Chapitre 5
Les listes doublement chaînées / circulaires

Module 20 (info4): Structures de données en C

2éme ANNEE LEESM

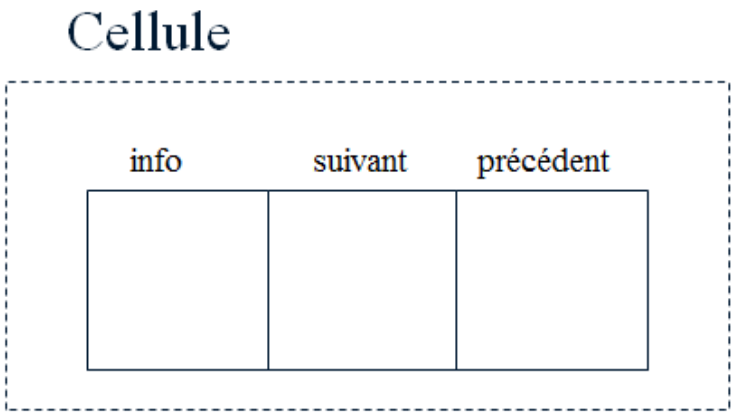
mlahby@gmail.com

5 mai 2021

Plan

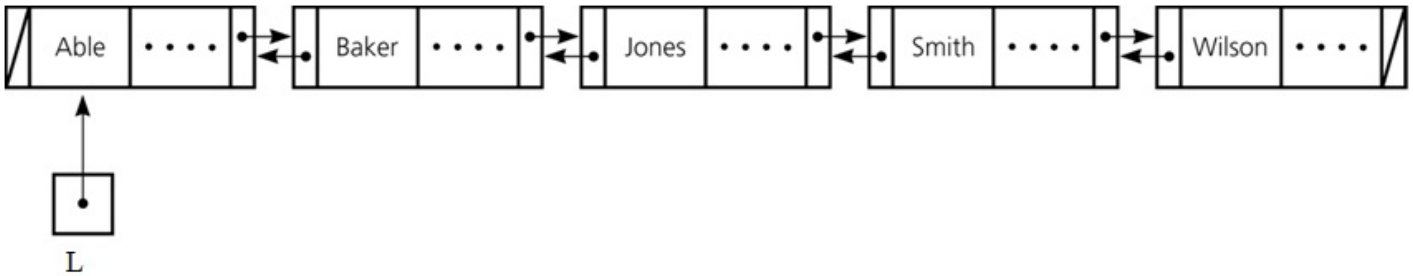
Définition

- Une liste doublement chaînée est une structure de données dynamique contenant une séquence finie des cellules.
- La cellule est la composante principale d'une liste doublement chaînée ;
- Une cellule est une structure qui comporte trois champs :
  - **Le champ info** : il contient des informations sur l'élément représenté par la cellule ;
  - **Le champ suivant** : il représente un pointeur qui contient l'adresse de la cellule suivante.
  - **Le champ précédent** : il représente un pointeur qui contient l'adresse de la cellule précédente.



Applications

- Doubly linked lists are useful for playing video and sound files with “rewind” and “instant replay”
- They are also useful for other linked data which require “rewind” and “fast forward” of the data
- Editeur de texte
- Algorithmes de retour sur trace, essais/erreurs (backtracking)
- Implantation des types de données abstraits :
  - piles (dernier entré - premier sorti)
  - files d’attente (premier entré - premier sorti)
  - double-queues
  - ensembles, tables de hachage



Liste simplement chaîneé vs. liste doublement chaînée

- **Points communs :**
  - Structures permettant de stocker une collection de données de même type.
  - L'espace mémoire utilisé n'est pas contigué.
  - La taille est inconnue à priori ;
  - Une liste doublement chaînée est constituée de cellules qui sont liées entre elles par des pointeurs.
  - Pour accéder à un élément quelconque d'une liste, il faut parcourir la liste jusqu' à cet élément.
- **Différences :**
  - Une liste doublement chaînée n'est PAS récursive !
  - On peut accéder directement au premier et dernier élément
  - On peut parcourir la liste dans les 2 sens (on peut donc revenir en arrière).

Naturelle avec un seul pointeur

- 1 cellule / élément
- une liste = pointeur sur le 1^{er} élément
- fin = NULL

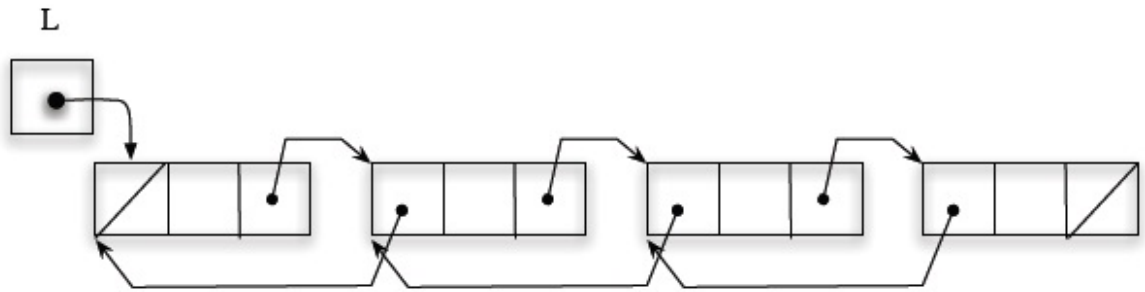
Syntaxe pour définifir une liste doublement chaînée

```
/* type des dcellules d'une liste */
typedef struct cellule{
    int info; //le champ info peut avoir n'importe quel type
    struct cellule *suiv; //pointeur contenant l'adresse de la cellule suivante
    struct cellule *prec; //pointeur vers la cellule précédente
}DListe;
```

Naturelle avec un seul pointeur

Déclaration d'une liste doublement chaînée

- Pour déclarer une variable de type liste doublement chaînée, il suffit de déclarer un pointeur sur le premier élément :
- Exemple : DListe * L ;



Avec deux pointeurs First/Last

- 1 cellule / élément
- une liste = un pointeur sur le **premier** élément + un poiteur sur le **dernier** élément
- fin = NULL

