Algorithmique / Programmation
Les listes : implémentations

1 Implémentation statique / contiguë : dans un tableau

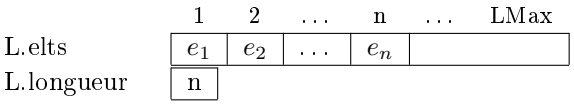
1.1 Le type

Les élements de la liste sont stockés dans un tableau (un vecteur) : la $i^{ème}$ case contient le $i^{ème}$ élément de la liste. Cette représentation limite donc la longueur de la liste qui ne peut dépasser la taille du tableau (d'où la nécessité d'un surdimensionnement pour ne pas avoir de problème en cas de nombreux ajouts). Afin de ne prendre en compte que les cases du vecteurs contenant les éléments de la liste, il est nécessaire de connaître sa longueur. Donc la liste sera représentée par un couple $< Tableau, Entier >$:

```
constantes
    LMax = ...

types
    /* t_element*/
    t_vect_elts  = LMax t_element
    t_list = enregistrement
        t_vect_elts   elts
        entier        longueur
    fin enregistrement t_list

    variables
       t_list   L
```



1.2 Implémentation des opérations

Cette représentation est parfaitement adaptée aux listes itératives (mais elle peut également être adaptée aux listes récursives, où la longueur ferait office de place de tête et où il n'y a aucun transfert de valeur à effectuer au milieu du tableau).

1.2.1 De simples correspondances

Type abstrait : Liste itérative	**Implémentation : type t_list**
λ : Liste	L : t_list
$\lambda \leftarrow$ *liste-vide*	L.longueur$\leftarrow$ 0
longueur(λ)	L.longueur
ième(λ, k)	L.elts[k]

1.2.2 Des algorithmes

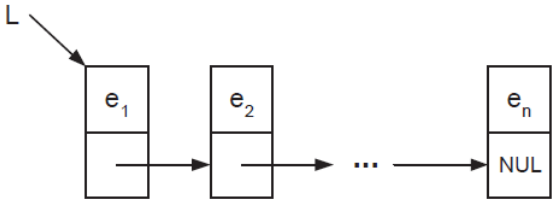
L'accès au $k^{ème}$ élément d'une liste est immédiat. Par contre, pour l'insertion et la suppression, il faut déplacer (à l'aide d'une boucle) tous les éléments de la place où s'effectue l'opération jusqu'à la fin de la liste. Si n est la longueur de la liste, une suppression ou un ajout demande au plus $n-1$ affectations (suppression ou insertion en première place = le pire des cas).

2 Implémentation dynamique / chaînée

2.1 Le type

On utilise les pointeurs pour chaîner entre eux les éléments successifs. Chaque élément est dans un enregistrement qui contient un pointeur sur l'enregistrement contenant l'élément suivant [1] (la valeur NUL pour le dernier élément). La liste est alors représentée par un pointeur sur son premier élément (sur l'enregistrement contenant le premier élément) si elle n'est pas vide, la valeur NUL dans le cas contraire.

1. Par abus de langage, on dira qu'il pointe sur l'élément suivant




```
types
    /* t_element*/
   t_pList = ↑ t_node

   t_node = enregistrement
       t_element   val
       t_pList     suiv
   fin enregistrement t_node

variables
   t_pList   L
```

2.2 Implémentation des opérations

La représentation est bien adaptée aux listes récursives.

La représentation d'une *place* est du même type que celle d'une *liste*. Une liste est représentée par la *place* de son premier élément.

Type abstrait : Liste récursive	**Implémentation : t_pList**
λ : Liste	L : t_pList
liste-vide	NUL
tête(λ)	L
fin(λ)	L↑.suiv
first(λ)	L↑.val
p : Place	p : t_pList
contenu(p)	p↑.val
succ(p)	p↑.suiv

L'opération cons

L'opération correspond à l'ajout en tête, qui est le moyen le plus simple d'ajouter un nouvel élément dans une liste chaînée. On ajoute le `t_element` e en tête de `t_pList` L :

```
allouer (new)     // new : variable de type t_pList
new↑.val ← e
new↑.suiv ← L
L ← new
```

Remarque

Il est bien entendu possible d'implémenter les opérations des *listes itératives*. Mais les listes chaînées nous font perdre un atout essentiel des tableaux : l'accès direct. L'accès au $i^{ème}$ ou le calcul de la longueur par exemple nécessitent de parcourir la liste. Par contre, on peut ajouter ou supprimer n'importe où sans faire de décalages (mais il faut parcourir pour trouver la place, sans oublier de garder l'accès à l'élément précédent...).