Exercices supplémentaires sur les listes chaînées

Exercice 231:

Ecrire une fonction récursive permettant de renvoyer la valeur du dernier élément dans une liste chaînée passée en paramètres.

Il est évident que pour pouvoir tester cette fonction, la liste doit avoir été préalablement remplie.

Notre algorithme/programme doit donc contenir en plus de la fonction demandée, les fonctions primitives permettant l'insertion en tête (ou en queue) d'une liste chaînée, et l'affichage de la liste remplie.

Solution :

Algorithme:

```
Algorithm exemple;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var L:List;n,v,i,dern:integer;
Function insertHead(L:List,v:integer):List;
Var p:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← L;
L ← p;
insertHead ← L;
End;
Procedure afficherListe(L:List);
Var p:List;
Begin
If L=Nil Then write("La liste est vide")
Else  Begin
      p ← L;
      While p≠Nil Do
            Begin
            Write(p^.val," ");
            p ← p^.next;
              End;
          End;
End;
Function dernierElement(L:List):integer;
Begin
If L=Nil Then dernierElement ← -1
```

```
Else If L^.next=Nil Then dernierElement ← L^.val
Else dernierElement ← dernierElement(L^.next);
End;
Begin
Write("Donner le nombre de valeurs à insérer dans la liste: ");
Read(n);
L ← Nil;
For i ← 1 to n do
      Begin
      Write("Donner la valeur ",i,": ");
      Read(v);
      L ← insertHead(L,v);
      End;
afficherListe(L);
dern ← dernierElement(L);
Write("Le dernier élemént est ",dern);
End.
```

Programme C:

```
#include <stdio.h>
#include <malloc.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;

liste insertHead(liste l,int v){
    liste p;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=l;
    l=p;
    return l;
}
void display(liste l){
    liste p;
    p=l;
    printf("La liste chainee:\n");
    while(p!=NULL){
        printf("%d, ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
int dernierElement(liste l){
    if(l==NULL)return -1;
    else if(l->suiv==NULL)return l->val;
    else return dernierElement(l->suiv);
}
main(){
liste l;int n,i,v,dern;
    printf("Donner le nombre de valeurs a inserer dans la liste: ");
```

```
    scanf("%d",&n);
    l=NULL;
    for(i=1;i<=n;i++){
        printf("\t Donner la valeur %d: ",i);
        scanf("%d",&v);
        l=insertHead(l,v);
    }
    display(l);
    dern=dernierElement(l);
    printf("Le dernier element est %d",dern);
    return 0;
}
```

Exercice 232:

- Ecrire une procédure récursive permettant de renvoyer le minimum et le maximum d'une liste chaînée passée en paramètre.
- Ecrire l'algorithme/programme principal qui permet de remplir une liste chaînée de réels suivant la stratégie insertion en queue, et qui cherche et affiche le minimum et la maximum de cette liste. La recherche du minimum et maximum doit se faire en appelant la procédure précédente.

Solution :

Algorithme:

```
Algorithm MinMax;
Type List =^node;
       node=Record
              Begin
              val:real;
              next:List;
              End;
Var L:List;n,i:integer;v,min,max:real;
Function insertTail(L:List,v:real):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                 q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure display(L:List);
Var p:List;
Begin
If L=Nil Then write("La liste est vide")
Else  Begin
      p ← L;
      While p≠Nil Do
```

```
            Begin
            Write(p^.val," ");
            p ← p^.next;
            End;
       End;
End;
Procedure calculerMinMax(L:List;Var min,max:real);
Begin
If L≠Nil then
      Begin
      If L^.next=Nil then
            Begin
            min ← L^.val;
            max ← L^.val;
            Fin
      Else  Begin
            calculerMinMax(L^.next,min,max);
            If L^.val < min Then min ← L^.val
            Else If L^.val > max Then max ← L^.val;
            End;
      End;
End;
Begin
Write("Donner le nombre de valeurs à insérer dans la liste: ");
Read(n);
L ← Nil;
For i ← 1 to n do
      Begin
      Write("Donner la valeur ",i,": ");
      Read(v);
      L ← insertTail(L,v);
      End;
afficherListe(L);
calculerMinMax(L,min,max);
Write("Le min est ",min);
Write("Le max est ",max);
End.
```

Programme C:

```
#include <stdio.h>
#include <malloc.h>
typedef struct node* liste;
typedef struct node{
               float val;
               liste suiv;
}node;
liste insertTail(liste l,float v){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v; p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
```

```
        while(q->suiv!=NULL)q=q->suiv;
        q->suiv=p;
    }
    return l;
}
void display(liste l){
    liste p;
    p=l;
    printf("La liste chainee:\n");
    while(p!=NULL){
        printf("%.2f, ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void calculerMinMax(liste l,float* min,float* max){
    if(l!=NULL){
        if(l->suiv==NULL){
            *min=l->val;
            *max=l->val;
        }
        else{
             calculerMinMax(l->suiv,min,max);
             if(l->val < *min)
                *min=l->val;
             else if(l->val > *max)
                *max=l->val;
        }
    }
}
main(){
liste l;int n,i;float v,min,max;
    printf("Donner le nombre de valeurs a inserer dans la liste: ");
    scanf("%d",&n);
    l=NULL;
    for(i=1;i<=n;i++){
        printf("\t Donner la valeur %d: ",i);
        scanf("%f",&v);
        l=insertTail(l,v);
    }
    display(l);
    calculerMinMax(l,&min,&max);
    printf("Le min est %.2f\n",min);
    printf("Le max est %.2f\n",max);
    return 0;
}
```

Exercice 233:

Ecrire un algorithme/programme permettant de saisir une série de notes d'une classe (la série se termine par une valeur en dehors de l'intervalle [0, 20]), et qui calcule la moyenne des notes de la classe, ainIf que le nombre de notes supérieures à la moyenne de la classe.

L'algorithme/programme doit se composer des sous-programmes suivants:

- Une fonction **insertTail** qui permet d'insérer un élément en queue d'une liste chaînée de réels.
- Une procédure **nbElements** qui compte et retourne le nombre d'éléments d'une liste chaînée de réels.
- Une fonction **somme** permettant de retourner la somme d'éléments d'une liste chaînée de réels.
- Une procédure **moyenne** qui calcule et renvoie la moyenne des notes (représentées par une liste chaînée de réels). Le calcul de la moyenne dans cette fonction doit se faire en utilisant les 2 sous-programmes **somme** et **nbElements** précédentes.
- Une fonction **nbSuppMoyenne** qui prend en paramètre une liste chaînée **L** et un réel **moy**, et qui renvoie le nombre d'éléments de **L** supérieurs à **moy**.
- L'algorithme/programme principal qui permet de saisir une suite de notes terminée par une valeur inférieure à 0 ou supérieure à 20, mettre ces notes dans une liste chaînée, et qui calcule et affiche la moyenne de notes, ainIf que le nombre de notes supérieures à la moyenne. Il est nécessaire d'exploiter les sous-programmes précédents.

Solution :

Algorithme:

```
Algorithm MinMax;
Type List =^node;
       node=Record
              Begin
              val:real;
              next:List;
              End;
Var tete:List;nb:integer;note,moy:real;
Function insertTail(L:List,v:real):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                  q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure nbElements(L:List;Var nb:integer);
Var p:List;
Begin
p ← L;
nb ← 0;
While p≠Nil Do
      Begin
      nb ← nb+1;
      p ← p^.next;
      End;
End;
```

```
Function somme(L:List):real;
Var p:List;s:real;
Begin
p ← L;
s ← 0;
While p≠Nil Do
      Begin
      s ← s+p^.val;
      p ← p^.next;
      End;
somme ← s;
End;
Procedure moyenne(L:List;Var moy:real);
Var s:real;nb:integer;
Begin
s ← somme(L);
nbElements(L,nb);
moy ← s/nb;
End;
Function nbSuppMoyenne(L:List;moy:real);
    liste p;int nb;
Begin
p ← l;
nb ← 0;
While p≠Nil Do
      Begin
      If p^.val>moy Then nb ← nb+1;
      p ← p^.next;
      End;
nbSuppMoyenne ← nb;
End;
Begin
tete ← Nil;
Write("Donner une note: ");
Read(note);
While note≥0 and note≤20 Do
      Begin
      tete ← insertTail(tete,note);
      Write("Donner une note: ");
      Read(note);
      End;
moyenne(tete,moy);
Write("La moyenne de la classe est ",moy);
nb ← nbSuppMoyenne(tete,moy);
Write("Le nombre de notes supérieures àa la moyenne est ",nb);
End;
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
```

```
                float val;
                liste suiv;
}node;
liste insertTail(liste l,float v){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
void nbElements(liste l,int *nb){
    liste p;
    p=l;
    *nb=0;
    while(p!=NULL){
        *nb=*nb+1;
        p=p->suiv;
    }
}
float somme(liste l){
    liste p;float s;
    p=l;
    s=0;
    while(p!=NULL){
        s=s+p->val;
        p=p->suiv;
    }
    return s;
}
void moyenne(liste l,float *moy){
    float s;int nb;
    s=somme(l);
    nbElements(l,&nb);
    *moy=s/nb;
}
int nbSuppMoyenne(liste l,float moy){
    liste p;int nb;
    p=l;
    nb=0;
    while(p!=NULL){
        if(p->val>moy)nb=nb+1;
        p=p->suiv;
    }
    return nb;
}
int main(){
    liste tete;float note,moy;int nb;
```

```
    tete=NULL;
    printf("Donner une note: ");
    scanf("%f",&note);
    while(note>=0 && note<=20){
        tete=insertTail(tete,note);
        printf("Donner une note: ");
        scanf("%f",&note);
    }
    moyenne(tete,&moy);
    printf("La moyenne de la classe est %.2f\n",moy);
    nb=nbSuppMoyenne(tete,moy);
    printf("Le nombre de notes superieures a la moyenne est %d",nb);
}
```

Exercice 234:

- Ecrire la fonction récursive **`nbElements()`** qui retourne le nombre d'éléments d'une liste chaînée passée en paramètre, et renvoie aussi dans des paramètres passées par variable le nombre d'éléments positifs et le nombre d'éléments négatifs..
- Tester cette fonction dans un algorithme/programme qui permet de remplir une liste chaînée par des valeurs saisies au clavier se terminant par 0, et qui compte et affiche le nombre d'éléments de cette liste, le nombre de ses éléments positifs et le nombre de ses éléments négatifs en appelant la fonction précédente.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var L:List;x,nb,nbPos,nbNeg:integer;
Function insertTail(L:List,v:integer):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                 q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure display(L:List);
Var p:List;
Begin
If L=Nil Then write("La liste est vide")
```