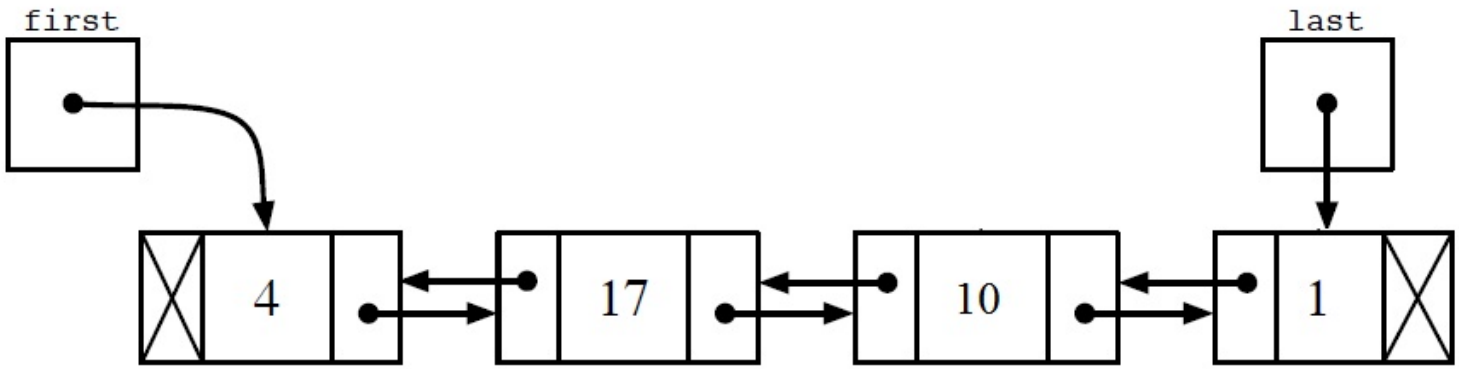
Définition d'une liste doublement chaînée

```
/* type DCellule d'une liste */
typedef struct cellule{
    int info; //le champ info peut avoir n'importe quel type
    struct cellule *suiv; //pointeur contenant l'adresse de la cellule suivante
    struct cellule *prec; //pointeur vers la cellule précédente
}DCellule;
/* type des listes doublement chainées*/
typedef struct {
    DCellule *first; //pointeur contenant l'adresse de la première cellule
    DCellule *last; //pointeur contenant l'adresse de la derniere cellule
}Dliste;
```

Avec deux pointeurs First/Last

Déclaration d'une liste doublement chaînée

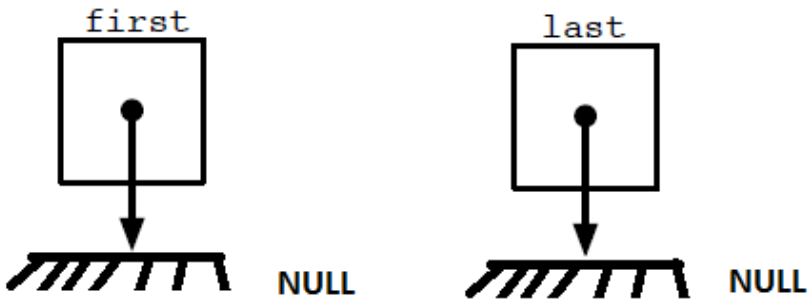
- Pour déclarer une variable de type liste doublement chaînée, il existe deux possibilités :
  - En utilisant une variable statique : Dliste L.
  - En utilisant une variable dynamique : Dliste *L.
- La variable L est une structure contenant deux champs : L.First et L.Last



Créer une liste vide :Dliste CreerListeVide()

Définition de la fonction

```
Dliste CreerListeVide()
{
      Dliste L ;
      L.first=NULL ;
      L.last=NULL ;
      return(L) ;
}
```



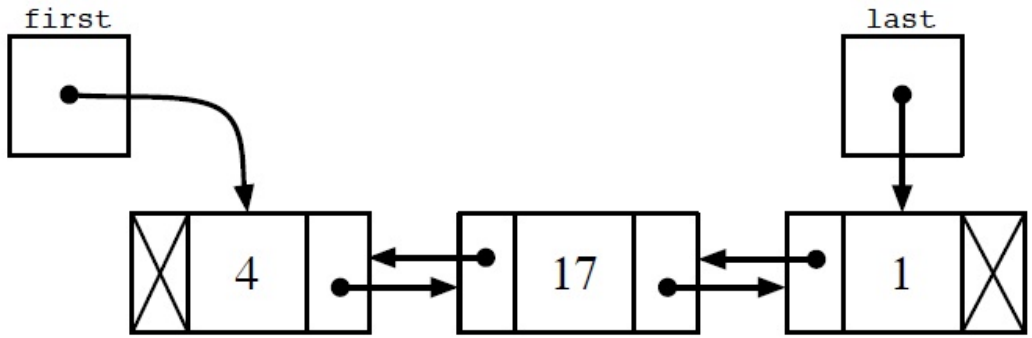
Tester si la liste est vide : int EstVide(Dliste L)

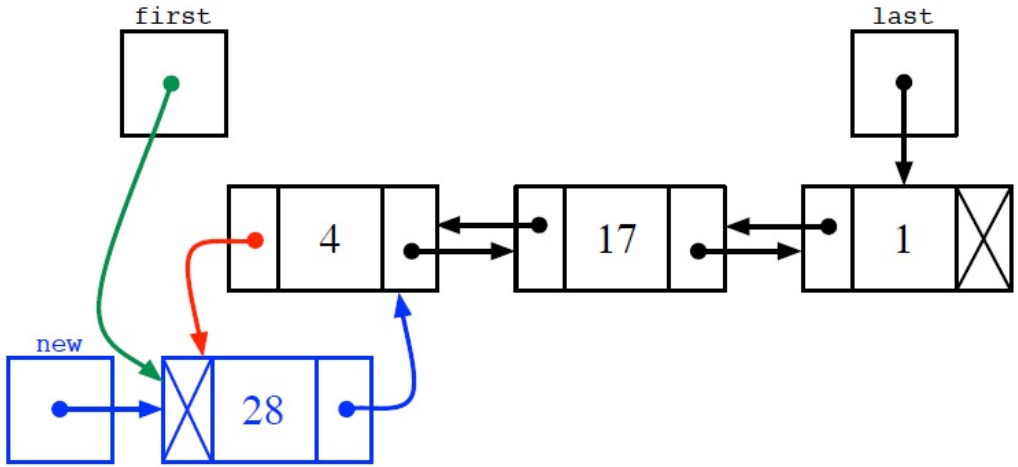
- La fonction **EstVide(Dliste L)** teste si la liste L est vide ou non,
- elle renvoie 1 si elle est vide, et 0 sinon.

