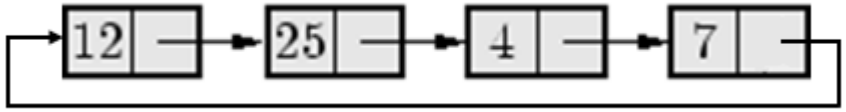
Chapitre5. LES LISTES CHAÎNÉES CIRCULAIRES

- DÉFINITION ET REPRESENTATION
- LISTE SIMPLEMENT CHAÎNÉES CIRCULAIRES
- LISTES DOUBLEMENT CHAÎNÉES CIRCULAIRES
- EXERCICES

2. Listes Simplement Chaînées Circulaires

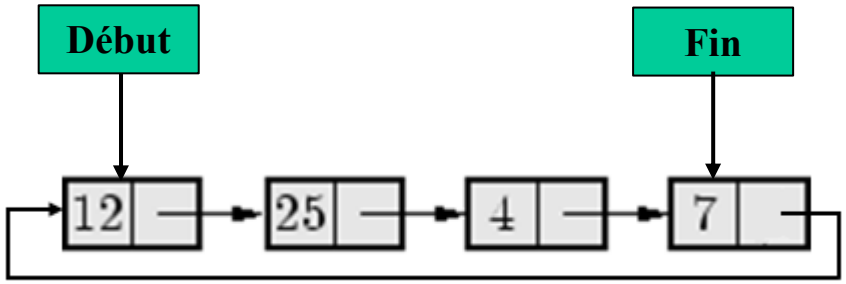
- Les Listes Simplement Chaînées Circulaires (LSCC) sont des Listes Simplement Chaînées, sauf que le dernier élément de la liste pointe sur le premier élément de la liste.



- La liaison entre les éléments se fait grâce à un pointeur
- Le pointeur suivant du dernier élément doit pointer vers le début de la liste (le premier élément de la liste).
- Pour accéder à un élément, la liste est parcourue en commençant avec le début (tête), le pointeur suivant permettant le déplacement vers le prochain élément.

2. Listes Simplement Chaînées Circulaires

- Pour avoir le contrôle de la liste, il est préférable de sauvegarder certains éléments : le premier élément debut, le dernier élément fin, le nombre d'éléments taille.



2. Listes Simplement Chaînées Circulaires

 Parmi les opérations nécessaires pour manipuler une liste simplement chaînée circulaire, on trouve :

- Initialisation
- Insertion d'un élément dans la liste
  - Insertion dans une liste vide
  - Insertion au début de la liste
  - Insertion à la fin de la liste
  - Insertion ailleurs dans la liste
- Suppression d'un élément dans la liste
  - Suppression au début de la liste
  - Suppression ailleurs dans la liste
- Affichage de la liste
- Destruction de la liste

2. Listes Simplement Chaînées Circulaires

- Prototype d'un élément de la liste

```
typedef struct ElementListe {
        int donnee;
        struct ElementListe *suivant ; } Element;
```

- Initialisation

```
void initialisation ( ) {
        debut = NULL; fin = NULL; taille = 0;
      }
```

2. Listes Simplement Chaînées Circulaires

Insertion d'un élément dans la liste

- Insertion dans une liste vide

```
ins_dans_liste_vide (int i) {
    Element *element;
    element = (Element *) malloc (sizeof (Element));
    element-> donnee = i ;
    element-> suivant = element; //circulaire
    debut = element;  fin = element;

    taille++;
}
```

2. Listes Simplement Chaînées Circulaires

Insertion d'un élément dans la liste

- Insertion au début de la liste

```
ins_debut_liste (int i) {
    Element *element;
    element = (Element *) malloc (sizeof (Element));
    element-> donnee = i ;
    element-> suivant = debut;
    debut = element;
    fin->suivant = debut; //circulaire
    taille++;
}
```

2. Listes Simplement Chaînées Circulaires

Insertion d'un élément dans la liste

- Insertion à la fin de la liste

```
ins_fin_liste (int i) {
    Element *element;
    element = (Element *) malloc (sizeof (Element));
    element-> donnee = i ;
    element-> suivant = NULL;
    fin->suivant = element;
    element->suivant=debut; //circulaire
    fin=element;
    taille++;
}
```

2. Listes Simplement Chaînées Circulaires

Insertion d'un élément dans la liste

- Insertion ailleurs de la liste

```
ins_liste (int i, int pos) {
        Element *courant;
        Element *element;
        int t ;
        element = (Element *) malloc (sizeof (Element));
        element-> donnee = i ;
        element-> suivant = NULL ;
        courant = debut;
        for (t = 1; t < pos; t++)    courant = courant->suivant;
        element->suivant = courant->suivant;
        courant->suivant = element;
        taille++;
}
```

