Université Bordeaux 1 Licence Informatique 2011/2012

Algorithmique 1

DS 1

Documents **non** autorisés

Exercice 1.1

```
fonction mystere(ref L: listeSC_car): vide;
var p: curseur;
debut
     si (L != NIL et L.premier^.pointeurSuivant != NIL) alors
         L.cle = L.premier;
         tant que (L.cle^.pointeurSuivant != NIL) faire
              p = L.cle^.pointeurSuivant;
              L.cle^.pointeurSuivant = p^.pointeurSuivant;
              p^.pointeurSuivant = L.premier;
              L.premier= p;
         fintq
     finsi
     L.cle = L.premier;
fin
```

1. Soit $L = [12, 18, 7, 9, 6]$, une liste non vide, d'entiers. Donner la trace d'exécution de `mystere(`L`)`.
2. Que fait la fonction mystere en général ?
3. Ecrire une version récursive de la fonction `mystere`.

Exercice 1.2

Dans une implémentation par allocation statique du type liste simplement chaînée, en supposant la constante `tailleStock = 6`,

1. Donner la valeur de champs suivants : L.premier ; L.cle ; L.premierLibre ; L.vListe. après l'instruction suivante `creerListe(L);`.
2. Ensuite on applique successivement les instructions suivantes :

```
insererEnTete(L,C);
insererApres(L,D);
insererEnTete(L,B);
insererApres(L,A);
suivant(L);
supprimerApres(L);
```

(a) donner la valeur de champs suivants :
`L.premier; L.cle; L.premierLibre;  L.vListe`.

(b) quelle est le contenu de la liste ?

Exercice 1.3

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;
   ```

2. Dans une implémentation par allocation dynamique du type liste doublement chaînée, écrire une fonction
   `supprime_toutes_les_occurences_multiples(ref L : listeDC_Car, val X: car): vide;`
   qui supprime dans la liste `L` toutes les occurences de `X` sauf la première, on fera appel aux primitives de `listeDC_car`.

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;
  ```