Définition de la fonction

```
EstVide(Dliste L)
{
    if (L.first==NULL)
            return 1 ;
    else
            return 0 ;
}
```

Ajouter au debut : Dliste AjoutDebut(Dliste L,int x)

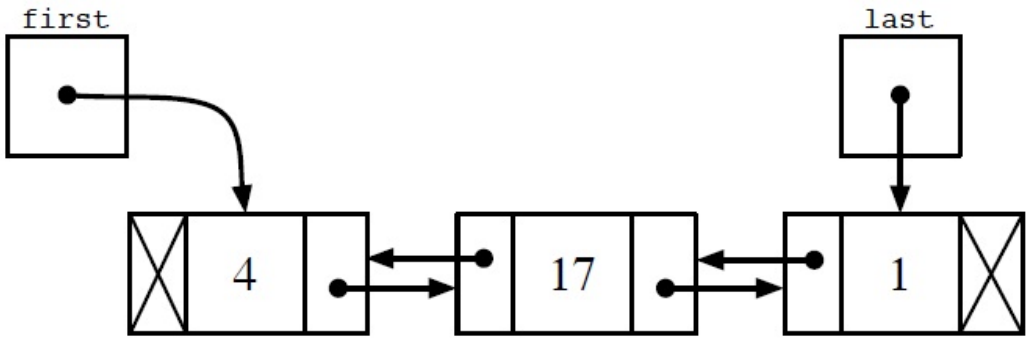


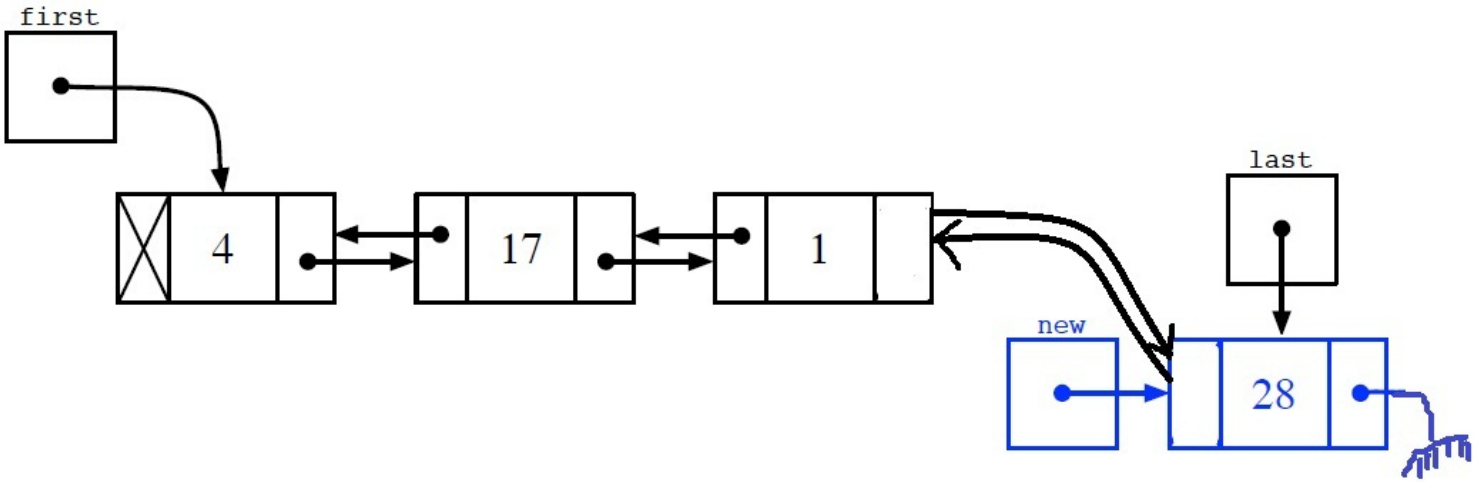


Ajouter au debut : Dliste AjoutDebut(Dliste L,int x)

```
Dliste AjoutDebut(Dliste L,int x)
    {     Dcellule *c;
          c=(Dcellule*)malloc(sizeof(Dcellule));//Réservation
          if( c!=NULL)
              {    c->info=x;
                   c->prec=NULL;
                   c->suiv=NULL;
                   //si L est vide
                   if (EstVide(L))
                       {    L.first=c;
                            L.last=c;
                       }
                   else
                       {
                            c->suiv=L.first;
                            L.first->prec=c;
                            L.first=c;
                       }
              }
          return L;
    }
```

Ajouter à la fin : Dliste AjoutFin(Dliste L,int x)





Ajouter à la fin : Dliste AjoutFin(Dliste L,int x)

```
Dliste AjoutFin(Dliste L,int x)
    {     Dcellule *c;
          c=(Dcellule*)malloc(sizeof(Dcellule));//Réservation
          if( c!=NULL)
               { c->info=x;
                 c->prec=NULL;
                 c->suiv=NULL;
                 //si L est vide
                 if (EstVide(L))
                     {  L.first=c;
                        L.last=c;
                     }
                 else
                     {  c->prec=L.last;
                        L.last->suiv=c;
                        L.last=c;
                     }
               }
          return L;

    }
```

Créer une cellule : Dcellule *newCellule() ;