```
Else Begin
      p ← L;
      While p≠Nil Do
            Begin
            Write(p^.val," ");
            p ← p^.next;
              End;
           End;
End;
Function nbElements(L:List;Var nbPos,nbNeg:integer):integer;
If L=Nil then
      Begin
      nbElements ← 0;
      nbPos ← 0;
      nbNeg ← 0;
      Fin
Else  Begin
       If L^.val > 0 Then nbPos ← nbPos+1
       Else nbNeg ← nbNeg + 1;
       nbElements ← 1+nbElements(L^.next,nbPos,nbNeg);
       End;
End;
Begin
L ← NULL;
Repeat
   Write("Donner un nombre entier: ");
   Read(x);
   If x≠0 Then L ← insertTail(L,x);
Until x=0;
display(L);
nbPos ← 0;
nbNeg ← 0;
nb ← nbElements(L,nbPos,nbNeg);
Write("Le nombre d'elements est: ",nb);
Write("Le nombre de valeurs positives est: ",nbPos);
Write("Le nombre de valeurs négatives est: ",nbNeg);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
typedef struct node* liste;
typedef struct node{
    int val;
    liste suiv;
}node;
liste insertTail(liste l,int v){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
```

```
        while(q->suiv!=NULL)q=q->suiv;
        q->suiv=p;
    }
    return l;
}
void display(liste l){
    liste p;
    p=l;
    printf("Les elemnts de la liste:\n");
    while(p!=NULL){
        printf("%d | ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
int nbElements(liste l,int* nbPos,int* nbNeg){
    if(l==NULL){
         return 0;
         *nbPos=0;
         *nbNeg=0;
    }
    else{
         if(l->val > 0)*nbPos=*nbPos+1;
         else *nbNeg=*nbNeg + 1;
         return 1+nbElements(l->suiv,nbPos,nbNeg);
    }
}
int main(){
    liste L;int x,nb,nbPos,nbNeg;
    L=NULL;
    do{
        printf("Donner un nombre entier: ");
        scanf("%d",&x);
        if(x!=0)L=insertTail(L,x);
    }while(x!=0);
    display(L);
    nbPos=0;
    nbNeg=0;
    nb=nbElements(L,&nbPos,&nbNeg);
    printf("Le nombre d'elements est: %d\n",nb);
    printf("Le nombre de valeurs positives est: %d\n",nbPos);
    printf("Le nombre de valeurs negatives est: %d\n",nbNeg);
}
```

Exercice 235:

- Ecrire la procédure **`separer()`** qui, à partir d'une liste chaînée d'entiers, crée deux autres listes telles que la première contient les nombres pairs et la deuxième ceux impairs. A la fin, la liste originale doit être vide.
- Ecrire l'algorithme/programme principal qui permet de saisir au clavier les éléments d'une liste chainée d'entiers, et qui la sépare en deux listes: la première contient les éléments pairs et la seconde les éléments impaires.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
      node=Record
             Begin
```

```
              val:integer;
              next:List;
              End;
Var L,LPairs,LImpairs:List;n,x:integer;
Function insertTail(L:List,v:integer):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure afficherListe(L:List;message:Chaine);
Var p:List;
Begin
p ← L;
Write(message);
If p=Nil Then write("vide")
Else   Begin
       While p≠Nil Do
            Begin
            Write(p^.val," ");
            p ← p^.next;
            End;
       End;
End;
Procedure separer(Var L,L2,L3:List);
Var p:List;
Begin
L2 ← Nil;
L3 ← Nil;
While L≠Nil Do
      Begin
      p ← L;
      If p^.val mod 2=0 Then L2 ← insertTail(L2,p^.val)
      Else L3 ← insertTail(L3,p^.val);
      L ← L^.next;
      Free(p);
      End;
End;
Begin
L ← Nil;
Write("Quelle est le nombre de valeurs à saisir? ");
Read(n);
For i ← 1 to n do
```

```
      Begin
      Write("Donner la valeur ",i,": ");
      Read(x);
      L ← insertTail(L,x);
      End;
afficherListe(L,"La liste initiale:");
separer(L,LPairs,LImpairs);
afficherListe(LPairs,"la liste des éléments pairs:");
afficherListe(LImpairs,"la liste des éléments impairs:");
afficherListe(L,"la liste original:");
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
    int val;
    liste suiv;
}node;
liste insertTail(liste l,int c){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=c; p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
void afficherListe(liste l,char message[]){
    liste p;
    p=l;
    printf("%s\n",message);
    if(p==NULL)printf("Vide\n");
    else{
        while(p!=NULL){
            printf("%d, ",p->val);
            p=p->suiv;
        }
        printf("\n");
    }
}
void separer(liste* l,liste* l2,liste* l3){
    liste p;
    (*l2)=NULL;
    (*l3)=NULL;
    while((*l)!=NULL){
        p=*l;
        if(p->val%2==0)*l2=insertTail(*l2,p->val);
        else (*l3)=insertTail(*l3,p->val);
```

```
        *l=(*l)->suiv;
        free(p);
    }
}
int main(){
    liste L,LPairs,LImpairs;int n,x;
    L=NULL;
    printf("Quelle est le nombre de valeurs a saisir? ");
    scanf("%d",&n);
    for(int i=1; i<=n;i++){
        printf("Donner la valeur %d: ",i);
        scanf("%d",&x);
        L=insertTail(L,x);
    }
    afficherListe(L,"La liste initiale:");
    separer(&L,&LPairs,&LImpairs);
    afficherListe(LPairs,"la liste des elements pairs:");
    afficherListe(LImpairs,"la liste des elements impairs:");
    afficherListe(L,"la liste original:");
    return 0;
}
```

Exercice 236:

On convient de représenter un nombre binaire $b_1b_2...b_n$, où chaque b_i est un 0 ou un 1, par une liste chaînée où chaque élément contient un b_i.

On vous demande dans cet exercice d'établir un programme permettant de lire un nombre entier positif et de le convertir en son équivalent binaire tout en stockant les bits du nombre binaire dans une liste chaînée. Le programme doit finalement afficher le nombre binaire résultant représenté par la liste chaînée.

Les étapes à suivre sont:

- Ecrire la fonction **`insertHead()`** qui permet d'insérer un élément en tête d'une liste chaînée d'entiers.
- Ecrire la fonction **`conversionDecimalBinaire()`** qui prend en paramètre un entier ***`n`*** et qui retourne son équivalent binaire sous forme d'une liste chaînée. L'insertion des bits dans cette liste se fera en appelant la fonction **`insertHead()`** précédente.
- Ecrire la fonction **`display()`** qui permet d'afficher un nombre binaire représenté par une liste chaînée d'entiers.
- En utilisant les fonctions précédentes, écrire l'algorithme/programme principal qui permet de saisir un nombre entier positif, de calculer son équivalent binaire, et d'afficher le résultat.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var tete:List;n:integer;
Function insertHead(L:List,v:integer):List;
Var p:List;
```

```
Begin
Allocate(p);
p^.val ← v;
p^.next ← L;
L ← p;
insertHead ← L;
End;
Procedure display(L:List);
Var p:List;
Begin
p ← L;
Write("(");
While p≠Nil Do
      Begin
      Write(p^.val," ");
      p ← p^.next;
      End;
Write(")2");
End;
Function conversionDecimalBinaire(n:integer):List;
Var L:List;r:integer;
Begin
L ← Nil;
While n≠0 Do
      Begin
      r ← n mod 2;
      n ← n div 2;
      L ← insertHead(L,r);
      End;
conversionDecimalBinaire ← L;
End;
Begin
tete ← Nil;
printf("Donner un nombre entier positif: ");
Read(n);
tete ← conversionDecimalBinaire(n);
Write("(",n,")10 = ");
display(tete);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertHead(liste l,int v){
    liste p;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=l;
```

```
    l=p;
    return l;
}
void display(liste l){
    liste p;
    p=l;
    printf("(");
    while(p!=NULL){
        printf("%d",p->val);
        p=p->suiv;
    }
    printf(")2\n");
}
liste conversionDecimalBinaire(int n){
    liste l;int r;
    l=NULL;
    while(n!=0){
        r=n%2;
        n=n/2;
        l=insertHead(l,r);
    }
    return l;
}
int main(){
    liste tete;int n;
    tete=NULL;
    printf("Donner un nombre entier positif: ");
    scanf("%d",&n);
    tete=conversionDecimalBinaire(n);
    printf("(%d)10 = ",n);
    display(tete);
}
```

Exercice 237:

Soit ***L*** une liste chaînée d'entiers représentant un nombre binaire, où chaque élément de ***L*** contient un bit du nombre binaire. On veut établir un algorithme/programme qui permet de saisir un nombre binaire, le mettre dans une telle liste chaînée, et qui calcule et affiche son équivalent décimal. Il vous ai alors demandé de:

- Ecrire la fonction **`insertHead()`** qui permet d'insérer un élément en tète d'une liste chaînée d'entiers.
- Ecrire la fonction **`extraireChiffres()`** qui prend en paramètre un entier positif ***n*** et qui retourne une liste chaînée composée par les chiffres de ***n***. L'insertion dans la liste chaînée doit se faire en appelant la fonction **`insertHead()`**.
- Ecrire la fonction **`nbElements()`** qui retourne le nombre d'éléments d'une liste chaînée passée en paramètre.
- Ecrire la procédure **`conversionBinaireDecimal()`** qui prend en entrée une liste chaînée représentant un nombre binaire, et qui renvoie son équivalent décimal. Utilisez la fonction **`nbElements()`** pour calculer le poids le plus fort du nombre binaire.

- Ecrire l'algorithme/programme principal qui permet de saisir un nombre entier positif composé uniquement des 0 et des 1 considéré comme nombre binaire, de le convertir en décimal, et d'afficher le résultat. Il faut exploiter les sous-programmes précédents.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var tete:List;n,b:integer;
Function insertHead(L:List,v:integer):List;
Var p:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← L;
L ← p;
insertHead ← L;
End;
Function extraireChiffres(n:integer):List;
Var L:List;r:integer;
Begin
L ← NULL;
While n≠Nil Do
      Begin
      r ← n mod 10;
      n ← n div 10;
      L ← insertHead(L,r);
      End;
extraireChiffres ← L;
End;
Function nbElements(L:List):integer;
Var p:List;nb:integer;
Begin
p ← l;nb ← 0;
While p≠Nil Do
      Begin
      nb ← nb+1;
      p ← p^.next;
      End;
nbElements ← nb;
End;
Procedure conversionBinaireDecimal(L:List,Var n:integer);
Var p:List;nb,poids:integer;
Begin
nb ← nbElements(L);
poids ← nb-1;
n ← 0;
```

```
While p≠Nil Do
      Begin
      n ← n + p^.val * 2^poids;
      p ← p^.next;
      poids ← poids-1;
      End;
conversionBinaireDecimal ← n;
End;
Begin
tete ← NULL;
Write("Donner un nombre binaire: ");
Read(b);
tete ← extraireChiffres(b);
conversionBinaireDecimal(tete,n);
Write("(",b,")2=(",n,")10");
End.
```

