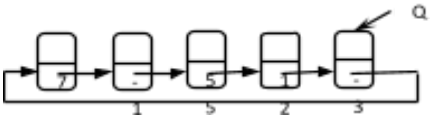
Liste circulaire

Une liste circulaire est une liste telle que **le dernier élément de la liste a pour successeur le premier élément**. On peut ainsi parcourir toute la liste à partir de n'importe quel élément.
Il est plus avantageux de remplacer l'indication sur le premier élément par une indication sur le dernier élément, ce qui donne facilement accès au dernier et au premier élément.

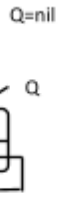
Déclaration
Type **ListC** = ^**Cellule**
Type **Cellule** : **Enregistrement**
info : <Type_elt> ;
svt : **ListC** ;
Feng ;



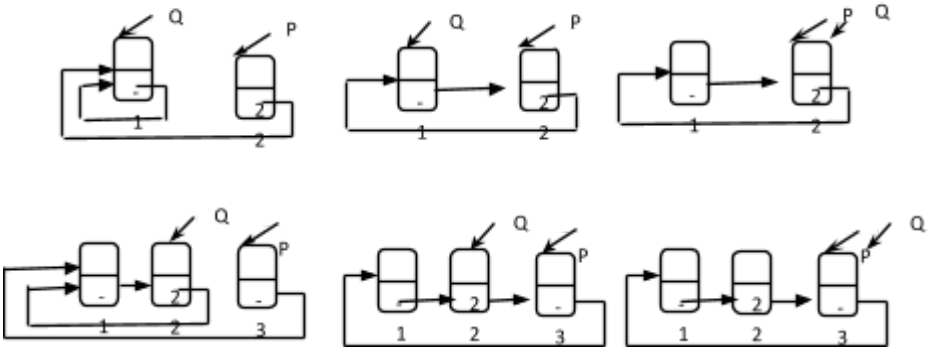
Exemple

Debut
T : Tab[3] **entier** ;
Q, P : **ListC** ;
i : **entier** ;

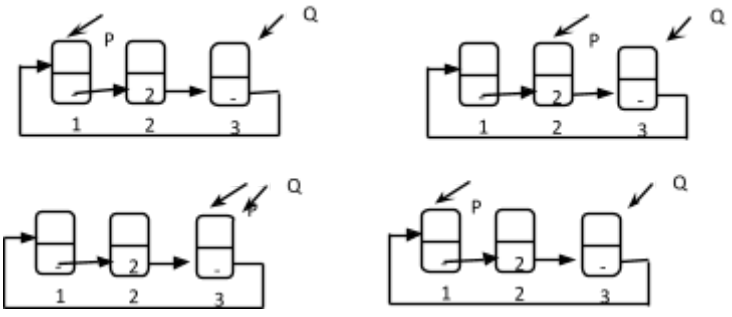
Q nil ;
T[1]=-1 ; T[2]=22 ; T[3]= -3 ;
allouer (Q) ;
Q^.info T[1] ;
Q^.svt Q ;



Pour (i 2 à 3)
Faire
Allouer (P) ;
P^.info T[i] ;
P^.svt Q^.svt ;
Q^.svt P;
Q P ;
Fait
Fin;

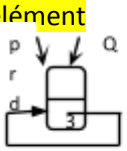


Affichage d'une liste Circulaire
Procédure Affiche_ListC(E/ Q : **ListC**)
Début
P : **ListC** ;
Si (Q ≠ nil) **alors** P Q^.svt ;
Répéter
Ecrire (P^.info) ;
P P^.svt;
Jusque(P=Q^.svt);
Sinon
Ecrire ("la liste est vide");
Fsi ;
Fin ;

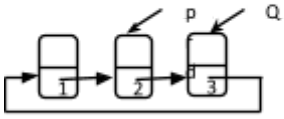


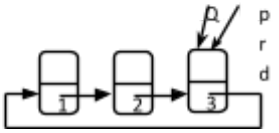
Suppression d'un élément après une adresse donnée

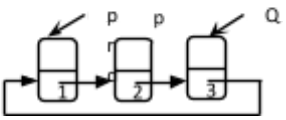
Procédure Suppression_apres_ListC (ES/ Q : **ListC**, ES/ Prd : **ListC**)
Début
P : **ListC** ;
Si (Prd=Q) **et** (Q^.svt=Q) **alors** //une liste avec un seul élément
Libérer (Q) ;
Q nil ;
Prd <- nil



Sinon
Si (prd^.svt=Q) **alors** Q prd ; **Fsi** ; //supp de l'élément d'adresse Q
P prd^.svt ;
prd^.svt P^.svt ;
Libérer (P) ;