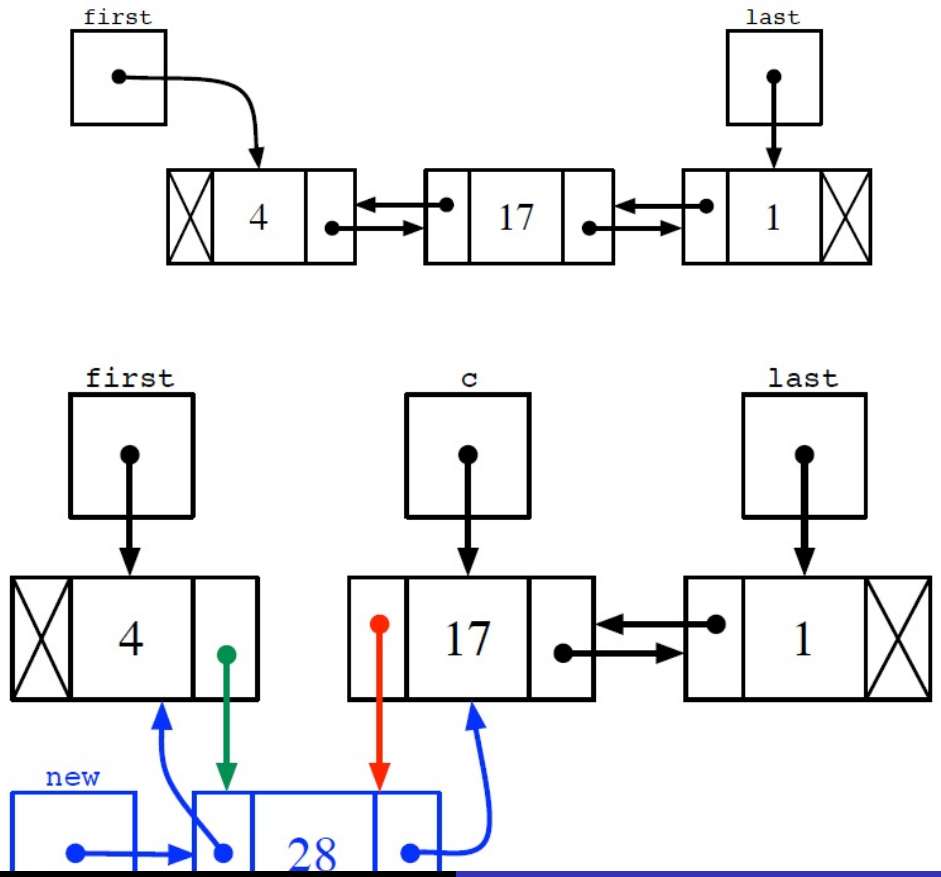
La fontion newCellule() permet de crée une nouvelle cellule dont l'élément est e, le suivant nxt et le précédent prev.

Défintion de la fonction

```
Dcellule *newCellule(Dcellule *prv, int e, Dcellule *nxt)
    {
    DCell *c ;
    c=malloc(sizeof(Dcellule)))
    if((c !=NULL)
        {
            c- >info=e ;
            c- >suiv=nxt ;
            c- >prec=prv ;
            return c ;
        }
    else
        {
            printf("Allocation ratée !") ;
            exit(EXIT_FAILURE) ;
        }
    }
```

Ajout dans une adresse donnée : Dliste insert(int e, Dcellule *c, Dliste L)

- La fonction **insert()** insère une dcellule contenant l'élément e à la place de la Dcellule pointée par c dans la liste L
- elle provoque une erreur si la liste est vide.



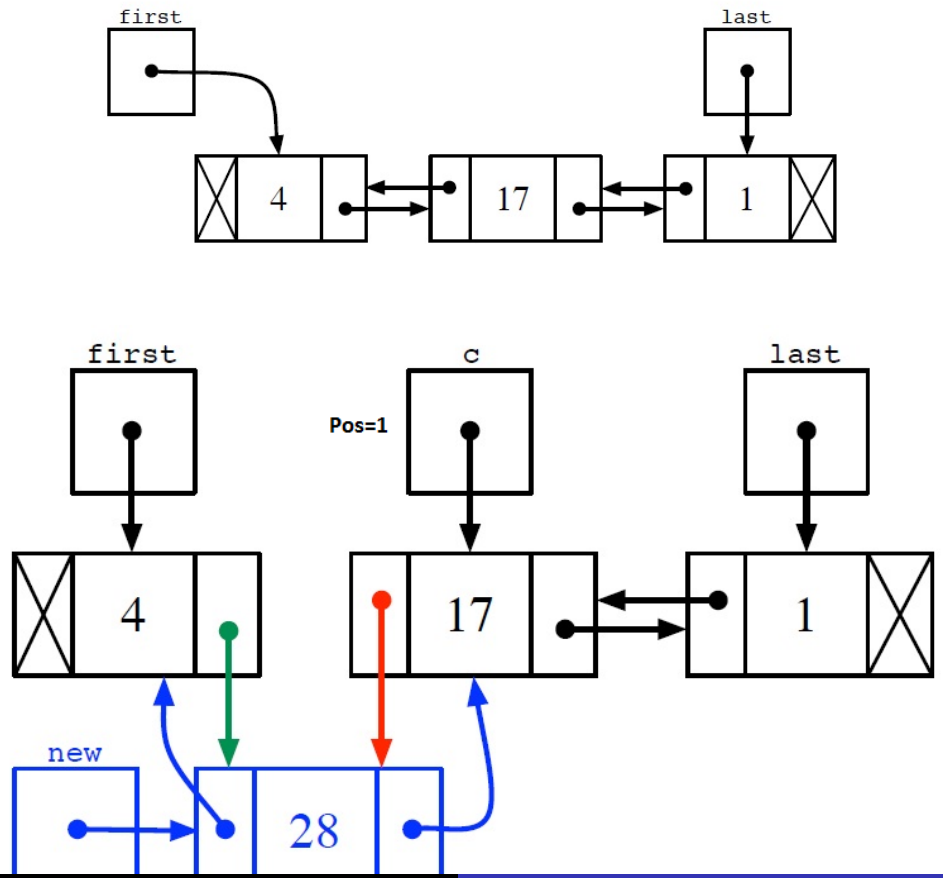
Ajout dans une adresse donnée : Dliste insert(int e, Dcellule *c, Dliste L)

Définition de la fonction

```
Dliste insert(int e, Dcellule *c, Dliste L)
    {
    Dcellule *new;
    if(c==l->first)
        L=AjoutDebut(L,e);
    else
        if( !EstVide(L) && c!=NULL)
        {
            new=newCellule(c->prec,e,c);
            c->prec=new;
            (new->prec)->suiv=new;
        }
        else
        {
            printf("insert impossible !");
            exit(EXIT_FAILURE);
        }
    return L;
    }
```

Ajout dans une position : Dliste insertPosition(int e, int pos, Dliste L)

- La fonction **insertPos()** insère une dcellule contenant l'élément e à la position pos de la liste L.
- elle provoque une erreur si la position n'est pas valide.