Pogramme C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertHead(liste l,int v){
    liste p;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=l;
    l=p;
    return l;
}
liste extraireChiffres(int n){
    liste l;int r;
    l=NULL;
    while(n!=NULL){
        r=n%10;
        n=n/10;
        l=insertHead(l,r);
    }
    return l;
}
int nbElements(liste l){
    liste p;int nb;
    p=l;nb=0;
    while(p!=NULL){
        nb=nb+1;
        p=p->suiv;
    }
    return nb;
}
```

```
void conversionBinaireDecimal(liste l,int* n){
    liste p;int nb,poids;
    nb=nbElements(l);
    poids=nb-1;
    *n=0;
    while(p!=NULL){
        *n=(*n)+p->val * pow(2,poids);
        p=p->suiv;
        poids=poids-1;
    }
    return n;
}
int main(){
    liste tete;int n,b;
    tete=NULL;
    printf("Donner un nombre binaire: ");
    scanf("%d",&b);
    tete=extraireChiffres(b);
    conversionBinaireDecimal(tete,&n);
    printf("(%d)2=(%d)10",b,n);
}
```

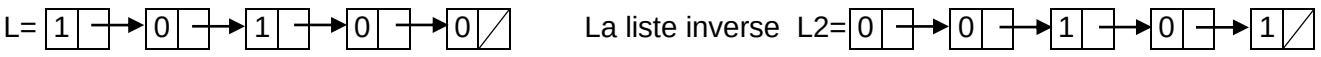
Exercice 238:

Nous avons vu dans les 2 exercices précédents qu'il est commun de représenter un nombre binaire par une liste chaînée d'entiers de façon à ce que chaque élément de cette liste chaînée contient un bit et que la tête pointe sur le bit du poids fort.

Dans cet exercice nous voudrons établir un algorithme/programme qui permet de saisir 2 nombres entiers, et de calculer leur somme, mais en binaire, avant de reconvertir le résultat en décimal. Le calcul de la somme en binaire suppose que chaque nombre binaire, y compris la somme, soit représenté par une liste chaînée.

Afin d'accomplir cet objectif, plusieurs étapes doivent être suivre et plusieurs sous-programmes doivent êtres mises en places.

- Ecrire la fonction **`insertHead()`** qui permet d'insérer un élément en tète d'une liste chaînée d'entiers.
- Ecrire la fonction **`ObtenirListeinverse()`** qui prend en entrée une liste chaînée d'entiers **`L`**, et qui retourne une autre liste composée des éléments de **`L`** mais dans l'ordre inverse. Par exemple si:



  Notons que l'obtention de la liste inverse peut se faire tout simplement en employant la fonction **`insertHead()`**.

- Ecrire la fonction **`conversionDecimalBinaire()`** qui prend en paramètre un entier ***`n`*** et qui retourne son équivalent binaire sous forme d'une liste chaînée. L'insertion des bits dans cette liste se fera en appelant la fonction **`insertHead()`** précédente.
- Ecrire la fonction **`sommeBinaire()`** qui prend en paramètres 2 nombres binaires représenté chacun par une liste chaînée, et qui retourne leur somme sous forme d'une liste chaînée aussi. Comme indication, vous devez utiliser les 2 fonctions **`ObtenirListeinverse()`** et **`insertHead`**().
- Ecrire la procédure **`conversionBinaireDecimal()`** qui prend en entrée une liste chaînée représentant un nombre binaire, et qui renvoie son équivalent décimal. Utilisez la fonction **`ObtenirListeinverse()`** pour faciliter le calcul des poids des bits.

- En utilisant les sous-programmes précédents, écrire l'algorithme/programme principal qui permet de saisir 2 nombres entiers positifs, de les convertir en binaire, de calculer leur somme en binaire, et puis de convertir cette somme en décimal et l'afficher.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var tete1,tete2,tete3:List;n1,n2,s:integer;
Function insertHead(L:List,v:integer):List;
Var p:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← L;
L ← p;
insertHead ← L;
End;
Function extraireChiffres(n:integer):List;
Var L:List;r:integer;
Begin
L ← NULL;
While n≠Nil Do
      Begin
      r ← n mod 10;
      n ← n div 10;
      L ← insertHead(L,r);
      End;
extraireChiffres ← L;
End;
Function conversionDecimalBinaire(n:integer):List;
Var L:List;r:integer;
Begin
L ← Nil;
While n≠0 Do
      Begin
      r ← n mod 2;
      n ← n div 2;
      L ← insertHead(L,r);
      End;
conversionDecimalBinaire ← L;
End;
Function inverser(L:List):List;
L2,p:List;
Begin
p ← L;L2 ← Nil;
While p≠Nil Do
```

```
      Begin
      L2 ← insertHead(L2,p^.val);
      p ← p^.next;
      End;
inverser ← L2;
End;
Function sommeBinaire(L1,L2:List):List;
Var L,p1,p2:List;s,r:integer;
Begin
If L1=Nil Then L ← L2
Else If L2=Nil Then L ← L1
Else  Begin
       L ← Nil;
       p1 ← inverser(L1);
       p2 ← inverser(L2);
       r ← 0;
       While p1≠Nil or p2≠Nil Do
           Begin
           s ← r;
           If p1≠Nil Then s ← s+p1^.val;
           If p2≠Nil Then s ← s+p2^.val;
           If s<2 then
                Begin
                L ← insertHead(L,s);
                r ← 0;
                Fin
           Else  Begin
                 r ← 1;
                 s ← s-2;
                 L ← insertHead(L,s);
                 End;
           If p1≠Nil Then p1 ← p1^.next;
           If p2≠Nil Then p2 ← p2^.next;
           End;
       If r≠0 Then L ← insertHead(L,r);
       End;
sommeBinaire ← L;
End;
Function conversionBinaireDecimal(L:List):integer;
Var p:List;n,i:integer;
Begin
p ← inverser(L);
n ← 0;i ← 0;
While p≠Nil Do
      Begin
      n ← n + p^.val * 2^i;
      i ← i+1;
      p ← p^.next;
      End;
conversionBinaireDecimal ← n;
End;
```

```
Begin
tete1 ← Nil;
Write("Donner un nombre entier positif: ");
Read(n1);
Write("Donner un autre nombre entier positif: ");
Read(n2);
tete1 ← conversionDecimalBinaire(n1);
tete2 ← conversionDecimalBinaire(n2);
tete3 ← sommeBinaire(tete1,tete2);
s ← conversionBinaireDecimal(tete3);
Write("La somme en décimal est ",s);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertHead(liste l,int v){
    liste p;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=l;
    l=p;
    return l;
}
liste conversionDecimalBinaire(int n){
    liste l;int r;
    l=NULL;
    while(n!=0){
        r=n%2;
        n=n/2;
        l=insertHead(l,r);
    }
    return l;
}
liste inverser(liste l){
    liste l2,p;
    p=l;l2=NULL;
    while(p!=NULL){
        l2=insertHead(l2,p->val);
        p=p->suiv;
    }
    return l2;
}
liste sommeBinaire(liste l1,liste l2){
    liste l,p1,p2;int s,r;
    if(l1==NULL)l=l2;
    else if(l2==NULL)l=l1;
```

```
    else{
        l=NULL;
        p1=inverser(l1);
        p2=inverser(l2);
        r=0;
        while(p1!=NULL || p2!=NULL){
            s=r;
            if(p1!=NULL)s=s+p1->val;
            if(p2!=NULL)s=s+p2->val;
            if(s<2){
                    l=insertHead(l,s);
                    r=0;
            }
            else{
                r=1;
                s=s-2;
                l=insertHead(l,s);
            }
            if(p1!=NULL)p1=p1->suiv;
            if(p2!=NULL)p2=p2->suiv;
        }
        if(r!=0)l=insertHead(l,r);
    }
    return l;
}
int conversionBinaireDecimal(liste l){
    liste p;int n,i;
    p=inverser(l);
    n=0;i=0;
    while(p!=NULL){
        n=n+p->val * pow(2,i);
        i=i+1;
        p=p->suiv;
    }
    return n;
}
int main(){
    liste tete1,tete2,tete3;int n1,n2,s;
    tete1=NULL;
    printf("Donner un nombre entier positif: ");
    scanf("%d",&n1);
    printf("Donner un autre nombre entier positif: ");
    scanf("%d",&n2);
    tete1=conversionDecimalBinaire(n1);
    tete2=conversionDecimalBinaire(n2);
    tete3=sommeBinaire(tete1,tete2);
    s=conversionBinaireDecimal(tete3);
    printf("la somme en decimal est %d",s);
}
```

Exercice 239:

On souhaite écrire un algorithme/programme modulaire permettant de saisir un ensemble de caractères, les mettre dans une liste chaînée, et puis il supprime de cette liste toutes les occurrences d'un caractère saiIf par l'utilisateur. Finalement le programme affiche la liste résultante.

