

## Algorithmique 1 : TD 2

### Feuille 2 : Listes simplement chaînées

Dans les exercices suivants on considère le type `listeSC`, liste simplement chaînée, défini en cours. Pour rappel voir annexe A.

#### Exercice 2.1

Soit l'implémentation par un tableau du type liste simplement chaînée `listeSC` par le type `listeSC_Car` avec une gestion de l'espace de stockage comme défini en cours. Pour rappel voir annexe B.

1. Que fait la fonction suivante :

```
fonction mystere(ref L: listeSC_Car, val X: car): boolean;
var p: curseur;
debut
  si L.premierLibre == 0 alors
    retourner faux;
  sinon
    p= prendreCellule(L);
    L.vListe[p].valeur= X;
    L.vListe[p].indexSuivant= L.premier;
    L.premier= p;
    L.cle= L.premier;
    retourner vrai;
fin
```

2. Ecrire les primitives :

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

3. Ecrire une primitive supplémentaire :

```
fonction est_dernier(val L: listeSC_Car): boolean;
qui renvoie Vrai lorsque la clé est sur le dernier élément de la liste L;
Dans la suite on pourra s'interroger sur l'utilité de disposer ou non de cette primitive.
```

4. Ecrire les fonctions :

- (a) fonction `appartient(val L: listeSC_Car, val X: car): boolean;`  
qui teste si l'élément `X` appartient à la liste `L`;
- (b) fonction `dernier(val L: listeSC_Car): car;`  
qui renvoie la valeur du dernier élément de la liste `L`;
- (c) fonction `rang(val L: listeSC_Car, val X: car): entier;`  
qui calcule et retourne la position de la première occurrence d'un élément `X` dans la liste `L`;
- (d) fonction `ajout_en_queue_sans_doublons(ref L: listeSC_Car, val X: car) : vide;`  
qui ajoute l'élément `X` en queue de la liste `L` s'il n'y est pas déjà;

## Exercice 2.2

Soit l'implémentation par allocation dynamique du type liste simplement chaînée `listeSC` par le type `listeSC_Car` comme défini en cours.

Pour rappel voir annexe C.

1. Que fait la fonction suivante :

```
fonction mystere(ref L:listeSC_car, val X: car) : vide;
var p: curseur;
debut
  new(p);
  p^.valeurElement= X;
  p^.pointeurSuivant= L.premier;
  L.premier= p;
  L.cle= L.premier;
fin;
```

2. Ecrire les primitives suivantes :

```
fonction insererEnTete(ref L: listeSC_car, val X:car): vide;
fonction supprimerApres(ref L: listeSC_car): vide;
```

3. Ecrire une primitive supplémentaire :

```
fonction est_dernier(val L: listeSC_Car): boolean;
qui renvoie Vrai lorsque la clé est sur le dernier élément de la liste L;
Dans la suite on pourra s'interroger sur l'utilité de disposer ou non de cette primitive.
```

4. Ecrire les fonctions suivantes avec la nouvelle implementation de `listeSC` par `listeSC_car` :

- (a) fonction `appartient(val L: listeSC_Car, val X: car): boolean;`  
qui teste si l'élément `X` appartient à la liste `L`;
- (b) fonction `supprimer_derniere_occurrence(ref L: listeSC_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).
- (c) fonction `ajout_en_tete_sans_doublons(ref L: listeSC_car, val X: car) : vide;`  
qui ajoute en tête dans la liste `L` l'élément `X` s'il n'est pas déjà présent dans la liste  
(sinon la fonction ne fait rien).
- (d) fonction `supprimer_premiere_occurrence(ref L: listeSC_car, val X: car) : vide;`  
qui supprime dans la liste `L` la première occurrence de l'élément `X` donné s'il existe,  
(sinon la fonction ne fait rien).

## Exercice 2.3

Proposer un algorithme de création d'une liste d'entiers de type `listeSC` qui ne contient pas de doublons et dont les éléments sont ordonnés dans l'ordre croissant des valeurs.

## Problème récurrent ( notre fil d'ariane)

### Exercice 2.4 Gestion d'une piste d'atterrissage des avions

Un avion est un enregistrement contenant :

- l'indicatif (6 caractères)
- la destination (30 caractères)
- l'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.

1. Définir les structures de données nécessaires.
2. Ecrire la fonction `Priorité` ainsi que la gestion complète de la piste.
3. Envisager le cas de suppression d'un élément quelconque de la file lorsque le pirate a mis sa menace de détournement à exécution.

Quelles sont les notions que vous venez de voir qui peuvent permettre d'amorcer le fil d'ariane ?

### Travail personnel

#### Exercice 2.5 Annales d'examen : Décembre 2010 (L2 Informatique)

Soit une suite de clés dans un tableau `T`. On considère un tableau de dimension `N` contenant des entiers appartenant à l'ensemble  $\{0, 1\}$ . Par exemple, le tableau de dimension 8 est donné par `T[1]=0, T[2]=1, T[3]=0, T[4]=0, T[5]=1, T[6]=0, T[7]=1, T[8]=1`.