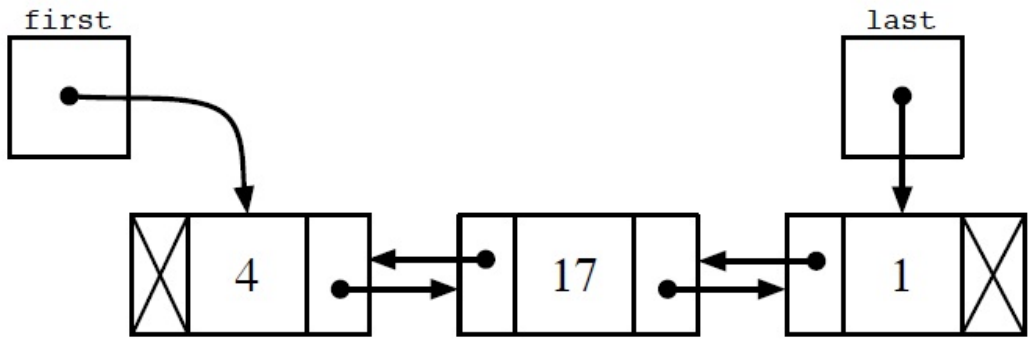


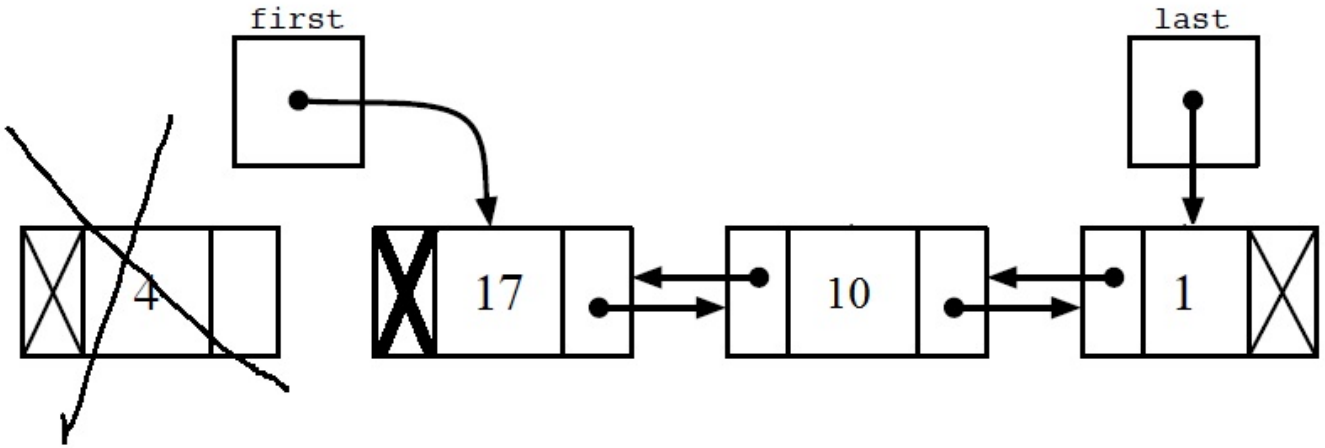
Ajout dans une position : Dliste insertPosition(Element e, int pos, Dliste L)

Définition de la fonction

```
Dliste insertPosition(int e, int pos, Dliste L)
    {
    Dcellule *tmp ;
    if(pos==0)
        L=AjoutDebut(L,e) ;
    else
        { tmp=l- >first ;
          while(tmp !=NULL && pos !=0)
              {     tmp=tmp- >suiv ;
                    pos=pos-1 ;
              }
          if(pos==0 && tmp==NULL)
                    L=AjoutFin(L,e) ;
          else
                    if(pos !=0) {printf("Position impossible !") ;
                          exit(EXIT_FAILURE) ;}
                    else { L=insert(e,tmp,L) ;
                          return L ;} }
    }
```

Supprimer au debut : Dliste SupprimerDebut(Dliste L)