2. Listes Simplement Chaînées Circulaires

Suppression d'un élément dans la liste

- Suppression au début de la liste

```
int supp_debut ( ) {
Element *supp_element;
if (taille == 0)    return -1;
supp_element = debut;
debut = debut-> suivant;
fin->suivant = debut ;   //Circulaire
if (taille == 1)    fin = NULL;
free(supp_element);
taille--;
return 0;
}
```

2. Listes Simplement Chaînées Circulaires

Suppression d'un élément dans la liste

- Suppression ailleurs de la liste

```
int supp_dans_liste (int pos) {
int i;  Element *courant,*precedent , *supp_element;
if (taille <= 1 || pos < 1 || pos > taille)    return -1;
courant = debut;
for (i = 1; i < pos; i++)
{precedent=courant; //Mémoriser l'élt avant courant
courant = courant->suivant; }
supp_element = courant; //élt à supprimer
precedent->suivant=courant->suivant; //Suppression de courant
```

2. Listes Simplement Chaînées Circulaires

Suppression d'un élément dans la liste

- Suppression ailleurs de la liste (suite)

```
if(pos == taille) //élt supprimé existe à la fin
      { precedent->suivant=debut; //Circulaire
        fin=precedent;
       }
free (supp_element);
taille--;
return 0;
}
```

2. Listes Simplement Chaînées Circulaires

Affichage de la liste

```
void affiche () {
Element *courant;
courant = debut;
int i=1;
while (i<=taille) {
      printf ("%d ", courant->donnee);
      i++;
      courant=courant->suivant;
    }
}
```

2. Listes Simplement Chaînées Circulaires

Destruction de la liste

```
void detruire () {
while (taille > 0)     supp_debut();
}
```

2. Listes Simplement Chaînées Circulaires

Exercice d'application

On considère une liste simplement chainée circulaire contenant des entiers.

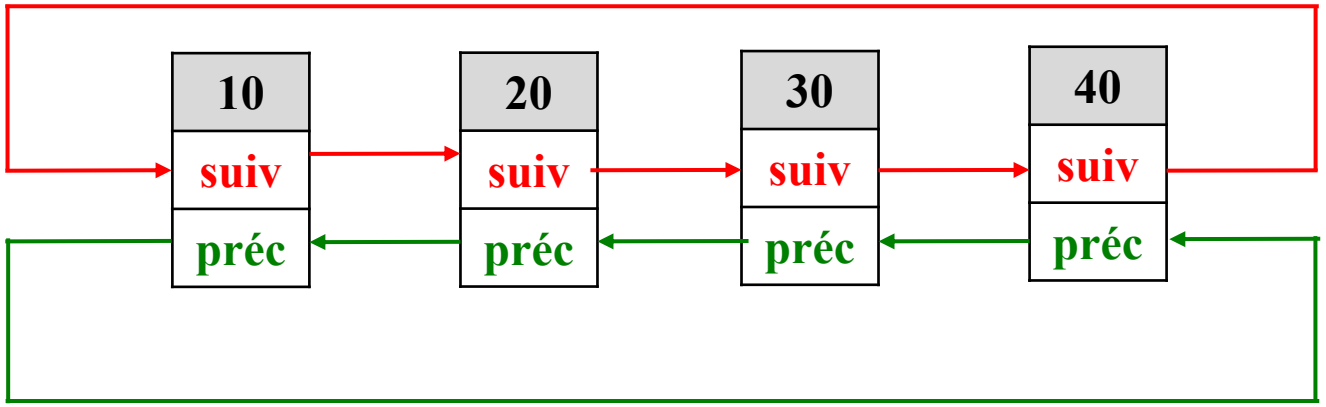
1- Donner les déclarations et les fonctions nécessaires pour manipuler une liste de type (LSCC).

2- En se basant sur le principe de manipulation d'une liste de type (LSCC), ajouter une fonction qui permet de supprimer les entiers négatifs.

3- Rédiger la fonction main() pour le test

3. Listes Doublement Chaînées Circulaires

 Les Listes Doublement Chaînées Circulaires (LDCC) sont des Listes Doublement Chaînées, sauf que le champ suivant du dernier élément de la liste pointe sur le premier élément et le champ précédent du premier élément de la liste pointe sur le dernier élément.



3. Listes Doublement Chaînées Circulaires

 Parmi les opérations nécessaires pour manipuler une liste Doublement chaînée circulaire, on trouve :

- Initialisation
- Insertion d'un élément dans la liste
  - Insertion dans une liste vide
  - Insertion au début de la liste
  - Insertion à la fin de la liste
  - Insertion avant un élément de la liste
  - Insertion après un élément de la liste
- Suppression d'un élément dans la liste
  - Suppression un élément de la liste
- Affichage de la liste
- Destruction de la liste

3. Listes Doublement Chaînées Circulaires

- Prototype d'un élément de la liste

```
typedef struct ElementListe {
        int info ;
        struct ElementListe *precedent ;
        struct ElementListe *suivant ; } Element;
```

- Initialisation

```
void initialisation ( ) {
        Debut = NULL; Fin = NULL; taille = 0;
      }
```

