CORRIGE TD10

POINTEURS (1)

Objectif :

- premières notions sur les pointeurs à assimiler (pointeur/élément pointé; parcours de liste chaînée, représentation de listes chaînées)

Durée : 1 séance

Souvent il peut être utile de commencer par un rappel de cours sur la notion de pointeurs avec petits dessins à l'appui pour qu'ils ré-entendent tout avec d'autres mots. A aborder : les notions de pointeurs, éléments pointés, accès si besoin aux champs de l'élément pointé, déclaration de liste chaînée etc.

1. Nombre d'éléments d'une liste

On a les déclarations suivantes :

```
type Element ;
type Liste is access Element ;
type Element is record
    Info : ..............  ;
    Suiv : Liste ;
end record ;
```

Version récursive (la plus simple …. il me semble)

```
function Nombre_Element_Rec(L : in Liste) return Natural is
begin
      if L = null then
            return 0;
      else
            return 1 + Nombre_Element_Rec(L.all.Suiv);
      end if;
end Nombre_Element_Rec;
```

Expliquer l'appel récursif sur la liste privée de son premier élément.

Version itérative :

```
function Nombre_Element_Iter(L : in Liste) return Natural is
      Aux : Liste := L;
      Nb : Natural := 0;
begin
      while Aux /= null loop
            Nb := Nb+1;
            Aux := Aux.all.Suiv;
      end loop;
      return Nb;
end Nombre_Element_Iter;
```

Expliquer le parcours de liste; l'utilisation d'un pointeur auxiliaire pour ce parcours.
Que se passe-t-il si on n'utilise pas ce pointeur annexe ?

- tout d'abord erreur : on veut modifier un paramètre in (L := L.all.suiv)
- si on met L ensuite en in out pour éviter ce problème : à la fin L vaut null et la liste des éléments est dépointée.

2. Ajout d'un élément en fin de liste

On considère le type `Liste` défini dans le cadre de l'exercice 1.

```
procedure Ajouter_Fin(L : in out Liste; E : in Natural) is
      Aux, P : Liste;
begin
      -- si la liste est vide alors
      -- la creer avec cet element
      if L = null then
            L := new Element'(E, null);
      -- sinon : il faut se placer a la fin
      -- et ajouter le nouvel element
      else
            -- se placer sur le dernier element
            Aux := L; -- Aux contient au moins 1 element
            while Aux.all.Suiv /= null loop
              Aux := Aux.all.Suiv;
            end loop;
            -- ajouter l'element en fin
            P := new Element'(E, null);
            Aux.all.Suiv := P;
      end if;
end Ajouter_Fin;
```

Comment modifier la déclaration du type Liste pour simplifier l'insertion en fin de liste ?
Le problème de l'insertion en fin : il faut parcourir toute la liste avant de placer le nouvel élément. Pour simplifier : avoir à la fois un pointeur sur le 1er élément de la liste mais aussi sur le dernier.
Une liste correspond alors `2 variables : début et fin : on va les regrouper dans un record.

La déclaration d'une liste chaînée devient alors :

```
type Element ;
type Lien is access Element ;
type Element is record
    Info : ..............  ;
    Suiv : Lien ;
end record ;

type Liste is record
    Debut : Lien;
    Fin : Lien;
end record;
```

Quand on déclare une variable de type Liste (L : Liste), L.Debut donne accès au 1er élément de la liste et L.Fin au dernier.
Pour insérer en fin : il suffit donc de chaîner le nouvel élément à L.Fin si celle ci existe déjà (sinon il faut la créer) et de déplacer le pointeur vers le dernier élément :

```
procedure Ajouter_Fin(L : in out Liste; E : in Natural) is
      P : Lien;
begin
      P := new Boite'(E, null);
      if L.Debut = null then
            L.Debut := P;
            L.Fin := P;
      else
            L.Fin.suiv := P;
            L.Fin := P;
      end if;
end Ajouter_Fin;
```

3. Eclatement de liste

On dispose d'une liste de nombres entiers. On veut éclater cette liste en une liste de nombres pairs et une liste de nombres impairs. L'ordre des nombres dans ces listes est indifférent.
On suppose que la déclaration est celle de l'exercice 1.

Réfléchir à différentes solutions (faire des schémas explicatifs):

1. recopier les éléments pairs pour les insérer dans une liste et même chose pour les éléments impairs. La liste initiale n'est pas modifiée. L'algorithme doit nécessairement comporter des new … puisqu'il y aura création de nouveaux éléments.
2. modifier le chaînage de la liste initiale. La liste initiale sera perdue ; les éléments ne sont pas recrées mais chaînés différemment.
3. faire une liste de liste.

On va présenter en correction la 2ème solution. Discuter si on fait une insertion en tête ou en fin dans les listes pairs et impairs. L'insertion en tête est possible puisque l'ordre est indifférent.

Insertion en tête : pas de cas particulier (liste vide ou pas); pas besoin de parcourir la liste pour trouver la fin. Quand on a le choix, c'est en général l'insertion que l'on choisit. Ce qui donne avec les déclarations du sujet exo1 :

```
Procedure Eclater ( L : in out Liste; Pairs :out Liste ; Impairs :out Liste) is
        Le_suivant : liste ;
Begin
      Pairs := null;
      Impairs :=null;
      Aux := L;
      While Aux /=null loop
        Le_Suivant :=Aux.all.suiv ; -- car on va perdre le chainage
        If  Aux.all.info rem 2 = 0 then
            -- ajout en debut de la liste des pairs
              aux.all.suiv:=Pairs;
            Pairs := Aux;
        else
                -- ajout en debut de la liste des impairs
              aux.all.suiv:=Impairs;
            Impairs := Aux;
        End if;
        Aux := Le_Suivant; -- pour passer a l element suivant
End loop;
End Eclater ;
```

Si insertion en fin : faire une mémorisation du dernier élément). **Il est donc préférable d'utiliser la déclaration vue à l'exercice 2 pour simplifier l'insertion en fin**.

```
procedure Eclater(L : in Liste; L_Pair, L_Impair : out Liste) is
      Aux : Lien := L.Debut;
begin
      L_Pair.Debut := null;
      L_Pair.Fin := null;
      L_ImPair.Debut := null;
      L_ImPair.Fin := null;

      while Aux /= null loop
            if Aux.all.Info rem 2 = 0 then
```

```
                -- element pair dans liste des pairs
                -- avec insertion en fin
                if L_Pair.Debut = null then
                      L_Pair.Debut := Aux;
                      L_Pair.Fin := Aux;
                      Aux := Aux.all.Suiv;
                      L_Pair.Fin.all.Suiv := null;
                else
                      L_Pair.Fin.all.Suiv := Aux;
                      L_Pair.Fin := L_Pair.Fin.all.Suiv;
                      Aux := Aux.all.Suiv;
                      L_Pair.Fin.all.Suiv := null;
                end if;

          else
                -- elements impairs dans listes des impairs
                -- avec insertion en fin
                if L_Impair.Debut = null then
                      L_Impair.Debut := Aux;
                      L_Impair.Fin := Aux;
                      Aux := Aux.all.Suiv;
                      L_Impair.Fin.all.Suiv := null;
                else
                      L_Impair.Fin.all.Suiv := Aux;
                      L_Impair.Fin := L_Impair.Fin.all.Suiv;
                      Aux := Aux.all.Suiv;
                      L_Impair.Fin.all.Suiv := null;
                end if;
          end if;
     end loop;
end Eclater;
```

Remarque : on ne peut pas utiliser la procédure ajouter_fin car celle-ci crée à chaque fois un nouvel élément (les éléments seraient dupliqués).