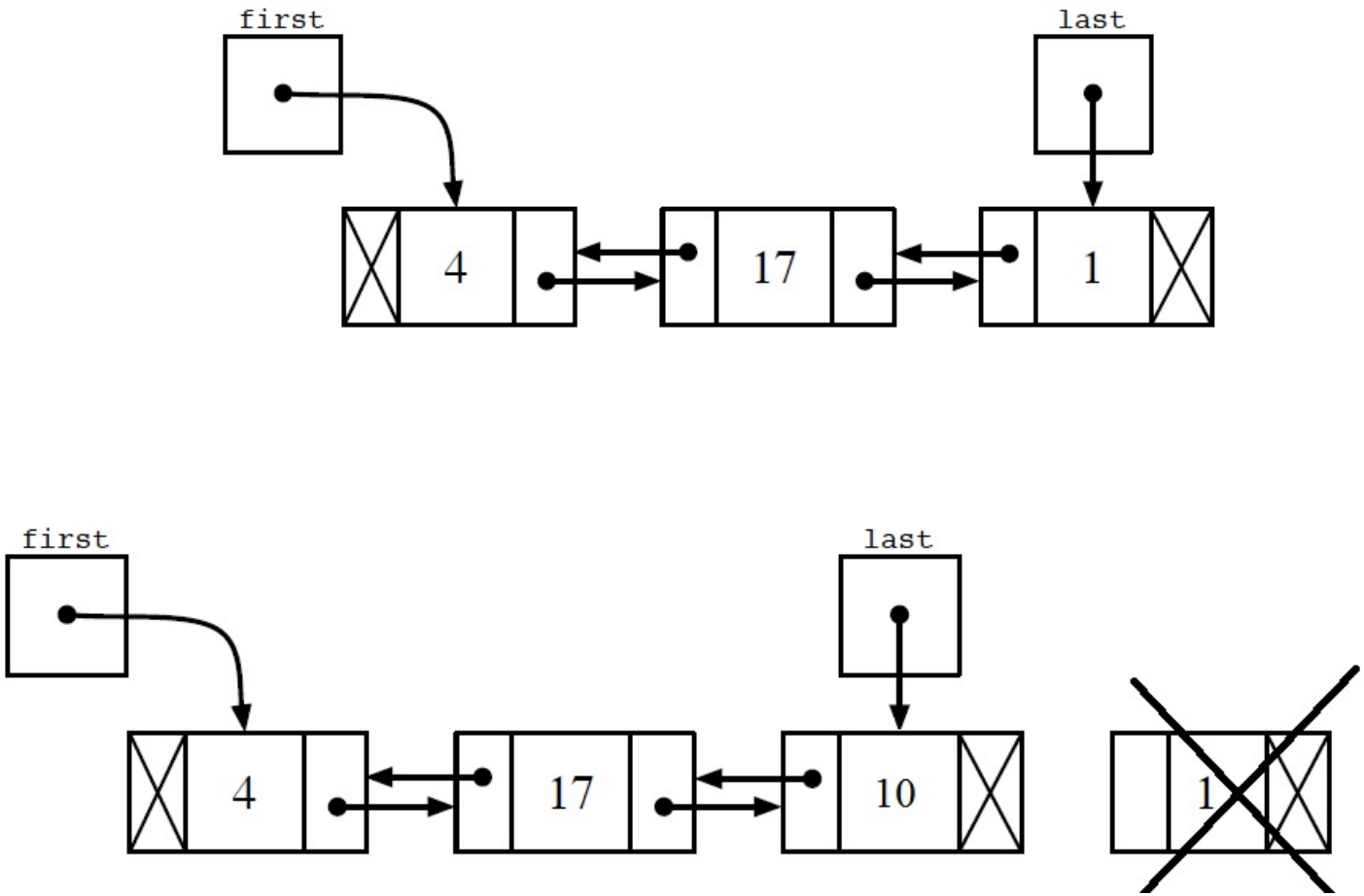
Supprimer au debut : Dliste SupprimerDebut(Dliste L)

- Définition de la fonction

```
Dliste SupprimerDebut(Dliste L)
    {
    Dcellule *c ;
    if(EstVide(L)==0)
        {
            c=L.first ;
            L.first=L.first- >suiv ;
            if (L.first==NULL)
                L.last=NULL ;
            else
                L.first- >prec=NULL ;
            free(c) ;
        }
    return L ;

    }
```

Supprimer à la fin :Dliste SupprimerFin(Dliste L)



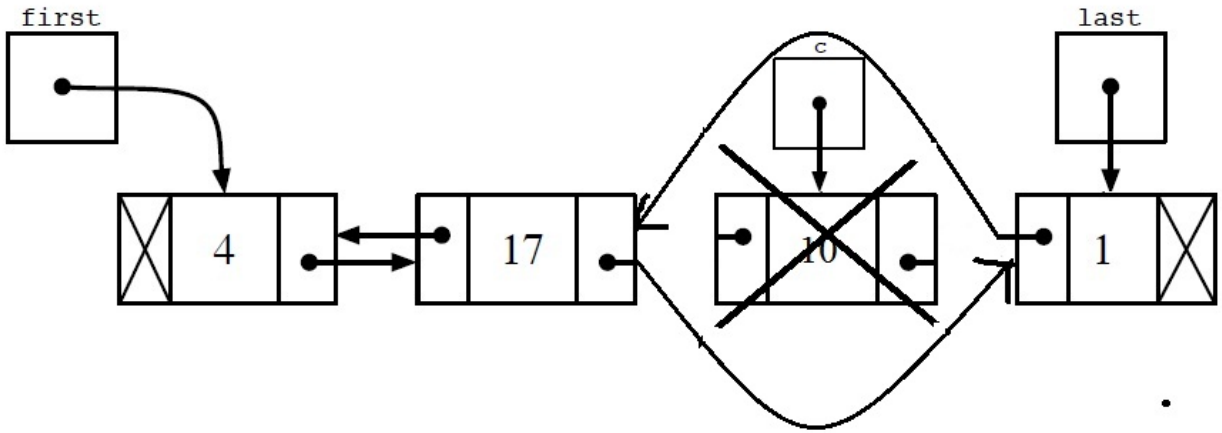
Supprimer à la fin : Dliste SupprimerFin(Dliste L)

- Définition de la fonction

```
Dliste SupprimerFin(Dliste L)
    {
    Dcellule *c ;
    if(EstVide(L)==0)
        {
            c=L.last ;
            L.last=L.last- >prec ;
            if (L.last==NULL)
                L.first=NULL ;
            else
                L.last- >suiv=NULL ;
            free(c) ;
        }
    return L ;
    }
```

Supprimer un élément : Dliste SupprimerVal(Dliste L,int x)

- La fonction **SupprimerVal()** supprime un élément x si il existe dans L
- elle retourne la liste initiale si x n'existe pas dans L
- Dans la figure, on va supprimer la valeur 10

