Algorithmique 1

DS 1

Documents **non** autorisés

Exercice 1.1

```
fonction mystere(ref U:tableau[1..N] d'entier): listeSC;
var i, tmp, rest: entier;
    l : ListeSC object;
debut
     creer_Liste(l);
     i = 2;
     tmp = U[1];
     insereEnTete(l, tmp);
     tant que (i <= N) faire
           si tmp < U[i]  alors
                              tmp = U[i] ;
                              insereApres(l,tmp);
           fsi;
           i = i + 1;
     ftq;
     retourner l;
fin;
```

1. Soit $u = 12, 18, 7, 34, 7, 45, 9, 6, 4, 56, 4, 6, 2, 34,$ une suite finie, non vide, d'entiers. Donner la trace d'exécution de `mystere(`u`)`.
2. Que fait la fonction mystere en général ?
3. Ecrire une version récursive de la fonction `mystere`.

Exercice 1.2

Dans l'exercice suivant, on considère 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 et traité en TD.

1. Soit la suite des instructions ci-dessous, représenter l'état de la mémoire après chaque instruction, en supposant le champs `tailleStock = 5` (cfr ANNEXE B).

```
creer_liste(l);
insereEnTete(l,a);
insereEnTete(l,b);
insereApres(l,c);
```

```
insereApres(l,d);
suivant(l);
supprimerApres(l);
```

2. Donner le contenu de la liste

Exercice 1.3

Dans l'exercice suivant, on souhaite traiter le "Problème de Joseph" :« Etant donné un nombre quelconque de personnes, on choisit le gagnant par élimination en procédant de la manière suivante : à chaque tour on élimine la n^{ieme} personne modulo le nombre de ceux qui reste, le gagnant est la dernière personne en piste».
Pour résoudre ce problème. on va utiliser le type abstrait `liste_circulaire` implémenté par `listeDC_car` avec l'allocation dynamique (cfr ANNEXE C).

1. Écrire les primitives suivantes du type `listeDC_car`.

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

   Dans ls suite, on va supposer que les fonctions `insererEnTete` et `supprimerEnTete` sont données.

2. Ecrire la fonction `joseph(val L:liste_circulaire objet , val N: entier):objet;`
   Exemple :
   soit `L={jacques, paul, claude, clotilde, martine, martin, justine, celine}`
   `joseph(L,3)= justine`
   `joseph(L,4)= martin`

Annexe A

```
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

```
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;
```

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

- Gestion de la liste des éléments libres.

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

- 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) : vide;
```

- 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

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

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

Dans le contexte considéré le **type curseur** est le type pointeur vers un élément. 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: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 insererApres(ref L:listeDC_car, val L:objet;) : vide;
  fonction insererEnTete(ref L:listeDC_car, val X:objet) : vide;
  fonction supprimerApres(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, curseur c) : vide;
  ```