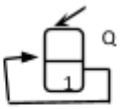


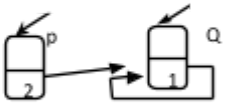


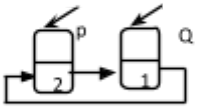
Fsi ;
Fin ;

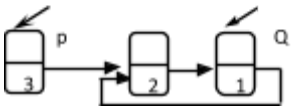
Ajout d'un élément en tête

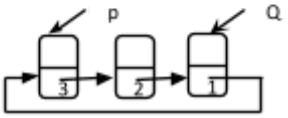
Procedure ajout_tete_ListeC(ES/ Q :**ListeC**, E/ e :**entier**)
Debut
p : **ListeC** ;
si(Q=nil) **alors** Q<- créer_nouedLC() ;
Q^.info <- e ;
Q^.svt <- Q ;
sinon p<- créer_nouedLC() ;
p^.info <- e ;
p^.svt <- Q^.svt ;
Q^.svt <- p;
Fsi ;
Fin





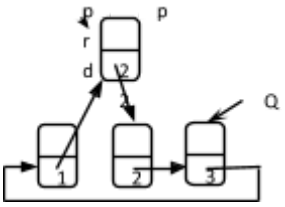


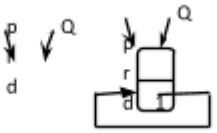


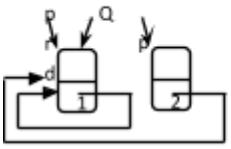


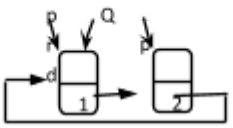
Ajout d'un élément après une adresse donnée

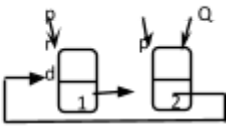
Procédure ajout_apres_ListC (ES/ Q, prd : **ListC**, E/ c : **Type_elt**)
Début
p : **ListC** ;
si (Q=nil) **alors** // la liste est vide
Q <- créer_noeudLC () ;
Q^.info c ;
Q^.svt Q ;
Prd <- Q ;
Sinon // deux cas
p <- créer_noeudLC () ;
p^.info c ;
p^.svt prd^.svt ;
prd^.svt p;
si (prd=Q) **alors** Q <- p ; **fsi**
prd p ;
Fsi ;
Fin ;

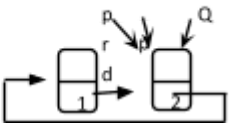






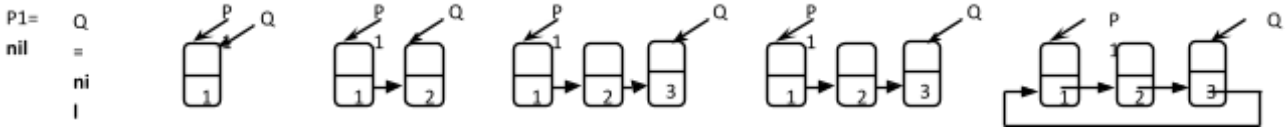




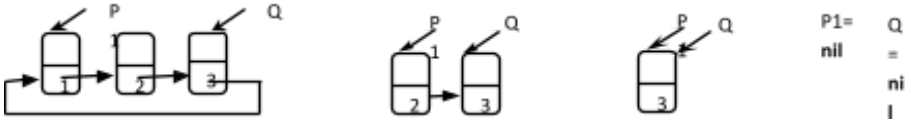


Remarque : à chaque insertion/suppression d'un élément d'une liste circulaire, la liste doit vérifier la définition d'une liste circulaire.

Cas d'insertion



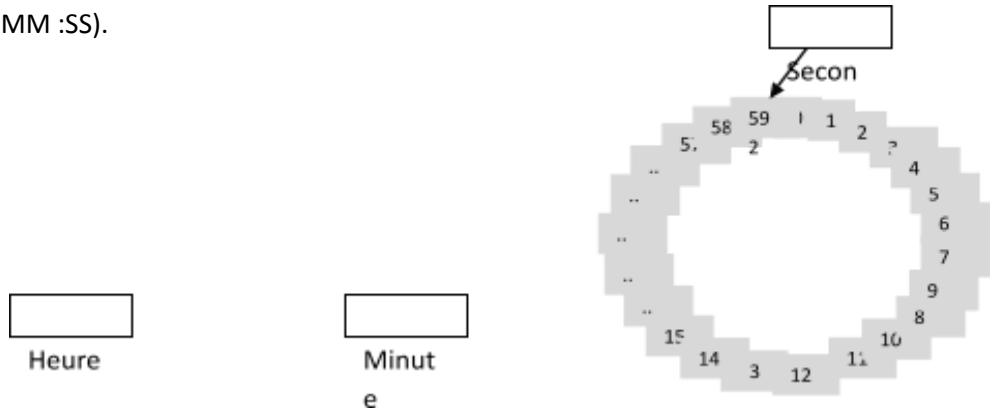
Cas de suppression



Exercice : Le but de cet exercice est de **simuler** le fonctionnement d'une horloge avec deux méthodes différentes. Nous adoptons le principe donné dans chaque partie.