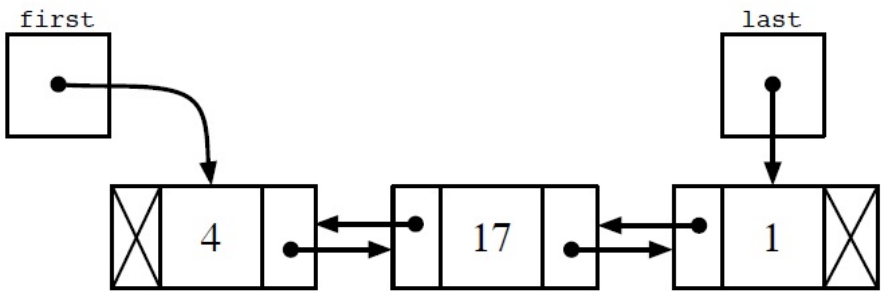


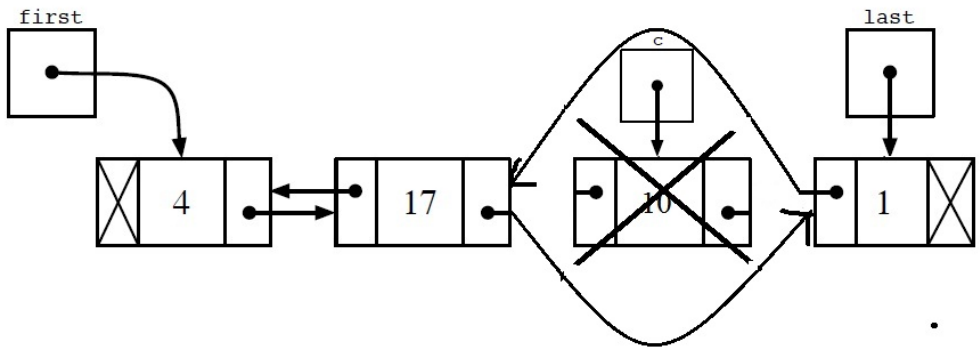
Supprimer un élément : Dliste SupprimerVal(Dliste L,int x)

```
Dliste SupprimerElement(Dliste L,int x)
    {   Dcellule *c,*p ;
        if(EstVide(L)==0)
            {    if (L.first− >info==x)
                    L=SupprimerDebut(L) ;
                 else
                    {    p=L.first ;
                         while(p !=NULL && p− >info !=x)
                            { p=p− >suiv ; }
                         if (p !=NULL)// x existe dans L
                            { if(p==L.last)
                                 {   L.last = p− >prec ;
                                      p− >prec− >suiv=NULL ; }
                              else
                                 {   p− >suiv− >prec = p− >prec ;
                                   p− >prec− >suiv = p− >suiv ; }
                              free(p) ;
                              }
                    }
            }
return L ;}
```

Supprimer un élément selon une adresse : int delete(Dcellule *c, Dliste L)

- La fonction **delete()** supprime et renvoie l'élément dans la Dcellule pointée par c dans la liste L
- elle provoque une erreur si la liste contient moins de 2 éléments





Supprimer un élément selon une adresse : int delete(Dcellule *c, Dliste L)

Définition de la fonction

```
int delete(Dcellule *c, Dliste L)
    {       int e;
    if(c==NULL || EstVide(L))
        {       printf("insert impossible!");
                exit(EXIT_FAILURE);
        }
    if(c==l->first)
        l->first=c->suiv;
    if(c==l->last)
        l->last=c->prec;
    e=c->info;
    if(c->suiv!=NULL)
        (c->suiv)->prec=c->prec;
    if(c->prec!=NULL)
        (c->prec)->suiv=c->suiv;
    free(c);
    return (e);
    }
```

Affichage d'une liste : void AfficherListe(Dliste *l)

Définition de la fonction

```
void AfficherListe(Dliste *l)
    {
    DCell *tmp;
    if(EstVide(L))
        printf("Liste vide");
    else
        {
            tmp=l->first;
            while(tmp!=NULL)
                {
                    printf("%d\n",tmp->info);
                    tmp=tmp->suiv;
                }
        }
    }
```

Rechercher un élément : Rechercher(Dliste L,int x)

Définition de la fonction

```
int Rechercher(Dliste L,int x)
    {
    Dcellule *tmp;
    if(EstVide(L))
        printf("Liste vide");
    else
        {
            tmp=l->first;
            while(tmp!=NULL && tmp->info!=x)
                {
                    tmp=tmp->suiv;
                }
            if(tmp!=NULL)
                return (1);
            else
                return (0);
        }
    }
```

Les primitives en utilisant un seul pointeur

```
 //Définir le type Dliste
typedef struct cellule{
    int info ;
    struct cellule *suiv ;
    struct cellule *prec ;
}Dliste ;
//////////Creer une liste vide
Dliste * CreerListeVide()
{
    Dliste *L ;
    L=NULL ;
    return L ;
}
//////////Tester si L est vide ou non
int EstVide(Dliste *L)
{    if (L==NULL)
        return 1 ;
    else
        return 0 ;
}
```

Les primitives en utilisant un seul pointeur

```
25  //////////AjoutDebut
26  Dliste * AjoutDebut(Dliste *L,int x)
27  {
28  Dliste *c;
29  //Réservation
30  c=(Dliste*)malloc(sizeof(Dliste));
31  if( c!=NULL)
32  {
33   c->info=x;
34   c->prec=NULL;
35   c->suiv=NULL  ;
36  //Branchement
37  //si L est vide
38  if (EstVide(L))
39    {
40     L=c;
41    }
42  else
43    {
44     c->suiv=L;
45     L->prec=c;
46     L=c;
47    }
48  }
49  return L;
50  }
```

Les primitives en utilisant un seul pointeur

```
////////////Adresse de la derniere cellule
Dliste * AddrDernCelluleV1(Dliste *L)
{
 Dliste *p=NULL;
 while(L!=NULL)
  {
   p=L;
   L=L->suiv;
  }
  return p;
}
////////////Adresse de la derniere cellule
Dliste * AddrDernCelluleV2(Dliste *L)
{
 if (L==NULL)
  return NULL;
while(L->suiv!=NULL)
  {
    L=L->suiv;
  }
 return L;
}
```

Les primitives en utilisant un seul pointeur

```
73    //////////Ajout Fin
74    Dliste * AjoutFin(Dliste *L,int x)
75    {
76    Dliste *c,*p;
77    //chercher l'adresse de la derniere cellule
78    p=AddrDernCelluleV1(L);
79    //Réservation
80    c=(Dliste*)malloc(sizeof(Dliste));
81    if( c!=NULL)
82    {
83     c->info=x;
84     c->prec=NULL;
85     c->suiv=NULL  ;
86     //branchement
87      if(p!=NULL) //L n'est pas vide
88       {
89        p->suiv=c;
90        c->prec=p;
91       }
92      else //L est vide
93      {
94        L=c;
95      }
96    }
97    return L;
98    }
```

