

Algorithmique 1

Feuille 2 : Listes

1 Listes simplement chaînées

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

```
listeSC= liste de type_predefini;
```

défini en cours avec les primitives suivantes :

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

Exercice 2.1

Soit l'implémentation (avec tableau) 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;
    vListe: stockListe;
    premier: curseur;
    premierLibre: curseur;
    cle:curseur;
finstructure;
```

1. Que fait la fonction suivante :

```

fonction mystere(ref L: listeSC_Car, val X: car): boolean;
var p: curseur;
debut
  si listeLibreVide(L) alors
    retourner(Faux)
  sinon
    p= prendreCellule(L);
    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): 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(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;
- (e) fonction supprimer_derniere_occurrence(ref 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;

listeSC_car=structure
  premier:curseur;
  cle:curseur;
finstructure

```

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: curseur;
debut
  new(p);
  p^.valeurElement= X;
  p^.pointeurSuivant= L.premier;
  L.premier= p;
fin;
```

2. Écrire les primitives suivantes :

```
fonction insererEnTete(val X: car, ref L: listeSC_car): vide;
fonction supprimerApres(ref L: listeSC_car): 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): booleen;
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 finListe(val L:liste d'objet):vide;
fonction precedent(val L:liste d'objet): vide;
```

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

1. Écrire les primitives du type listeDC_car.
2. Écrire les fonctions ajout_en_queue_sans_doublons et supprimer_derniere_occurrence

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 ?

3 Compléments

Exercice 2.5

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 chaînées, implémentées avec des tableaux, et doublement chaînées, implémentées avec l'allocation dynamique.