionné sur un élément. On peut alors prendre le parti que l'itérateur après suppression est positionné sur l'élément suivant. Mais que se passe-t-il lorsqu'il n'y a plus de suivant ? Il faut nécessairement envisager que l'itérateur puisse devenir indéfini, c'est-à-dire qu'il pointe sur `Vide`, contrairement à l'hypothèse que nous avions faite au début du TP

Q 13 Définir une nouvelle exception `IterateurIndefini`.

Q 14 Modifier le code de toutes les fonctions nécessaires pour qu'elles déclenchent l'exception `IterateurIndefini` si besoin. Indiquez quelles sont les fonctions modifiées.

Q 15 Implanter une fonction `supprimer` qui prend en paramètre un itérateur et supprime l'élément pointé par l'itérateur de la liste. Il passe à l'élément suivant si il existe, ou rend l'itérateur indéfini sinon.

Q 16 Réaliser les tests 7 et 8.

Gestion de plusieurs itérateurs

La solution mise en œuvre convient tant qu'il n'y a qu'un seul itérateur en jeu. En effet, rien n'empêche de définir plusieurs itérateurs, dont deux pointant le même élément, et de procéder à la suppression de l'élément par l'un des deux itérateurs. Dans ce cas, l'autre itérateur doit être indéfini, mais cela ne se fait pas automatiquement ...

Q 17 Avec une liste et deux itérateurs, reproduire le schéma décrit ci-dessus.

Q 18 Décrire dans votre compte-rendu, en français, une solution pour remédier à ce problème.

Q 19 BONUS Si vous avez le temps mettez la en œuvre.