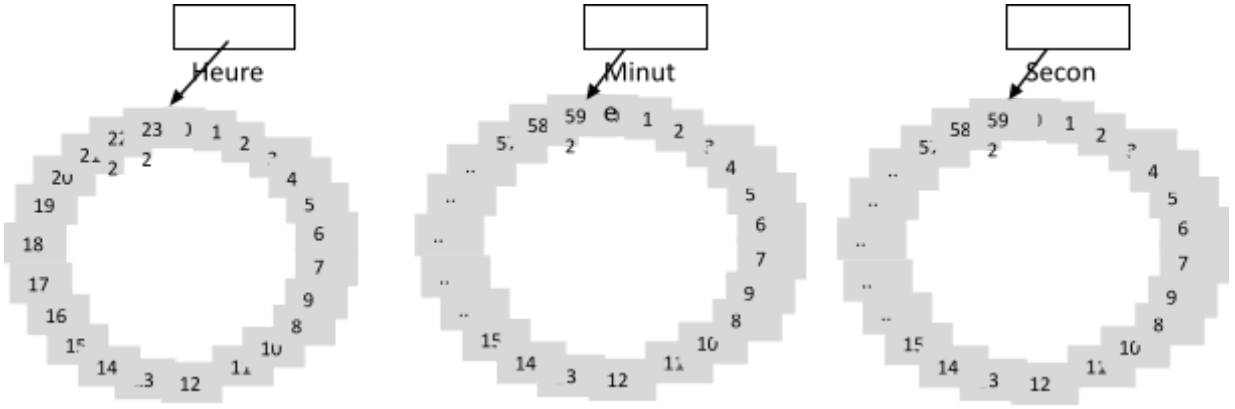
Partie 1 : on veut afficher heure, minute et seconde à partir **d'une liste circulaire** représentant les secondes (0,1,…,59) d'une horloge.

1. Définir le type de cette liste.
2. Ecrire une action Const_List_Circ( ) qui construit la liste circulaire : **Seconde**.
3. Utiliser la liste circulaire et deux variables : **minute** et **heure** pour afficher heure, minute et seconde (HH :MM :SS).



Partie 2 : on veut afficher heure, minute et seconde à partir de trois listes circulaires représentant heure, minute et seconde respectivement.

1. Construire les trois listes : **Heure**, **Minute** et **Seconde**.
2. Utiliser les trois listes pour simuler le fonctionnement d'une horloge (HH :MM :SS).



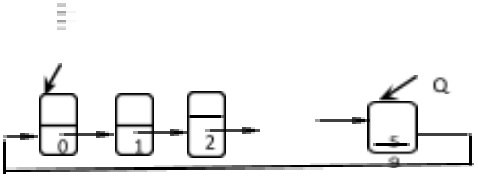
Solution

Partie 1

```
Type     ListeC = ^Cellule
Type     Cellule : Enregistrement
           info : entier ;
           svt : ListeC ;
         Feng ;
```

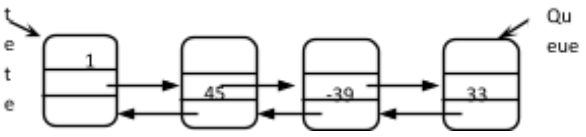
Fonction Const_ListeC(E/ n : **entier**) : **ListeC** **Debut** Q : **ListeC** ; Q<- nil ; **Pour** (i<- n-1 à 0) **Faire** ajout_tete_ListeC(Q , i) ; **Fait ;** Retourne (Q) ; **Fin**	**Fonction** Const_ListeC(E/ n : **entier**) : **ListeC** **Debut** Q, prd : **ListeC** ; Q<- nil ; prd<-nil ; **Pour** (i<- 0 à n-1) **Faire** ajout_apres_ListC (Q , prd, i) ; **Fait ;** Retourne (Q) ; **Fin**

```
                                          H  M   S
Début                                     0  0   0
  Q, cour : ListC ;
  nb, minute, heure : entier ;
  nb<- 60 ;
  b : booleen ;
b<- vrai ;
  Q <- Const_ListeC(nb) ;
 minute <- 0 ;    heure <- 0 ;

 Tant que (b=vrai ) // boucle infinie
  Faire
    cour <- Q^.svt ;
    répéter
       écrire (heure, minute, cour^.info)
       cour <- cour^.svt ;
    jusqu'à(cour =Q ^.svt) ;

    si (minute=59) alors minute <- 0 ;
                        si(heure = 23) alors heure <- 0
                                       sinon heure <- heure +1 ;
                        fsi ;
                   sinon minute <- minute +1 ;
   fsi ;
  fait ;
fin ;
```



Liste symétrique

Une liste symétrique ou bidirectionnelle est une liste telle que chaque élément désigne l'élément suivant et l'élément précédent. L'intérêt de ce type de liste réside dans le fait qu'il est facile d'extraire un élément à partir de sa place. Il n'est pas nécessaire de parcourir la liste pour retrouver le précédent.



