

# Algorithmique 1

## TD 2

### 1 Listes simplement chaînées

Dans les exercices suivants on considèrera le type `listeSC`, liste simplement chaînée,

`listeSC` = liste de type `type_predefini`;

défini en cours avec les primitives suivantes :

```
fonction premier(val L: listeSC): ^type_predefini;
fonction suivant(val L: listeSC, val P: ^type_predefini): ^type_predefini;
fonction listeVide(val L: listeSC): booleen;
fonction creerListe(ref L: listeSC): vide;
fonction insererApres(val X: type_predefini, ref L: listeSC,
                    val P: ^type_predefini): vide;
fonction insererEnTete(val X: type_predefini, ref L: listeSC): vide;
fonction supprimerApres(ref L: listeSC, val P: ^type_predefini): vide;
fonction supprimerEnTete(ref L: listeSC): vide;
```

#### Exercice 2.1

Soit l'implémentation de `listeSC` par `listeSC_Car` avec une gestion de l'espace de stockage comme défini en cours :

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

1. Que fait la fonction suivante :

```
fonction mystere(ref L: listeSC_Car, val X: car): booleen;
var p: entier;
debut
    si listeLibreVide(L) alors
```

```

    retourner(Faux)
sinon
    p= prendreCellule(L, L.premierLibre);
    L.vListe[p].valeur= X;
    L.vListe[p].indexSuivant= L.premier;
    L.premier= p;
    retourner(Vrai);
fin

```

2. Écrire les primitives :

```

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

```

3. Écrire les fonctions :

- (a) fonction `dernier(val L: listeSC_Car): car;`  
qui renvoie la valeur du dernier élément de la liste L;
- (b) fonction `appartient(val L: listeSC_Car, val X: car): boolean;`  
qui teste si l'élément X appartient à la liste L;
- (c) fonction `rang(val L: listeSC_Car, val X: car): entier;`  
qui calcule et retourne le rang de la première occurrence d'un élément X dans la liste L;
- (d) fonction `ajout_en_queue_sans_doublons(val 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;
- (e) fonction `supprimer_derniere_occurrence(val L: listeSC_Car, val X: car) : vide;`  
qui supprime dans la liste L la dernière occurrence de élément X donné (s'il existe, sinon la fonction ne fait rien).

## Exercice 2.2

Soit l'implémentation de `listeSC` par `listeSC_car` avec l'allocation dynamique :

```

cellule= structure
    valeurElement: car;
    pointeurSuivant: ^cellule;
finstructure;
type listeSC_car= ^cellule;

```

La liste vide est représentée par NIL.

La création d'une cellule `c` est effectuée par la fonction `new`,

```

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

```

1. Que fait la fonction suivante :

```

procedure mystere(ref L:listeSC_car, val X: car)
var p: ^cellule
debut
    new(p);

```

```

    p^.valeurElement= X;
    p^.pointeurSuivant= L;
    L= p;
fin;

```

2. Écrire les primitives suivantes :

```

fonction insererEnTete(val X: car, ref L: listeSC_car): vide;
fonction supprimerApres(ref L: listeSC_car, val P: ^cellule): vide;
fonction supprimerEnTete(ref L: listeSC_car): vide;

```

3. Réécrire les fonctions suivantes avec la nouvelle implementation de listeSC par listeSC\_car :

```

fonction dernier(val L: listeSC_car): car;
fonction appartient(val L: listeSC_car, val X: car): boolean;
fonction rang(val L: listeSC_car, val X: car): entier;
fonction ajout_en_queue_sans_doublons(val L: listeSC_car, val X: car) : vide;
fonction supprimer_derniere_occurrence(val L: listeSC_car, val X: car) : vide;

```

## 2 Listes doublement chaînées

Dans les exercices suivants on considèrera le type listeDC, liste doublement chaînée, défini en cours, les primitives étant les mêmes que pour le type listeSC en plus de :

```

fonction dernier(val L: listeDC): ^type_predefini;
fonction precedent(val L: listeDC, val P: ^type_predefini): ^type_predefini;

```

### Exercice 2.3

Soit l'implémentation de listeDC par listeDC\_car avec l'allocation dynamique :

```

cellule= structure
    valeurElement: car;
    pointeurPrecedent: ^cellule;
    pointeurSuivant: ^cellule;
finstructure;
type listeDC_car= ^cellule;

```

1. Écrire les primitives du type listeDC\_car.
2. Écrire les fonctions ajout\_en\_queue\_sans\_doublons et supprimer\_derniere\_occurrence

## 3 Compléments

### Exercice 2.4

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

On donnera deux versions, en utilisant respectivement des listes simplement, implémentées avec des tableaux et doublement chaînées, implémentées avec l'allocation dynamique.