1. Ecrire la fonction `verifier` qui vérifie que les éléments du tableau appartiennent à l'ensemble  $\{0, 1\}$  et que le nombre de zéros est le même que le nombre de 1.
2. En utilisant les primitives du type abstrait `listeSC`, écrire la fonction `tableauListe` qui a pour paramètre un tableau d'entiers `T` et fournit en sortie une liste simplement chaînée `L` dans le même ordre que le tableau d'entrée (sur l'exemple, `L=(0,1,0,0,1,0,1,1)`).
3. Ecrire cette même fonction `tableauListe` en utilisant l'implémentation dynamique sans appel aux primitives du type abstrait `listeSC`.

#### Réalisables après le cours sur les Piles

4. Ecrire une fonction `facteur01` qui à partir de la liste `L` simplement chaînée fournit la pile des indices `j` tels que `T[j]=0` et `T[j+1]=1` dans l'ordre inverse (sur l'exemple `[6,4,1[` où 1 est le sommet de pile).
5. Ecrire une fonction `extraitPair` qui extrait de la pile toutes les positions ayant un numéro pair et les range dans une file dans un ordre croissant sans changer la pile (sur l'exemple `[4,6[` où 4 est le premier de la file).
6. On souhaite écrire une fonction `supZeroPair` qui supprime dans la liste `L`, les valeurs 0 qui correspondent à des indices pairs dans le tableau (sur l'exemple, la nouvelle liste

est  $(0,1,0,1,1,1)$ ). La structure de liste simplement chaînée n'est pas adaptée. Pourquoi? Quelles sont les modifications à apporter dans la fonction `tableauListe`? Ecrire la fonction `supZeroPair`.

## ANNEXE A Liste simplement chaînée

listeSC= liste de type\_predefini;

défini en cours avec les primitives suivantes :

– **Accès**

```
fonction valeur(val L:liste d'objet) : objet;
fonction debutListe(ref L:liste d'objet) :vide;
fonction suivant(ref L:liste d'objet) : vide;
fonction listeVide(val L:liste d'objet) : booleen;
fonction getCleListe(val L: liste d'objet) : curseur;
```

– **Modification**

```
fonction creerListe(ref L:liste d'objet) : vide;
fonction insererApres(ref L:liste d'objet;
    val x:objet;) : vide;
fonction insererEnTete(ref L:liste d'objet
    val x:objet) : vide;
fonction supprimerApres(ref L:liste d'objet) : vide;
fonction supprimerEnTete(ref L:liste d'objet) : vide;
fonction setCleListe(ref L: liste d'objet, val c:curseur) : vide;
fonction detruireListe(ref L:liste d'objet) : vide;
```

## ANNEXE B Implémentation de listeSC par un tableau avec gestion de l'espace de stockage

Dans le contexte considéré le **type curseur** est le type entier.

```
curseur= entier;
car= type_predefini;
elementListe= structure
    valeur: car;
    indexSuivant: curseur;
finstructure;
stockListe= tableau[1..tailleStock] d'elementListe;
listeSC_Car= structure
    tailleStock: entier;
    vListe: stockListe;
    premier: curseur;
    premierLibre: curseur;
    cle:curseur;
finstructure;
```

– **Gestion de la liste des éléments libres**

```
fonction prendreCellule(ref L: listeSC_Car) : curseur;
fonction mettreCelluleEnTete(ref L: listeSC_Car, val P: curseur) : vide;
fonction listeLibreVide(val L: listeSC_Car) : booleen;
```

– **Accès**

```

fonction valeur(ref L: listeSC_Car) : car;
fonction debutListe(ref L: listeSC_Car) : vide;
fonction suivant(ref L: listeSC_Car) : vide;
fonction listeVide(val L: listeSC_Car) : booleen;
- Modification
fonction creer_liste(ref L: listeSC_Car) : vide;
fonction insererApres(ref L: listeSC_Car, val X: car) : booleen;
fonction insererEnTete(ref L: listeSC_Car, val X: car) : booleen;
fonction supprimerApres(ref L: listeSC_Car) : vide;
fonction supprimerEnTete(ref L: listeSC_Car) : vide;
fonction detruireListe(ref L: listeSC_Car) : vide;

```

### ANNEXE C Implémentation de listeSC par allocation dynamique

Dans le contexte considéré le **type curseur** est le type pointeur vers un élément.

```

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

listeSC_car=structure
    premier: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;
- Accès
fonction valeur(val L: listeSC_car) : car;
fonction debutListe(ref L: listeSC_car) : vide;
fonction suivant(ref L: listeSC_car) : vide;
fonction listeVide(val L: listeSC_car) : booleen;
- Modification
fonction creer_liste(ref L: listeSC_car) : vide;
fonction insererApres(ref L: listeSC_car, val X: car) : vide;
fonction insererEnTete(ref L: listeSC_car, val X: car) : vide;
fonction supprimerApres(ref L: listeSC_car) : vide;
fonction supprimerEnTete(ref L: listeSC_car) : vide;
fonction detruireListe(ref L: listeSC_car) : vide;

```