En effet, l'algorithme/programme à établir doit être modulaire, c'est à dire qu'il ne doit pas centraliser tout le code dans un seul bloc, mais le répartir en plusieurs modules (sous-programmes). Vous pouvez entre autres établir les modules suivants:

- Une fonction **`insertTail()`** qui permet d'insérer un élément en queue d'une liste chaînée de caractères.
- Une procédure **`supprimerToutesOccurrences()`** qui prend en paramètre une liste chaînée **`L`** et un caractère **`c`** et qui supprime de **`L`** toutes les occurrence du caractère **`c`**.
- Une procédure **`afficherListe()`** permettant d'afficher les éléments d'une liste chaînée de caractères passée en paramètre.
- L'algorithme/programme principal qui combine les sous-programmes précédents pour répondre au besoin demandé.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
       node=Record
              Begin
              val:character;
              next:List;
              End;
Var L:List;n,i:integer;v:character;
Function insertTail(L:List,c:character):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← c;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                 q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure afficherListe(L:List;message:Chaine);
Var p:List;
Begin
p ← L;
Write(message);
If p=Nil Then write("vide")
Else   Begin
       While p≠Nil Do
            Begin
            Write(p^.val," ");
```

```
            p ← p^.next;
            End;
       End;
End;
Procedure supprimerToutesOccurrences(Var L:List;c:character);
Var p,q:List;
Begin
While L≠Nil and L^.val=c Do
       Begin
       p ← l;
       L ← L^.next;
       Free(p);
       Fin
If L≠Nil then
       Begin
       q ← L;p ← L^.next;
       While p≠Nil Do
            Begin
            If p^.val=c then
                 Begin
                 q^.next ← p^.next;
                 Free(p);
                 p ← q^.next;
                 End;
             Else  Begin
                   p ← p^.next;
                   q ← q^.next;
                   End;
            End;
       End;
End;
Begin
L ← NULL;
Write("Quel est le nombre de caractères à saisir? ");
Read(n);
For i ← 1 to n do
       Begin
       Write("Donner le caractère ",i,": ");
       Read(v);
       L ← insertTail(L,v);
       End;
afficherListe(L,"La liste initiale:");
printf("Quel est le caractère que voudrez-vous supprimer? ");
Read(v);
supprimerToutesOccurrences(L,v);
afficherListe(L,"la liste après suppression:");
End.
```

Programme C:

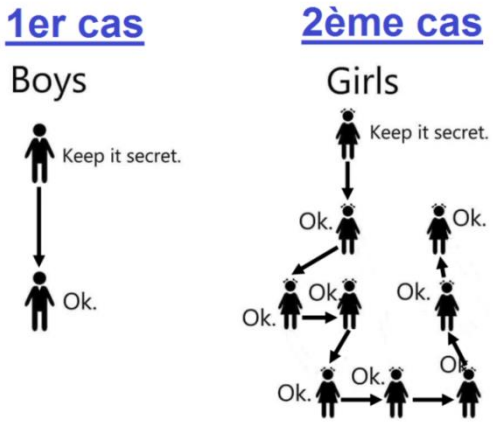
```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
```

```
    char val;
    liste suiv;
}node;
liste insertTail(liste l,char c){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=c; p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
void afficherListe(liste l,char message[]){
    liste p;
    p=l;
    printf("%s\n",message);
    while(p!=NULL){
        printf("%c, ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void supprimerToutesOccurrences(liste* l,char c){
    liste p,q;
    while(*l!=NULL && (*l)->val==c){
        p=l;
        (*l)=(*l)->suiv;
        free(p);
    }
    if(*l!=NULL){
        q=(*l);p=(*l)->suiv;
        while(p!=NULL){
           if(p->val==c){
             q->suiv=p->suiv;
             free(p);
             p=q->suiv;
           }
           else{
                p=p->suiv;
                q=q->suiv;
           }
        }
     }
}
int main(){
    liste L;int n;char v,p;
    L=NULL;
    printf("Quel est le nombre de caracteres a saisir? ");
    scanf("%d",&n);
    scanf("%c",&p);
```

```
    for(int i=1; i<=n;i++){
        printf("Donner le caractere %d: ",i);
        scanf("%c",&v);
        scanf("%c",&p);
        L=insertTail(L,v);
    }
    afficherListe(L,"La liste initiale:");
    printf("Quelle est le caractere que voudrez-vous supprimer? ");
    scanf("%c",&v);
    supprimerToutesOccurrences(L,v);
    afficherListe(L,"la liste apres suppression:");
    return 0;
}
```

Exercice 240:

Considérons la problématique illustrée par la figure suivante:



En fait, s'il suffit de représenter le premier cas par deux variables ou un tableau de deux cases, le deuxième cas est plus compliqué.

1) Proposez une structure de données pour modéliser le deuxième cas
2) On veut sauvegarder le message transféré par chaque personne à la personne suivante afin de voir à la fin, If Le message a été transféré honnêtement ou pas, et dans le cas échéant, identifier les personnes qui n'ont pas transmis avec précision le message reçu.

   Proposez une solution:

   - Du point de vue structure de données, est ce que la structure proposée précédemment par vous est suffisante ou on doit la modifier.
   - Du point de vue traitement, quels sont les sous-programmes que vous proposez pour accomplir l'objectif visé.

Solution :

Algorithme:

```
Algorithm calcul;
Type List =^node;
      node=Record
             Begin
             personne:chaine[20];
             message:chaine[100];
             next:List;
             End;
```

```
Var L,L2:List;reponse:integer;nom1,nom2:Chaine[20];message:Chaine[100];
Procedure display(L:List);
Var p:List;
Begin
p ← L;
While p≠Nil Do
      Begin
      Write(p^.personne);
      p ← p^.next;
      End;
End;
Function inserer_queue(pers:Chaine[20];mess:Chaine[100];L:List):List;
Var p,d:List;i:integer;
Begin
Allocate(d);
d^.personne ← pers;
d^.message ← mess;
d^.next ← Nil;
If L=Nil Then L ← d
Else   Begin
       p ← l;
       While p^.next≠Nil Do
           p ← p^.next;
       p^.next ← d;
       End;
inserer_queue ← L;
End;
Function identierPersonnes(L:List):List;
Var L2,p,q:List;
Begin
L2 ← Nil;
p ← L;q ← L^.next;
While q≠Nil Do
      Begin
      If q^.message ≠ "" and p^.message ≠ q^.message then
            L2=inserer_queue(q^.personne,q^.message,L2);
      p ← p^.next;
      q ← q^.next;
      End;
identierPersonnes ← L2;
End;
Begin
L ← Nil;
Write("Nom du premier personne: ");
Read(nom1);
Repeat
   Write("Nom du personne suivant: ");
   Read(nom2);
   Write(nom2,", je vais te dire une chose mais garde le secret");
   Write("Message: ");
   Read(message);
   L ← inserer_queue(nom1,message,L);
```

```
    Write(nom2," va garder le secret (1:oui,2:non)?:");
    Read(reponse);
    nom1 ← nom2;
Until reponse=1;
L ← inserer_queue(nom2,"",L);
L2 ← identierPersonnes(L);
Write("Les personnes qui n'ont pas transféré le message honnêtement sont: ");
display(L2);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct node* LISTE ;
typedef struct node {
     char personne[20];
     char message [100];
     LISTE suiv;
     }node;
void display(LISTE l){
    LISTE p ;
    p=l;
    while(p!=NULL){
        printf("\t%s\n",p->personne);
        p=p->suiv;
    }
}
LISTE inserer_queue (char pers[20],char mess[100],LISTE l) {
    LISTE p,d;int i;
    d=malloc(sizeof(node));
    strcpy(d->personne,pers);
    strcpy(d->message,mess);
    d->suiv=NULL;
    if(l==NULL)l=d;
    else{
         p=l;
        while(p->suiv!=NULL) {
            p=p->suiv;
        }
        p->suiv=d;
    }
    return l ;
}
LISTE identierPersonnes(LISTE l){
    LISTE l2,p,q;
    l2=NULL;
    p=l;q=l->suiv;
    while(q!=NULL){
       if(strcmp(q->message,"")!=0 && strcmp(p->message,q->message)!=0)
             l2=inserer_queue(q->personne,q->message,l2);
       p=p->suiv;
       q=q->suiv;
```

```
    }
    return l2;
}
int main(){
  LISTE l,l2;int reponse;char nom1[20],nom2[20];char message[100];char p;
  l=NULL;
  printf("Nom du premier personne: ");
  gets(nom1);
  do{
      printf("Nom du personne suivant: ");
      gets(nom2);
      printf("%s, je vais te dire une chose mais garde le secret\n",nom2);
      printf("Message: ");
      gets(message);
      l=inserer_queue(nom1,message,l);//inserer la premiere personne emeteur
      printf("%s va garder le secret (1:oui,2:non)?:",nom2);
      scanf("%d",&reponse);
      scanf("%c",&p);
      strcpy(nom1,nom2);
  }
  while(reponse==2);
  l=inserer_queue(nom2,"",l);
  l2=identierPersonnes(l);
  printf("\nLes personnes qui n'ont pas transfere le message honnetement
sont:\n");
  display(l2);
  return 0;
}
```

Exercice 241:

On considère un nombre entier positif `n` composé de deux chiffres distincts (les deux chiffres sont différents). On définit la "liste vers 9" associée à `n` comme l'ensemble des nombres obtenus par la méthode suivante :

- On calcule la valeur absolue de la différence entre `n` et son miroir (le miroir de `n` est le nombre formé en inversant les chiffres de `n`) ;
- On répète le même traitement sur le résultat obtenu ;
- On répète ce processus jusqu'à obtenir une différence égale à 9.

L'ensemble constitué par le nombre initial et les résultats des différences forme la "liste vers 9" de `n`.

Il vous est demandé d'établir un algorithme/programme permettant d'introduire un nombre entier positif composé de deux chiffres obligatoirement différents, de générer sa "**liste vers 9"**, et enfin de l'afficher.

La liste vers 9 est implémentée dans cet exercice par une liste chaînées d'entiers.

Utiliser des sous-programmes.

Remarque:

Le miroir d'un nombre `n` peut être calculé par: `10 * (n mod 10) + n div 10`

Exemple :

Soit `n` = 18; son miroir est 81.

- |18 - 81| = 63 ; son miroir est 36
- |63 - 36| = 27 ; son miroir est 72

- |27 -72| = 45 ; son miroir est 54
- |45 - 54| = 9; on arrête

A fin du traitement, la liste vers 9 de 18 est : **18, 63, 27, 45, 9**

Solution :

Algorithme:

```
Algorithm listeVers9;
Type List =^node;
      node=Record
             Begin
             val:integer;
             next:List;
             End;
Var L:List;n:integer;
Function insertTail(L:List,v:integer):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
     q ← L;
     While q^.next≠Nil Do
                q ← q^.next;
     q^.next ← p;
     End;
insertTail ← L;
End;
Procedure display(L:List);
Var p:List;
Begin
Begin
p ← L;
Write("La liste vers 9: ");
While p≠Nil Do
     Begin
     Write(p^.val," ");
     p ← p^.next;
     End;
End;
Function listeVers9(n:integer):List;
Var L:List;m,diff:integer;
Begin
L ← Nil;
L ← insertTail(L,n);
Repeat
   m ← 10*(n mod 10)+n div 10;
   diff ← n-m;
   if(diff<0)diff ← diff*(-1);
   n ← diff;
```

```
   L ← insertTail(L,n);
Until diff=9;
listeVers9 ← L;
End;
Begin
Write("Donner un nombre composé de 2 chiffres: ");
Read(n);
If n<10 or n>99 Then write("Ce nombre ne contient pas 2 chiffres")
Else   Begin
       L ← listeVers9(n);
       display(L);
       End;
End.
```

Programme C:

```
#include <stdio.h>
#include <malloc.h>
typedef struct node* Liste;
typedef struct node{
    int val;
    Liste suiv;
}node;
Liste insertTail(Liste L,int v){
    Liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(L==NULL)L=p;
    else{
       q=L;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return L;
}
void display(Liste L){
    Liste p;
    p=L;
    printf("La liste vers 9:\n");
    while(p!=NULL){
        printf("%d | ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
Liste listeVers9(int n){
    Liste L;
    int m,diff;
    L=NULL;
    L=insertTail(L,n);
    do{
        m=10*(n%10)+n/10;
        diff=n-m;
```

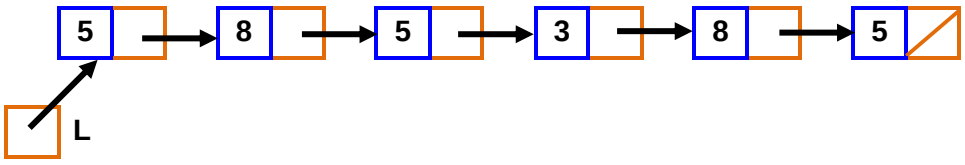
```
        if(diff<0)diff=diff*(-1);
        n=diff;
        L=insertTail(L,n);
    }while(diff!=9);
    return L;
}
int main() {
    int n;Liste L;
    printf("Donner un nombre compose de 2 chiffres: ");
    scanf("%d",&n);
    if(n<10 || n>99)printf("Ce nombre ne contient pas 2 chiffres");
    else{
        L=listeVers9(n);
        display(L);
    }
    return 0;
}
```

Exercice 242:

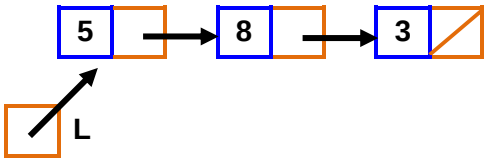
- Écrire une fonction qui permet d'éliminer les éléments redondants dans une liste chaînée passée en paramètres. La fonction retourne la nouvelle liste comme résultat.

  Exemple:

  Avant:



  Après:



- Tester cette fonction dans un programme principale.

Solution :

Algorithme:

```
Algorithm filtrage;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var L:List;
Function insertHead(L:List,v:integer):List;
Var p:List;
Begin
Allocate(p);
p^.val ← v;
```

```
p^.next ← L;
L ← p;
insertHead ← L;
End;
Procedure afficherListe(L:List);
Var p:List;
Begin
Begin
p ← L;
Write("Les éléments de la liste: ");
While p≠Nil Do
      Begin
      Write(p^.val," ");
      p ← p^.next;
      End;
End;
Function eliminerRedondants(L:List):List;
Var p,q,r:List;
Begin
p ← L;
While p≠Nil Do
      Begin
      q ← p;
      r ← q^.next;
      While r≠Nil Do
            Begin
            If p^.val=r^.val then
                  Begin
                  q^.next ← r^.next;
                  Free(r);
                  Fin
            Else q ← q^.next;
            r ← q^.next;
            End;
      p ← p^.next;
      End;
eliminerRedondants ← L;
End;
Begin
L ← Nil;
L ← insertHead(5,L);
L ← insertHead(8,L);
L ← insertHead(3,L);
L ← insertHead(5,L);
L ← insertHead(8,L);
L ← insertHead(5,L);
afficherListe(L);
L ← eliminerRedondants(L);
Write("Après suppression des élements redondants,");
afficherListe(L);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste ;
typedef struct node {
     int val;
     liste suiv;
     }node;
liste insertHead(int v,liste l){
    liste q=malloc(sizeof(node));
    q->val=v;
    q->suiv=l;
    l=q;
    return l;
}
void afficherListe(liste l){
    liste p ;
    p=l;
    printf("Les elements de la liste:\n");
    while(p!=NULL){
        printf("%d  ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
liste eliminerRedondants(liste l){
    liste p,q,r;
    p=l;
    while(p!=NULL){
        q=p;
        r=q->suiv;
        while(r!=NULL){
            if(p->val==r->val){
               q->suiv=r->suiv;
               free(r);
            }
            else q=q->suiv;
            r=q->suiv;
        }
        p=p->suiv;
    }
    return l;
}
int main(){
  liste L=NULL;int a;
  L=insertHead(5,L);
  L=insertHead(8,L);
  L=insertHead(3,L);
  L=insertHead(5,L);
  L=insertHead(8,L);
  L=insertHead(5,L);
  afficherListe(L);
  L=eliminerRedondants(L);
```

```
  printf("Apres suppression des elements redondants,");
  afficherListe(L);
  return 0;
}
```

Exercice 246:

- Ecrire une fonction qui prend en entrée un tableau d'entiers **T**, non ordonné, et qui retourne en sortie une liste chaînées composée des éléments de **T** de façon à ce que les éléments de la liste soient triés dans l'ordre croissant.

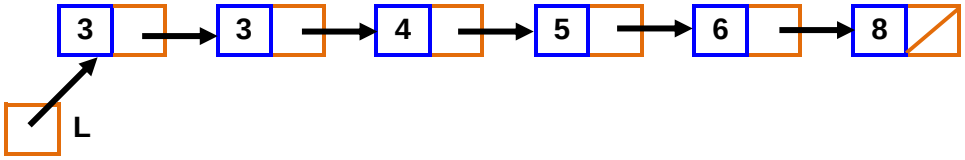
  Notons que le tri doit se faire lors de l'insertion dans la liste chaînée pas préalablement dans le tableau.

  Exemple:

  Tableau:

0	1	2	3	4	5
8	**3**	**5**	**4**	**3**	**6**

  Liste chaînée résultante:



- Ecrire l'algorithme/ programme principal qui permet de saisir au clavier les éléments d'un tableau de 6 entiers, et qui les insère d'une façon ordonnée dans une liste chaînée (en appelant la fonction précédente), et puis affiche cette dernière.

Solution :

Algorithme:

```
Algorithm tri;
Const n=6;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var L:List;t:Tableau[n] d'entier;i:integer;
Procedure afficherListe(L:List);
Var p:List;
Begin
Begin
p ← L;
Write("Les éléments de la liste: ");
While p≠Nil Do
      Begin
      Write(p^.val," ");
      p ← p^.next;
      End;
End;
Function tableauToListe(t:Tableau[n] d'entier):List;
Var L,p,q,r:List;i:integer;
```

```
Begin
L ← Nil;
For i ← 0 to n-1 Do
      Begin
      Allocate(p);
      p^.val ← t[i];
      If L=Nil or L^.val > p^.val then
            Begin
            p^.next ← L;
            L ← p;
            End;
      Else  Begin
            q ← L;r ← L^.next;
            While r≠Nil and r^.val < p^.val Do
                  Begin
                  q ← q^.next;
                  r ← r^.next;
                  End;
            q^.next ← p;
            p^.next ← r;
            End;
      End;
tableauToListe ← L;
End;
Begin
For i ← 0 To n-1 Do
      Begin
      Write("Donner la valeur ",i,": ");
      Read(t[i]);
      End;
L ← tableauToListe(t);
afficherListe(L);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste ;
typedef struct node {
     int val;
     liste suiv;
     }node;
#define n 6
void afficherListe(liste l){
    liste p ;
    p=l;
    printf("Les elements de la liste:\n");
    while(p!=NULL){
        printf("%d  ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
```

```
liste tableauToListe(int t[n]){
    liste l,p,q,r;int i;
    l=NULL;
    for(i=0;i<n;i++){
        p=malloc(sizeof(node));
        p->val=t[i];
        if(l==NULL || l->val>p->val){
            p->suiv=l;
            l=p;
        }
        else{
            q=l;r=l->suiv;
            while(r!=NULL && r->val<p->val){
                    q=q->suiv;
                    r=r->suiv;
            }
            q->suiv=p;
            p->suiv=r;
        }
    }
    return l;

}
int main(){
  int t[n];int i;liste L;
  for(i=0;i<n;i++){
    printf("Donner la valeur %d: ",i);
    scanf("%d",&t[i]);
  }
  L=tableauToListe(t);
  afficherListe(L);
  return 0;
}
```

Exercice 247:

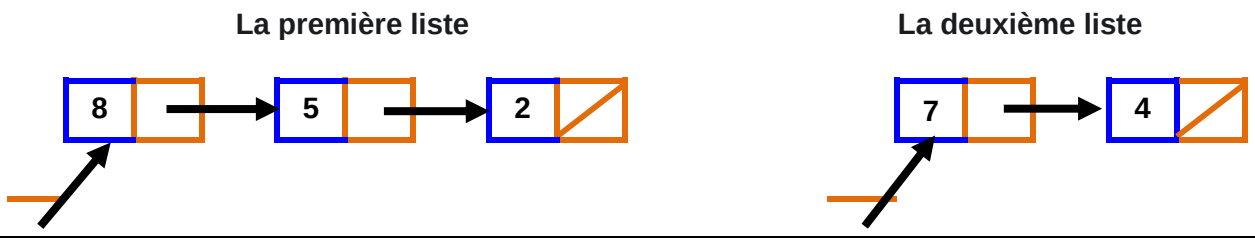
- Ecrire la fonction **`fusionAlternee()`** qui prend en entrée deux listes chaînées et qui crée et renvoie une autre liste chaînée résultante de la fusion alternée des deux listes. Ainsi, la liste fusion est obtenue de la réunion des deux listes d'entrée, mais ses éléments de doivent être alternativement de l'une et de l'autre des deux listes. If Les listes ne sont pas de même longueur, la liste fusion se terminera comme la plus longue des deux listes en entrée.

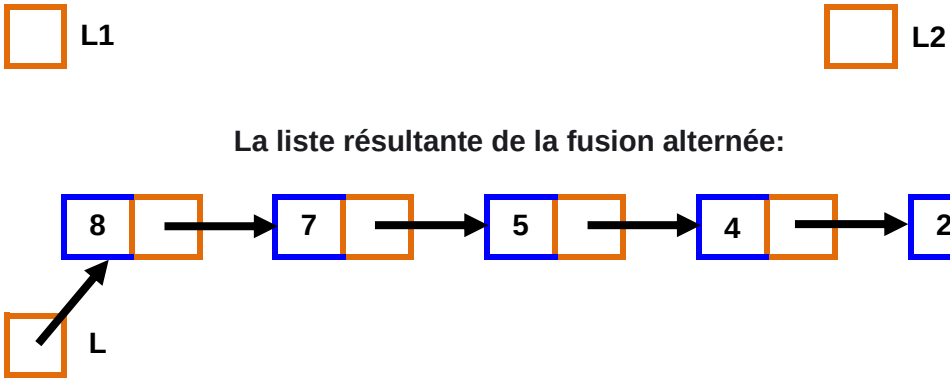
  Remarque:

  On vous demande dans cette exercice une version destructive de la fonction demandée, c'est à dire qu'on ne conservera pas les listes chaînées d'entrée.

  Il n'est pas ainsi autorisé de créer de nouveaux éléments...Enlevez plutôt les éléments des deux listes et les insérez dans la nouvelle liste.

  Exemple:





- Ecrire l'algorithme/ programme principal qui permet de saisir au clavier les éléments de deux listes chainées d'entiers, et qui les fusionne en appelant la fonction **`fusionAlternee()`** précédente, et puis affiche la liste résultante.

Solution :

Algorithme:

```
Algorithm fusion;
Const n=6;
Type List =^node;
       node=Record
              Begin
              val:integer;
              next:List;
              End;
Var L,L1,L2:List;i,n,v:integer;
Function insertTail(L:List;v:integer):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                 q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure display(L:List;message:Chaine);
Var p:List;
Begin
p ← L;
Write(message);
If p=Nil Then write("vide")
Else   Begin
       While p≠Nil Do
            Begin
            Write(p^.val," ");
            p ← p^.next;
            End;
```

```
      End;
End;
Function fusionAlternee(Var L1,L2:List):List;
Var L,p,q:List;
Begin
L ← Nil;
While L1≠Nil or L2≠Nil Do
     Begin
     If L1≠Nil then
          Begin
          p ← L1;
          L1 ← L1^.next;
          If L=Nil then
               Begin
               p^.next ← L;
               L ← p;
               Fin
          Else  Begin
               q ← L;
               While q^.next≠Nil Do
                    q ← q^.next;
               q^.next ← p;
               p^.next ← Nil;
               End;
          End;
     p ← L2;
     If L2≠Nil then
          Begin
          p ← L2;
          L2 ← L2^.next;
          If L=Nil then
               Begin
               p^.next ← L;
               L ← p;
               End;
          Else  Begin
               q ← L;
               While q^.next≠Nil Do
                    q ← q^.next;
               q^.next ← p;
               p^.next ← NULL;
               End;
          End;
     End;
fusionAlternee ← L;
End;
Begin
L1 ← Nil;
L2 ← Nil;
Write("Donner le nombre d'éléments de la liste 1: ");
Read(n);
For i ← 1 to n do
```

```
      Begin
      Write("Donner l'élément ",i,": ");
      Read(v);
      L1 ← insertTail(v,L1);
      End;
display(L1,"La liste 1:");
Write("Donner le nombre d'éléments de la liste 2: ");
Read(n);
For i ← 1 to n do
      Begin
      Write("Donner l'élément ",i,": ");
      Read(v);
      L2 ← insertTail(v,L2);
      End;
display(L2,"La liste 2:");
L ← fusionAlternee(L1,L2);
display(L,"La liste résultante de la fusion alternée:");
End.
```

Programme C:

```
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
    int val;
    liste suiv;
}node;
liste insertTail(int e,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=e;
    p->suiv=NULL;
    if(l==NULL){
        l=p;
    }
    else{
        q=l;
        while(q->suiv!=NULL){
            q=q->suiv;
        }
        q->suiv=p;
    }
    return l;
}
void display(liste l,char message[]){
    liste p;
    printf("%s\n",message);
    if(l==NULL)printf("La liste est vide\n",message);
    else{
        p=l;
        while(p!=NULL){
            printf("%d | ",p->val);
            p=p->suiv;
        }
```

```
        printf("\n");
    }
}
liste fusionAlternee(liste* l1,liste* l2){
    liste l,p,q;
    l=NULL;
    while(*l1!=NULL || *l2!=NULL){
        if(*l1!=NULL){
            p=*l1;
            *l1=(*l1)->suiv;
            if(l==NULL){
                p->suiv=l;
                l=p;
            }
            else{
                q=l;
                while(q->suiv!=NULL)
                    q=q->suiv;
                q->suiv=p;
                p->suiv=NULL;
            }
        }
        p=*l2;
        if(*l2!=NULL){
            p=*l2;
            *l2=(*l2)->suiv;
            if(l==NULL){
                p->suiv=l;
                l=p;
            }
            else{
                q=l;
                while(q->suiv!=NULL)
                    q=q->suiv;
                q->suiv=p;
                p->suiv=NULL;
            }
        }
    }
    return l;
}
main(){
    liste L1,L2,L;
    L1=NULL;
    L2=NULL;
    int n,v,i;
    printf("Donner le nombre d'elements de la liste 1: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L1=insertTail(v,L1);
    }
```

```
    display(L1,"La liste 1:");
    printf("Donner le nombre d'elements de la liste 2: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L2=insertTail(v,L2);
    }
    display(L2,"La liste 2:");
    L=fusionAlternee(&L1,&L2);
    display(L,"La liste resultante de la fusion alternee:");
}
```

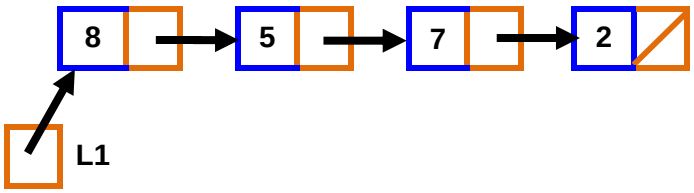
Exercice 248:

- Ecrire la procédure **`difference()`** qui prend en entrée deux listes chaînées **`L1`** et **`L2`**, et qui permet de construire la liste **`L = L1 - L2`** contenant tous les éléments appartenant à **`L1`** et n'appartenant pas à **`L2`**.

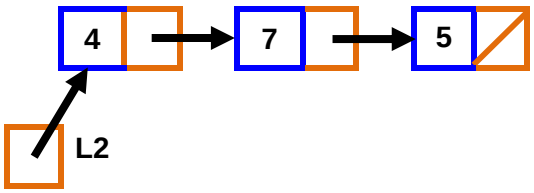
  On demande une version non-destructive de la procédure souhaitée, c'est à dire que les listes chaînée d'entrée doivent être conservées.

  Exemple:

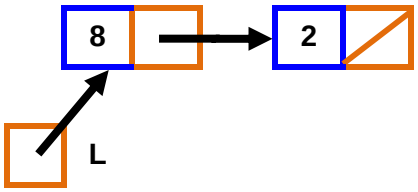
La première liste



La deuxième liste



La liste différence (`L = L1 - L2`):



- Ecrire l'algorithme/programme principal qui permet de saisir au clavier les éléments de deux listes chainées d'entiers, et qui construit la liste résultante de leur différence, et l'affiche.

Solution :

Algorithme:

```
Algorithm difference;
Const n=6;
Type List =^node;
       node=Record
               Begin
               val:integer;
               next:List;
               End;
Var L,L1,L2:List;i,n,v:integer;
Function insertTail(L:List;v:integer):List;
Var p,q:List;
```

```
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                  q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure display(L:List;message:Chaine);
Var p:List;
Begin
p ← L;
Write(message);
If p=Nil Then write("vide")
Else   Begin
       While p≠Nil Do
            Begin
            Write(p^.val," ");
            p ← p^.next;
            End;
       End;
End;
Procedure difference(L1,L2:List;Var L:List);
Var p,q:List;tr:Boolean;
Begin
L ← NULL;
p ← L1;
While p≠Nil Do
      Begin
      q ← L2;
      tr ← False;
      While q≠Nil and tr=False Do
            Begin
            If p^.val=q^.val Then tr ← True
            Else q ← q^.next;
            End;
      If tr=False Then L ← insertTail(p^.val,L);
      p ← p^.next;
      End;
End;
Begin
L1 ← Nil;
L2 ← Nil;
Write("Donner le nombre d'éléments de la liste 1: ");
Read(n);
For i ← 1 to n do
      Begin
```

```
      Write("Donner l'élément ",i,": ");
      Read(v);
      L1 ← insertTail(v,L1);
      End;
display(L1,"La liste 1:");
Write("Donner le nombre d'éléments de la liste 2: ");
Read(n);
For i ← 1 to n do
      Begin
      Write("Donner l'élément ",i,": ");
      Read(v);
      L2 ← insertTail(v,L2);
      End;
display(L2,"La liste 2:");
difference(L1,L2,L);
display(L,"La liste L = L1 - L2:");
End.
```

Programme C:

```
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
    int val;
    liste suiv;
}node;
liste insertTail(int e,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=e;
    p->suiv=NULL;
    if(l==NULL){
        l=p;
    }
    else{
        q=l;
        while(q->suiv!=NULL){
            q=q->suiv;
        }
        q->suiv=p;
    }
    return l;
}
void display(liste l,char message[]){
    liste p;
    printf("%s\n",message);
    if(l==NULL)printf("La liste est vide\n",message);
    else{
        p=l;
        while(p!=NULL){
            printf("%d | ",p->val);
            p=p->suiv;
        }
        printf("\n");
```

```
    }
}
void difference(liste l1,liste l2,liste* l){
    liste p,q;int tr;
    *l=NULL;
    p=l1;
    while(p!=NULL){
        q=l2;
        tr=0;
        while(q!=NULL && tr==0){
            if(p->val==q->val)tr=1;
            else q=q->suiv;
        }
        if(tr==0)*l=insertTail(p->val,*l);
        p=p->suiv;
    }
}
main(){
    liste L1,L2,L;
    L1=NULL;
    L2=NULL;
    int n,v,i;
    printf("Donner le nombre d'elements de la liste 1: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L1=insertTail(v,L1);
    }
    display(L1,"La liste 1:");
    printf("Donner le nombre d'elements de la liste 2: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L2=insertTail(v,L2);
    }
    display(L2,"La liste 2:");
    difference(L1,L2,&L);
    display(L,"La liste L=L1-L2:");
}
```

Exercice 249:

- Ecrire la fonction **`caracterePlusFrequent()`** qui prend en entrée une liste chaînée de caractères représentant une chaîne de caractères, et qui recherche le caractère le plus fréquent dans la liste (celui qui a le plus grand nombre d'occurrences).
- Ecrire l'algorithme/programme principal qui permet de saisir au clavier une chaîne de caractères tout en stockant ses caractères dans une liste chaînée, et qui en cherche le caractère le plus fréquent en appelant la fonction **`caracterePlusFrequent()`** précédente.

Solution :

Algorithme:

```
Algorithm difference;
Const n=6;
Type List =^node;
       node=Record
              Begin
              val:character;
              next:List;
              End;
Var L:List;n,i:integer;v,carPlusFreq:character;
Function insertTail(L:List;v:character):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                 q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure display(L:List;message:Chaine);
Var p:List;
Begin
p ← L;
Write(message);
If p=Nil Then write("vide")
Else   Begin
       While p≠Nil Do
            Begin
            Write(p^.val," ");
            p ← p^.next;
            End;
       End;
End;
Function caracterePlusFrequent(L:List):character;
Var p,q:List;c,carPlusFreq:character;nb,nbMax:integer;
Begin
p ← L;
nbMax ← 0;
While p≠Nil Do
      Begin
      c ← p^.val;
      nb ← 1;
      q ← p^.next;
      While q≠Nil Do
            Begin
            If q^.val=c Then nb ← nb+1;
            q ← q^.next;
```

