Contrôle continu 2009-2010
Pointeurs - Récursivité – Listes
(pas de documents autorisés)

Questions de cours (6 points)

- Soit une liste de taille n. Quelle est l'occupation en mémoire d'une liste doublement chainée ?

  Pour chaque cellule, n pointeurs et n objets auquel s'ajoutent la structure de gestion de la liste soit 2 pointeurs.

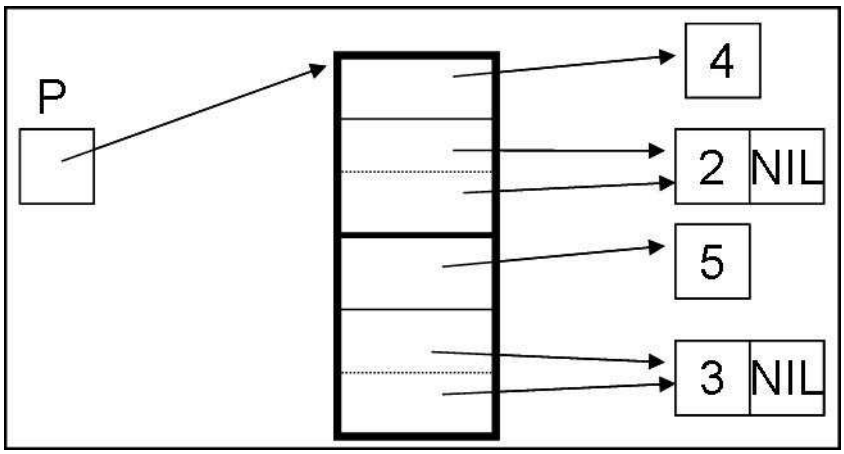
- Ecrire l'algorithme de la primitive insérerAprès dans les listes doublement chainées en implémentation dynamique. **Vu en TD**

```
fonction insérerAprès(ref L:listeSC d'objet, val x:objet):vide;
var nouv:^cellule ;;
 début
    new(nouv);
    nouv^.valeurElement=x;
    nouv^.pointeurSuivant=L.cle^.pointeurSuivant;
    nouv^.pointeurPrécedent=L.cle;
    L.cle^.pointeurSuivant=nouv;
    si nouv^.pointeurSuivant==NIL alors
            L.dernier=nouv ;
    sinon
            nouv^.pointeurSuivant^.pointeurPrecedent=nouv ;
    finsi
 fin
finfonction
```

- Dessinez la mémoire, après la suite d'opérations suivante :

```
objet=structure
        un :^entier ;
        deux : listeSC d'entier ;
var p : ^ tableau[1..2] d'objet ;
new(p) ;
pour i=1 à 2 faire
  new(p^[i].un) ;
  p^[i].un^=i+3 ;
  creerListe(p^[i].deux) ;
  insérerEnTete(p^[i].deux, i+1) ;
finpour
```

On considère que l'implémentation est faite en allocation dynamique.



Dans ce contexte, peut-on écrire

```
var I: entier ;
I=p^[2]^
```

Pourquoi ?

No ! I est une variable de type entier, p^[2]^ est de type objet.

Exercice 1 (6 points)

Ecrire une fonction récursive qui calcule a^n pour *a* réel et *n* entier positif. On suppose qu'on ne dispose pas de la fonction puissance. Votre fonction est-elle récursive terminale ? Pourquoi ?

Première solution

```
fonction puissance(val a :réel ; val n :entier) :réel ;
  début
    si n=1 alors
      retourner(a)
    sinon
     retourner(a*puissance(a,n-1))
    finsi
finfonction
```

Cette fonction n'est pas en forme récursive terminale puisque la dernière opération sera une multiplication.

Deuxième solution

```
fonction puissance(val a :réel ; val n :entier ;ref P :réel) :réel ;
  début
    si n=0 alors
      retourner(P)
    sinon
     retourner(puissance(a,n-1,P*a))
    finsi
finfonction
```

L'appel s'effectue par puissance(a,n,1). Cette fonction est en forme récursive terminale car la dernière opération est un appel à la fonction ou un simple retour d'une variable.