3. Listes Doublement Chaînées Circulaires

Insertion dans une liste vide

```
int ins_dans_liste_vide (int info)
{
Element *nou_element;
if ((nou_element = (Element*) malloc (sizeof(Element)))
==NULL)     return -1;
nou_element-> info=info;
nou_element->precedent = nou_element; //Circulaire
nou_element->suivant = nou_element;   //Circulaire
Debut = nou_element;
Fin = nou_element;
Taille++;
return 0;
}
```

3. Listes Doublement Chaînées Circulaires

Insertion au début de la liste

```
int ins_debut_liste (int info)
{
Element *nou_element;
if ((nou_element = (Element*) malloc (sizeof (Element)))
==NULL)     return -1;
nou_element->info=info;
nou_element->suivant = Debut;
Debut->precedent = nou_element;
nou_element->precedent = Fin; //Circulaire
Fin->suivant=nou_element;     //Circulaire
Debut = nou_element;
Taille++;
return 0;
}
```

3. Listes Doublement Chaînées Circulaires

Insertion à la fin de la liste

```
int ins_fin_liste (int info)
{
Element *nou_element;
if ((nou_element = (Element*) malloc (sizeof (Element)))
==NULL)     return -1;
nou_element->info=info;
nou_element->precedent = Fin;
Fin->suivant = nou_element;
nou_element->suivant = Debut; //Circulaire
Debut->precedent=nou_element; //Circulaire
Fin = nou_element;
Taille++;
return 0;
}
```

3. Listes Doublement Chaînées Circulaires

Insertion avant un élément de la liste

```
int ins_avant (int info, int pos)
{ int i;
Element *nou_element, *courant;
if ((nou_element = (Element*) malloc (sizeof (Element))) ==NULL)
        return -1;
nou_element->info=info;
courant = Debut;
for (i = 1; i < pos; ++i) courant = courant->suivant;
courant->precedent->suivant=nou_element;
nou_element->precedent=courant->precedent;
nou_element->suivant = courant;
courant->precedent=nou_element;
if(pos==1) {Debut=nou_element;Debut->precedent=Fin;
Fin->suivant=Debut;}    //Circulaire
Taille++;       return 0;
```

3. Listes Doublement Chaînées Circulaires

Insertion après un élément de la liste

```
int ins_apres (int info, int pos)
{int i;
Element *nou_element, *courant;
if ((nou_element = (Element*) malloc (sizeof (Element))) ==NULL)
return -1;
nou_element->info=info;
courant = Debut;
for (i = 1; i < pos; ++i) courant = courant->suivant;
nou_element->suivant = courant->suivant;
courant->suivant->precedent=nou_element;
nou_element->precedent = courant;
courant->suivant=nou_element;
if(pos==Taille) {Fin=nou_element;Fin->suivant=Debut;
Debut->precedent=Fin;}//Circulaire
Taille++;       return 0;
```

3. Listes Doublement Chaînées Circulaires

Suppression d'un élément de la liste

```
int supp (int pos)
{int i;
Element *supp_element,*courant;
if(Taille == 0) return -1;
if(pos == 1) { /* suppression de 1er élément */
    supp_element = Debut;
    Debut = Debut->suivant;
    if(Taille==1){Debut=NULL;Fin=NULL;}
    else {Debut->precedent = Fin;Fin->suivant=Debut;}//Circulaire
    }
else if(pos == Taille) { /* suppression du dernier élément */
                         supp_element = Fin;
                         Fin->precedent->suivant =
Debut;//Circulaire
                         Fin = Fin->precedent;
                         Debut->precedent=Fin; //Circulaire
                         }
```

3. Listes Doublement Chaînées Circulaires

Suppression d'un élément de la liste (Suite)

```
else {  /* suppression ailleurs */
                courant = Debut;
                for(i=1;i<pos;++i) courant = courant->suivant;
                supp_element = courant;
                courant->precedent->suivant = courant->suivant;
                courant->suivant->precedent = courant->precedent;
      }
free(supp_element->info);
free(supp_element);
Taille--;
return 0;
}
```

3. Listes Doublement Chaînées Circulaires

Afficher la liste entière

```
affiche()
{ /* affichage en avançant */
        Element *courant;
        int i;
        courant = Debut; /* point du départ le 1er élément */
        for(i=1;i<=Taille;i++)
        {
                printf("%d ", courant->info);
                courant = courant->suivant;
        }
}
```

3. Listes Doublement Chaînées Circulaires

Destruction de la liste

```
detruire ()
{
while (Taille > 0)     supp(1);
}
```

3. Listes Doublement Chaînées Circulaires

Exercice d'application

On considère une liste Doublement chainée circulaire contenant des entiers.

1- Donner les déclarations et les fonctions nécessaires pour manipuler une liste de type (LDCC).

2- Ajouter une fonction permettant la recherche du minimum et du maximum.

3- Rédiger la fonction main() pour le test