```
            End;
      If nb>nbMax then
            Begin
            nbMax ← nb;
            carPlusFreq ← c;
            End;
      p ← p^.next;
      End;
caracterePlusFrequent ← carPlusFreq;
End;
Begin
L ← Nil;
Write("Veuillez saisir une chaine de caractères: ");
Read(v);
Write(v);
While v≠13 Do   /*13 c'est le code ASCCI de la touche "Entrée"*/
      Begin
      L ← insertTail(v,L);
      Read(v);
      Write(v);
      End;
display(L,"La liste :");
carPlusFreq ← caracterePlusFrequent(L);
Write("Le caractere le plus fréquent est ",carPlusFreq);
End.
```

Programme C:

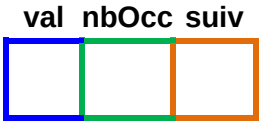
```
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
    char val;
    liste suiv;
}node;
liste insertTail(char e,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=e;
    p->suiv=NULL;
    if(l==NULL){
        l=p;
    }
    else{
        q=l;
        while(q->suiv!=NULL){
            q=q->suiv;
        }
        q->suiv=p;
    }
    return l;
}
void display(liste l,char message[]){
    liste p;
    printf("%s\n",message);
```

```
    if(l==NULL)printf("La liste est vide\n",message);
    else{
        p=l;
        while(p!=NULL){
            printf("%c | ",p->val);
            p=p->suiv;
        }
        printf("\n");
    }
}
char caracterePlusFrequent(liste L){
    liste p,q;
    char c,carPlusFreq;int nb,nbMax;
    carPlusFreq='\0';
    p=L;
    nbMax=0;
    while(p!=NULL){
        c=p->val;
        nb=1;
        q=p->suiv;
        while(q!=NULL){
            if(q->val==c){
                nb++;
            }
            q=q->suiv;
        }
        if(nb>nbMax){
            nbMax=nb;
            carPlusFreq=c;
        }
        p=p->suiv;
    }
    return carPlusFreq;
}
main(){
    liste L;
    L=NULL;
    int n,i;
    char v,carPlusFreq;
    printf("Veuillez saisir une chaine de caracteres: ");
    v=getch();
    printf("%c",v);
    while(v!=13){   /*13 c'est le code ASCCI de la touche "Entrée"*/
        L=insertTail(v,L);
        v=getch();
        printf("%c",v);
    }
    display(L,"La liste :");
    carPlusFreq=caracterePlusFrequent(L);
    if(carPlusFreq!='\0')
        printf("Le caractere le plus frequent est '%c'",carPlusFreq);
    else printf("La liste est vide");
}
```

Exercice 250:

Soit une liste chaînée **L**. On appelle liste compressée, correspondante à **L**, la liste chaînée composée des éléments de **L** sans redondances. De ce fait, aucun élément n'apparait plus d'une fois dans une liste compressée.

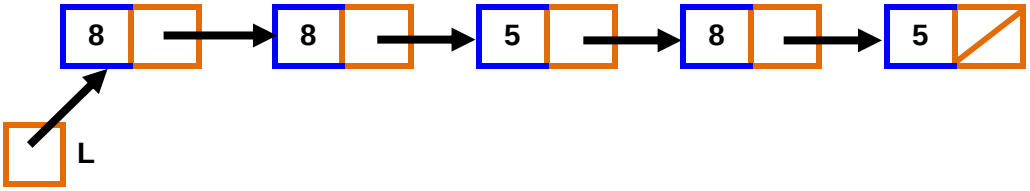
Il est à noter que, les éléments de la liste compressée n'ont pas la même structure que les éléments d'une liste chaînée habituelle. Chaque élément de la liste compressée possède en plus des champs **val** et **suiv**, un autre champ (nommé **nbOcc** par exemple) stockant le nombre d'occurrences de cet élément dans **L**. Il est donc de la forme suivante:



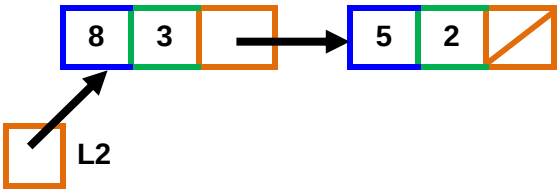
- Donner la/les structure(s) de données modélisant ce problème.
- Ecrire la fonction qui, étant donnée une liste chaînée d'entiers **L**, retourne la liste compressée correspondante. Notez bien qu'il ne faut pas modifier la liste d'entrée **L**.
- Ecrire l'algorithme/ptogramme principal qui permet de saisir au clavier les éléments d'une liste chaînée, et qui construit la liste compressée correspondante en appelant la fonction précédente, et l'affiche.

Exemple:

Liste chainée:



Liste compressée correspondante:



Solution :

Algorithme:

```
Algorithm difference;
Const n=6;
Type List =^node;
       node=Record
               Begin
               val:integer;
               next:List;
               End;
Type ListeCompress =^node2;
       node2=Record
               Begin
               val:integer;
               nbOcc:integer;
```

```
            next:ListCompress;
            End;

Var L:List;L2:ListComporess;n,v,i:integer;
Procedure afficherListe(L:List);
Var p:List;
Begin
p ← L;
Write(Les éléments de la liste:);
While p≠Nil Do
      Begin
      Write(p^.val," ");
      p ← p^.next;
      End;
End;
Procedure afficherListeCompressee(L:ListCompress);
Var p:ListCompress;
Begin
p ← L;
Write(Les éléments de la liste compressée:);
While p≠Nil Do
      Begin
      Write(p^.val," | ",p^.nbOcc," fois");
      p ← p^.next;
      End;
End;
Function insertTail(v:integer;L:List):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
     q ← L;
     While q^.next≠Nil Do
               q ← q^.next;
     q^.next ← p;
     End;
insertTail ← L;
End;
Function insertTail2(v,nb:integer;L:ListCompress):ListCompress;
Var p,q:ListCompress;
Begin
Allocate(p);
p^.val ← v;
p^.nbOcc ← nb;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
     q ← L;
     While q^.next≠Nil Do
```

```
                q ← q^.next;
      q^.next ← p;
      End;
insertTail2 ← L;
End;
Function compresser(L:List):ListCompress;
Var L2,p2:ListCompress;p,q:List;nb:integer;tr:Boolean;
Begin
L2 ← Nil;
p ← L;
While p≠Nil Do
      Begin
      p2 ← L2;
      tr ← False;
      While p2≠Nil and tr=False Do
            Begin
            If p2^.val=p^.val Then tr ← True
            Else p2 ← p2^.next;
            End;
      If tr=True Then p2^.nbOcc ← p^.nbOcc+1
      Else L2 ← insertTail2(p^.val,1,L2);
      p ← p^.next;
      End;
compresser ← L2;
End;
Begin
L ← Nil;
Write("Donner le nombre d'éléments de la liste: ");
Read(n);
For i ← 1 to n do
      Begin
      Write("Donner l'élément ",i,": ");
      Read(v);
      L ← insertTail(v,L);
      End;
afficherListe(L);
L2 ← compresser(L);
afficherListeCompressee(L2);
End.
```

Programme C:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste ;
typedef struct node {
      int val;
      liste suiv;
      }node;
typedef struct node2* listeCompress ;
typedef struct node2 {
      int val;
      int nbOcc;
```

```
     listeCompress suiv;
     }node2;

void afficherListe(liste l){
    liste p ;
    p=l;
    printf("Les elements de la liste:\n");
    while(p!=NULL){
        printf("%d ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void afficherListeCompressee(listeCompress l){
    listeCompress p ;
    p=l;
    printf("Les elements de la liste compressee:\n");
    while(p!=NULL){
        printf("%d | %d fois\n",p->val,p->nbOcc);
        p=p->suiv;
    }
    printf("\n");
}
liste insertTail(int v,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
listeCompress insertTail2(int v,int nb,listeCompress l){
    listeCompress p,q;
    p=malloc(sizeof(node2));
    p->val=v;
    p->nbOcc=nb;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
listeCompress compresser(liste l){
    listeCompress l2,p2;liste p,q;int nb,tr;
    l2=NULL;
```

```
    p=l;
    while(p!=NULL){
        p2=l2;tr=0;
        while(p2!=NULL && tr==0){
            if(p2->val==p->val)tr=1;
            else p2=p2->suiv;
        }
        if(tr==1)p2->nbOcc++;
        else l2=insertTail2(p->val,1,l2);
        p=p->suiv;
    }
    return l2;
}
main(){
    liste L;
    listeCompress L2;
    L=NULL;
    int n,v,i;
    printf("Donner le nombre d'elements de la liste: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L=insertTail(v,L);
    }
    afficherListe(L);
    L2=compresser(L);
    afficherListeCompressee(L2);
}
```

Exercice 251:

1. Ecrire la fonction **`premiereOcc(L,v)`** qui renvoie l'adresse de la première occurrence de la valeur **`v`** dans la liste **`L`**. Si **`v`** n'existe pas dans **`L`**, la fonction doit retourner **`Nil`**.
2. Ecrire la fonction **`insererApres(L,v,e)`** qui insère la valeur **`v`** après la première occurrence de la valeur **`e`** dans la liste chaînée d'entiers **`L`**. Si aucune auccurrence de **`e`** n'existe, la valeur **`v`** est insérée au Begin de la liste si elle est inférieure à la valeur de la tête, et à la fin de la liste sinon.
3. Tester cette dernière dans un algorithme/ programme principal.