Déclaration

```
Type     ListSy = ^Cellule
Type     Cellule : Enregistrement
           info : <Type_elt> ;
           svt : ListSy ;
           prd : ListSy ;
       Feng ;
```

Ajouter un élément en tête de liste

```
Procédure Ajout_tete_ListeSy(ES/ tete : ListSy , E/ x : Type_elt)
Debut
  nouv : ListSy;

     nouv <- créer_noeudLSy () ;
     nouv^.info    x ;
     nouv^.svt    tete ;
     nouv^.prd    nil ;
     si(tete ≠ nil) alors tete^.prd    nouv ; Fsi
     tete    nouv ;
Fin ;
```

Ajouter un élément après une adresse

```
Procédure Ajout_apres_ListeSy(ES/ prd : ListSy , E/ x : Type_elt)
Debut
  P, nouv : ListSy;

     p <- prd^.svt ;
     nouv <- créer_noeudLSy () ;
     nouv^.info    x ;
     nouv^.svt     prd^.svt ;
     prd^.svt     nouv;
     nouv^.prd    prd ;
     si (p ≠ nil) alors p^.prd    nouv ; Fsi
     prd    nouv ;
Fin ;
```

Suppression d'un élément en tête de liste

```
Procédure Supp_tete_ListeSy(ES/ tete : ListSy)
Debut
  P: ListSy;

      p <- tete ;
      tete=     tete^.svt ;
      si (tete ≠ nil) alors tete^.prd     nil ; Fsi
     liberer(p) ;
Fin ;
```

Suppression d'un élément se trouvant à une adresse p

```
Procédure Suppression_element (ES/ p : ListSy)
Debut
  t, prd : ListSy;

      prd <- p^.prd ;
      t     p^.svt ;
      prd^.svt     t ;

     si (p^.svt ≠ nil) alors t^.prd     prd ; Fsi
     Liberer (p) ;
Fin ;
```

Affichage d'une liste symétrique

```
Procédure Affiche_ListeSy (E/ p : ListSy)
Debut

      Tant que (p ≠ nil)
        Faire
          Ecrire (p^.info) ;
          p     p^.svt ;
       Fait ;
Fin ;
```

Exemple de création d'une liste symétrique :

Création d'une liste symétrique LIFO

```
Début
  tete : ListSy ;
  i, n, x : entier ;

  tete <- nil ;
  Lire (n) ;
  Pour (i<-1 à n)
   faire
     Lire(x) ;
     Ajout_tete_ListeSy(tete  , x) ;
  Fait ;
Fin ;
```

Création d'une liste symétrique FIFO

```
Début
   prd, tete : ListSy ;
   x, i, n : entier ;

   tete <- nil ;
   Lire (x, n) ;
   Ajout_tete_ListeSy(tete , x) ;
   prd<- tete ;
   Pour (i<-2 à n)
    faire
      Lire(x) ;
      Ajout_apres_ListeSy(prd  , x) ;
   Fait ;
Fin ;
```

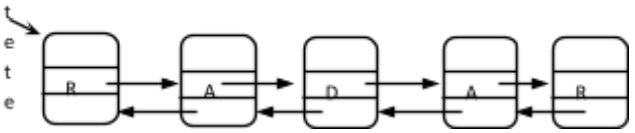
Exercice :

Un mot étant représenté par une liste symétrique de caractères.

01/ Créer la liste symétrique.

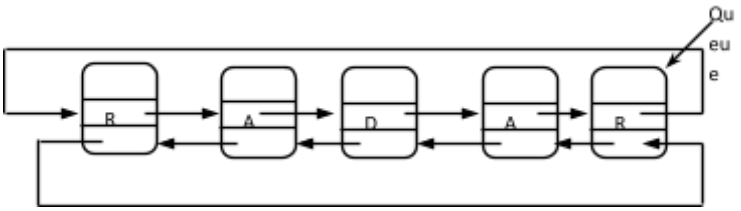
2/ Ecrire une fonction qui indique si un mot est palindrome ou non. Un mot est palindrome si l'ordre de ses lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche. Exemple : radar, kayak, elle,…

Exemple : pour le mot=RADAR on obtient :



Liste circulaire bidirectionnelle

C’est une liste telle que chaque élément désigne l’élément suivant et l’élément précédent et le dernier élément de la liste a pour successeur le premier élément.



Exercice : donner les actions qui manipulent ce type de liste.