Les primitives en utilisant un seul pointeur

```
111   //////////SupprimerDebut; Solution 1
112   Dliste * SupprimerDebutV1(Dliste *L)
113   {
114   Dliste *c;
115   if(!EstVide(L))
116   {
117    c=L;
118    L=L->suiv;
119    if(!EstVide(L))
120      {
121        L->prec=NULL;
122        free(c);
123      }
124    else
125      {
126       free(c);
127      }
128   }
129   return L;
130   }
```

Les primitives en utilisant un seul pointeur

```
131  //////////SupprimerDebut:  Solution 2
132  Dliste * SupprimerDebutV2(Dliste *L)
133 {
134  Dliste *c;
135  //si L est vide
136  if(L==NULL)
137   return L;
138  //Si L contient une seule cellule
139  c=L;
140  if (L->suiv=NULL)
141     {
142        L=NULL;
143      }
144  else //L contient plusieur cellules
145     {
146       L=L->suiv;
147       L->prec=NULL;
148     }
149   free(c);
150  return L;
151 }
```

Les primitives en utilisant un seul pointeur

```
152  //////////SupprimerFIn
153  Dliste * SupprimerFin(Dliste *L)
154 {
155  Dliste *c;
156  //chercher l'adresse de la derniere cellule
157  c=AddrDernCelluleV1(L);
158  if (c==NULL)
159    return L;
160  else
161    {
162     if (c->prec==NULL) //L contient 1 cellule
163       {
164         L=NULL;
165         free(c);
166       }
167     else //L contient plus d'une cellule
168      {
169       c->prec->suiv=NULL;
170       free(c);
171     }
172  }
173  return L;
174 }
```

Les primitives en utilisant un seul pointeur

```
175  Dliste * ChercherAdresse(Dliste *L,int x)
176 {
177  while(L!=NULL)
178  { if(L->info==x)
179       return L;
180    L=L->suiv;
181  }
182  return L;// L est NULL
183 }
184  ///////////////
185  Dliste * SupprimerElement(Dliste *L,int x)
186 { Dliste *c,*p;
187  p=ChercherAdresse(L,x);
188  //si x n'existe pas ou L est vide
189  if(p==NULL)
190    return L;
191  if(p->prec==NULL)
192    L=SupprimerDebutV1(L);
193  else
194    if (p->suiv==NULL)
195       L=SupprimerFin(L);
196    else
197       { c=p;
198         p->prec->suiv=p->suiv;
199         p->suiv->prec=p->prec;
200         free(c);
201       }
202  return L;
203 }
```

Les primitives en utilisant un seul pointeur

```
216  Dliste * InsertOrd(Dliste *L, int x)
217 {
218  Dliste *p,*c,*t;
219  p=NULL;
220  t=L;
221  while(t!=NULL && t->info<x)
222   {
223      p=t;
224      t=t->suiv;
225    }
226  //si L est NULL ou x<L->info
227  if (p==NULL)
228   L=AjoutDebut(L,x);
229  else
230    {
231      c=p->suiv;
232      c=AjoutDebut(c,x);
233      p->suiv=c;
234      c->prec=p;
235    }
236  return L;
237 }
```

Les primitives en utilisant un seul pointeur

```
229  int chercherElt(Dliste *L, int x)
230  {
231  while(L!=NULL)
232    {
233     if (L->info==x)
234       return 1;
235     L=L->suiv;
236    }
237  return 0;
238  }
239  ////////////
240  main()
241  {
242  Dliste *L;
243  L=CreerListeVide()    ;
244  L=AjoutDebut(L,10);
245  L=AjoutDebut(L,6);
246  L=AjoutDebut(L,4);
247  L=AjoutDebut(L,2);
248  //L=AjoutFin(L,10);
249  //L=AjoutFin(L,20);
250  AfficherListe(L);
251  printf("\n");
252  L=InsertOrd(L,1);
253  AfficherListe(L);
254  getch();
255  }
```

Liste circulaire

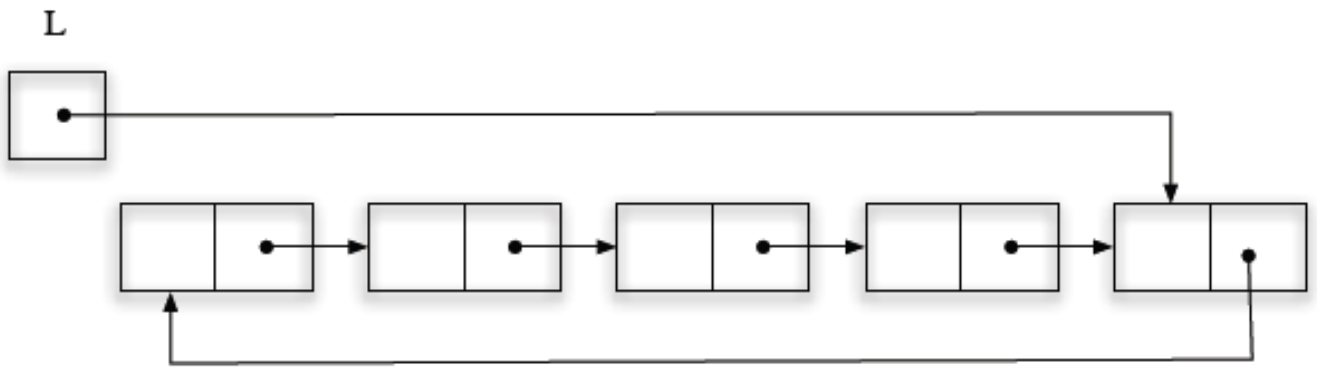
Définition

- Une liste est dite circulaire lorsque sa fin contient un pointeur sur la tête de la liste.
- Lorsqu'on parcourt la liste circulaire, on ne peut plus s 'arrêter lorsque NULL mais lorsqu'on revient sur la tête !
- Au lieu de repérer la tête de liste, il est plus judicieux de repérer la queue de la liste car le suivant de la queue est la tête !
- Intéressant pour les Files d 'attente

Types

- On distingue deux types de listes circulaires :
  1. Liste circulaire simplement chaînée
  2. Liste circulaire doublement chaînée

Liste circulaire simplement chaînée



Liste circulaire doublement chaînée