Exercice 2 (8 points)
Ecrire une fonction qui prend en entrée une liste d'entier et qui fournit en sortie une liste telle que tous les nombres multiples de 3 se retrouvent en tête de la liste et les autres éléments sont dans la suite de la liste dans le même ordre que la liste d'entrée. Précisez si la liste de sortie est simplement chainée ou doublement chainée et pourquoi. Donnez la complexité en temps et en mémoire de votre algorithme.

On suppose que l'on dispose d'une fonction accessoire
Fonction estMultiple3(val n :entier) :booléen ;
/ vrai si n est un multiple de 3 */*

Prenons un exemple ! Si la liste d'entrée (1,3,4,6,7,2,9,5) alors la liste de sortie est par exemple (3,6,9,1,4,7,2,5) mais peut-être aussi (9,6,3,1,4,7,2,5).
<u>Première solution</u>
Si la liste est doublement chainée par les entiers non multiples de 3, il suffira d'ajouter après le dernier. On peut donc choisir une liste doublement chainée pour avoir un temps d'exécution plus court. Pour les entiers multiples de 3, l'ordre n'étant pas imposé, on a intérêt à faire un insérerEnTete et donc cela sera plus facile de construire (9,6,3,1,4,7,2,5).

```
Fonction listeTrois (val L :liste d'entier) :listeDC d'entier ;
  Var LSortie :listeDC d'entier ;
  Début
    creerListe(LSortie) ;
    si !listeVide(L) alors
          débutListe(L) ;
           insererEnTete(LSortie,valeur(L)) ;
           suivant(L ) ;
           tantque !estFinListe(L) faire
                  si estMultiple3(valeur(L)) alors
                        insererEnTete(LSortie,valeur(L))
                  sinon
                        finListe(Lsortie) ;
                        insererAprès(LSortie,valeur(L))
                  finsi
                  suivant(L) ;
          fintantque
    finsi
    retourner(LSortie)
  fin
finfonction
```

<u>Deuxième solution</u>
On décide d'utiliser des listes simplement chainé. C'est possible à condition de maintenir dans la clé de la liste un accès vers le dernier. Pour les entiers multiples de 3, l'ordre n'étant pas imposé, on a intérêt à faire un insérerEnTete et donc cela sera plus facile de construire (9,6,3,1,4,7,2,5).

```
Fonction listeTrois (val L :liste d'entier) :listeSC d'entier ;
  Var LSortie :listeDC d'entier ;
  Début
    creerListe(LSortie) ;
    si !listeVide(L) alors
          débutListe(L) ;
           insererEnTete(LSortie,valeur(L)) ;
          devutListe(LSortie) ;
           suivant(L ) ;
           tantque !estFinListe(L) faire
                  si estMultiple3(valeur(L)) alors
                        insererEnTete(LSortie,valeur(L))
                  sinon
                        insererAprès(LSortie,valeur(L)) ;
                        suivant(LSortie) ;
                  finsi
                  suivant(L) ;
          fintantque
    finsi
    retourner(LSortie)
  fin
finfonction
```

Pour les deux solutions

Soit n le nombre d'éléments de la liste d'entrée. La complexité en temps correspond à celle d'une seule boucle tantque donc la complexité en temps est O(n). La complexité mémoire correspond à la liste doublement chainée ou simplement chainée LSortie donc la complexité mémoire est O(n).

Cependant la seconde solution est plus performante en mémoire puisque les listes simplement chainées sont moins encombrantes que les listes doublement chainées.

Primitives et Fonctions

Liste Simplement Chainées

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 créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet ;val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;
fonction detruireListe(ref L:lisetd'objet):vide;</pre>

Liste Doublement Chainées (ajout de)

fonction finListe(ref L:liste d'objet):vide;
fonction précédent(ref L::liste d'objet): vide;

Fonctions sur les listes

fonction estFinListe(val L:liste d'objet):booléen;
fonction appartient(ref L:liste d'objet; ref x:objet): booléen ;