

Feuille 3 : Listes doublement chaînées

Dans les exercices suivants on considère le type `listeDC`, liste doublement chaînée.

IMPLÉMENTATION DU TYPE `listeDC` PAR ALLOCATION DYNAMIQUE

Dans le contexte considéré le **type curseur** est le type pointeur vers une cellule.

```
curseur= ^cellule;
car= type_predefini;
cellule= structure
    valeurElement: car;
    pointeurPrecedent: curseur;
    pointeurSuivant: curseur;
finstructure;

listeDC_car= structure
    premier: curseur;
    dernier: curseur;
    cle:curseur;
finstructure;
```

La liste vide est représentée par `NIL`.

— Gestion de la mémoire

```
fonction new(ref p: ^cellule) : vide;
fonction delete(ref p: ^cellule) : vide;
```

Exercice 3.1

Soit l'implémentation de `listeDC` par `listeDC_car` avec l'allocation dynamique. Pour rappel voir annexe A.

1. Écrire les primitives du type `listeDC_car`.

```
fonction insererApres(ref L: listeDC_car, val X: car): vide;
fonction insererEnTete(ref L: listeDC_car, val X: car): vide;
fonction supprimerApres(ref L: listeDC_car): vide;
fonction supprimerEnTete(ref L: listeDC_car): vide;
```

2. Écrire les fonctions :

- (a) fonction `dernier(val L: listeDC_car): car`;
qui renvoie la valeur du dernier élément de la liste `L`;
- (b) fonction `ajout_en_queue_sans_doublons(ref L: listeDC_car, val X: car) : vide`;
qui ajoute l'élément `X` en queue de la liste `L` s'il n'y est pas déjà;
- (c) fonction `supprimer_derniere_occurrence(ref L: listeDC_car, val X: car) : vide`;
qui supprime dans la liste `L` la dernière occurrence de l'élément `X` donné (s'il existe, sinon la fonction ne fait rien).

Exercice 3.2

En mathématiques, quand on considère un ensemble d'objets $\{x_1, x_2, \dots, x_N\}$, on fait abstraction de l'ordre des objets et l'on ne répète pas plusieurs fois un même objet. Ainsi, l'ensemble des éléments qui composent la liste (a, b, c, a, d, e, c, d) est $\{a, b, c, d, e\}$.

1. Définir le type `ensemble` d'objets.
2. Ecrire une fonction qui convertit une liste `L` de type `listeDC_car` en un ensemble d'objets.
3. Ecrire une fonction qui calcule la réunion de deux ensembles constituée des éléments qui sont au moins dans un des ensembles.
4. Ecrire une fonction qui calcule l'intersection de deux ensembles constituée des éléments qui sont dans les deux ensembles.

Problème récurrent

Exercice 3.3 Gestion d'une piste d'atterrissage

Un avion est caractérisé par un enregistrement contenant :

- un indicatif (6 caractères)
- sa destination (30 caractères)
- son autonomie résiduelle de carburant, comptée en heures de vol (entier)
- deux booléens indiquant s'il y a un pirate à bord et s'il y a le feu.

Le problème consiste à

1. définir les structures de données nécessaires à la gestion d'une piste d'atterrissage ;
2. définir et écrire une fonction calculant la priorité d'un avion pour l'utilisation de la piste ;
3. définir et écrire les fonctions nécessaires à la gestion complète de la piste (on envisagera la suppression d'un avion piraté de la file d'attente lorsque le pirate a mis sa menace de détournement à exécution).

Quelles notions vues dans ce TD peuvent permettre d'amorcer la résolution du problème ?

ANNEXE A

— Accès

```
fonction valeur(val L:listeDC_car) : car;
fonction debutListe(ref L:listeDC_car) : vide;
fonction finListe(ref L:listeDC_car): vide;
fonction suivant(ref L:listeDC_car) : vide;
fonction precedent(ref L:listeDC_car) : vide;
fonction listeVide(val L:listeDC_car) : booleen;
fonction getCleListe(val L:listeDC_car) : curseur;
```

— Modification

```
fonction creerListe(ref L:listeDC_car) : vide;
fonction insererAprès(ref L:listeDC_car, val L:objet;) : vide;
fonction insererEnTete(ref L:listeDC_car, val X:objet) : vide;
fonction supprimerAprès(ref L:listeDC_car) : vide;
fonction supprimerEnTete(ref L:listeDC_car) : vide;
fonction detruireListe(ref L:listeDC_car) : vide;
fonction setCleListe(ref L:listeDC_car, val c curseur) : vide;
```