Solution :

Algorithme:

```
Algorithm insertion;
Type List =^node;
       node=Record
               Begin
               val:integer;
               next:List;
               End;
Var L,L1,L2:List;i,n,v:integer;
Function insertTail(L:List;v:integer):List;
Var p,q:List;
Begin
```

```
Allocate(p);
p^.val ← v;
p^.next ← Nil;
If L=Nil Then   L ← p;
Else  Begin
      q ← L;
      While q^.next≠Nil Do
                  q ← q^.next;
      q^.next ← p;
      End;
insertTail ← L;
End;
Procedure afficherListe(L:List);
Var p:List;
Begin
p ← L;
Write(Les éléments de la liste:);
While p≠Nil Do
       Begin
       Write(p^.val," ");
       p ← p^.next;
       End;
End;
Function premiereOcc(L:List;v:integer):List;
Var p,q:List;tr:Boolean;
Begin
p ← L;
tr ← False;
While p≠Nil and tr≠False Do
      Begin
      If p^.val=v Then tr ← True
      Else p ← p^.next;
      End;
premiereOcc ← p;
End;
Function insererApres(L:List;v,e:integer):List;
Var p,q:List;
Begin
Allocate(p);
p^.val ← v;
q ← premiereOcc(L,e);
If q≠Nil then
      Begin
      p^.next ← q^.next;
      q^.next ← p;
      Fin
Else  Begin
      If L=Nil or L^.val>v then
            Begin
            p^.next ← L;
            L ← p;
            Fin
```

```
      Else Begin
           q ← L;
           While q^.next≠Nil Do
                q ← q^.next;
           q^.next ← p;
           p^.next ← Nil;
           End;
      End;
insererApres ← L;
End;
Begin
L ← Nil;
Write("Donner le nombre d'éléments de la liste: ");
Read(n);
For i ← 1 to n do
      Begin
      Write("Donner l'élément ",i,": ");
      Read(v);
      L ← insertTail(v,L);
      End;
display(L);
Write("Donner la valeur à insérer: ");
Read(v);
Write("Après quelle valeur voulez vous l'insérer: ");
Read(e);
L ← insererApres(L,v,e);
Write("Après l'insertion,");
display(L);
End.
```

Programme C:

```c
#include <stdlib.h>
typedef struct node* Liste;
typedef struct node{
    int val;
    Liste suiv;
}node;
Liste insertTail(int e,Liste l){
    Liste p,q;
    p=malloc(sizeof(node));
    p->val=e;
    p->suiv=NULL;
    if(l==NULL){
        l=p;
    }
    else{
        q=l;
        while(q->suiv!=NULL){
            q=q->suiv;
        }
        q->suiv=p;
    }
    return l;
```

```
}
void afficherListe(Liste l){
    Liste p ;
    p=l;
    printf("Les elements de la liste:\n");
    while(p!=NULL){
        printf("%d ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
Liste premiereOcc(Liste L,int v){
    Liste p,q;int tr;
    p=L;tr=0;
    while(p!=NULL && tr==0){
        if(p->val==v)tr=1;
        else p=p->suiv;
    }
    return p;
}
Liste insererApres(Liste L,int v,int e){
    Liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    q=premiereOcc(L,e);
    if(q!=NULL){
        p->suiv=q->suiv;
        q->suiv=p;
    }
    else{
        if(L==NULL || L->val>v){
            p->suiv=L;
            L=p;
        }
        else{
            q=L;
            while(q->suiv!=NULL){
                q=q->suiv;
            }
            q->suiv=p;
            p->suiv=NULL;
        }
    }
    return L;
}
main(){
    Liste L;
    L=NULL;
    int n,v,e,i;
    printf("Donner le nombre d'elements de la liste: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
```

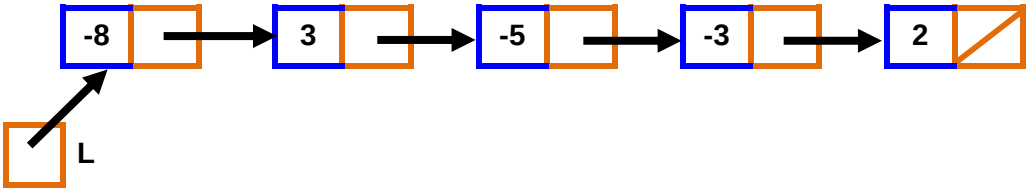
```
        scanf("%d",&v);
        L=insertTail(v,L);
    }
    afficherListe(L);
    printf("Donner la valeur a inserer: ");
    scanf("%d",&v);
    printf("Apres quelle valeur voulez vous l'inserer: ");
    scanf("%d",&e);
    L=insererApres(L,v,e);
    printf("Apres l'insertion,");
    afficherListe(L);
}
```

Exercice 251:

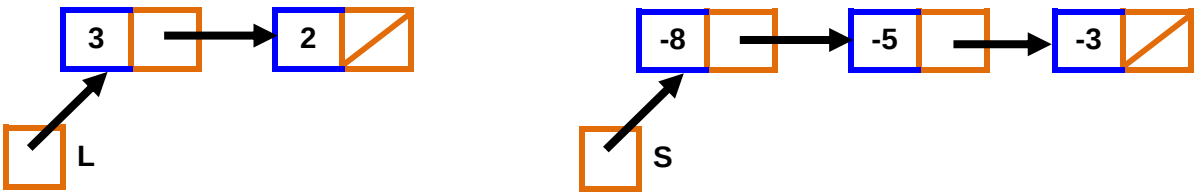
- Ecrire la procédure **`Eclater(L,S)`** qui supprime à partir de **`L`** les valeurs négatives et les met dans une nouvelle liste **`S`**. Les valeurs négatives doivent être insérées dans la nouvelle liste **`S`** dans le même ordre de leur apparition dans la liste originale **`L`**.
- Ecrire la fonction principale **`main()`** qui permet de saisir **`n`** valeurs au clavier et les mettre dans une liste chaînée **`L`** (Le nombre **`n`** est donné par l'utilisateur), affiche la liste de départ **`L`**, appelle la procédure **`Eclater()`**, et qui affiche finalement les 2 listes résultantes.

Exemple:

Liste chainée de départ `L`:



Après l'éclatement, on aura 2 listes `L` et `S`:



Réponse:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertHead(liste l,int v){
    liste p;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=l;
    l=p;
    return l;
```

```
}
void display(liste l){
    liste p;
    p=l;
    while(p!=NULL){
        printf("%d | ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void Eclater(liste *L,liste *S){
    liste p,prec,q,r;
    *S=NULL;
    while(*L!=NULL && (*L)->val<0){
            p=*L;
            *L=(*L)->suiv;
            p->suiv=NULL;
            if(*S==NULL)*S=p;
            else{
                q=*S;
                while(q->suiv!=NULL)q=q->suiv;
                q->suiv=p;
            }
    }
    if(*L!=NULL){
        prec=*L;p=(*L)->suiv;
        while(p!=NULL){
           if(p->val<0){
                prec->suiv=p->suiv;
                r=p;
                p=p->suiv;
                r->suiv=NULL;
                if(*S==NULL)*S=r;
                else{
                    q=*S;
                    while(q->suiv!=NULL)q=q->suiv;
                    q->suiv=r;
                }
           }
           else{
                prec=p;
                p=p->suiv;
           }
        }
    }
}
int main(){
    int n,i,v;liste L,S;
    L=NULL;
    printf("Donner le nombre d'elements a saisir: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
```

```
        scanf("%d",&v);
        L=insertHead(L,v);
    }
    printf("La liste de depart L:\n");
    display(L);
    Eclater(&L,&S);
    printf("Apres l'eclatement:\n");
    printf("La liste L:\n");
    display(L);
    printf("La liste S:\n");
    display(S);
}
```

Exercice 252:

- Ecrire la fonction **`palindrome()`** qui prend en entrée une liste chaînée de caractères représentant un mot, et qui vérifie si ce mot est un palindrome ou non. On dit qu'un mot est un palindrome s'il se lit de la même façon de gauche à droite et de droite à gauche.

  Exemple:

  Les mots: "été", "radar", "laval", sont des palindromes.

- Ecrire la fonction principale **`main()`** qui permet de saisir au clavier un mot en stockant ses lettres dans une liste chaînée de caractères, et qui teste ensuite s'il s'agit d'un palindrome ou pas en appelant la fonction précédente.

Réponse:

```
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
    char val;
    liste suiv;
}node;
liste insertTail(char e,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=e;
    p->suiv=NULL;
    if(l==NULL){
        l=p;
    }
    else{
        q=l;
        while(q->suiv!=NULL){
            q=q->suiv;
        }
        q->suiv=p;
    }
    return l;
}
void display(liste l){
    liste p;
    if(l==NULL)printf("Le mot est vide\n");
    else{
        printf("\nLe mot saisi est:\n");
```

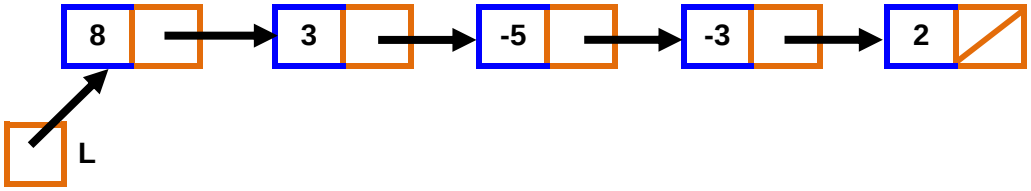
```
        p=l;
        while(p!=NULL){
            printf("%c",p->val);
            p=p->suiv;
        }
        printf("\n");
    }
}
int palindrome(liste L){
    int pal;
    liste p,q,r;
    if(L==NULL)return 0;
    else{
       p=L;
       q=NULL;
       pal=1;
       while(p!=q && p->suiv!=q && pal==1){
            r=p;
            while(r->suiv!=q)
                r=r->suiv;
            q=r;
            if(p->val != r->val)pal=0;
            else p=p->suiv;
       }
       return pal;
    }
}
main(){
    liste L;
    L=NULL;
    char v;
    printf("Veuillez saisir une chaine de caracteres: ");
    v=getch();
    printf("%c",v);
    while(v!=13){   /*13 c'est le code ASCCI de la touche "Entrée"*/
        L=insertTail(v,L);
        v=getch();
        printf("%c",v);
    }
    display(L);
    if(palindrome(L)==1)printf("Le mot est un palindrome");
    else printf("Le mot n'est pas un palindrome");
}
```

Exercice 253:

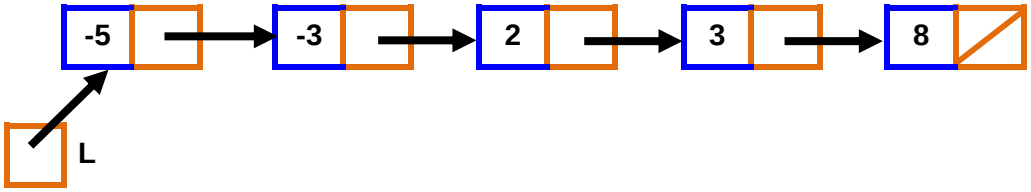
- Ecrire la procédure **`Trier()`** qui permet de trier une liste chaînée passée en paramètres dans l'ordre croissant. Le tri s'effectue par permutation de valeurs d'éléments. De ce fait, le premier élément contiendra la plus petite valeur dans la liste, le deuxième élément contiendra la deuxième petite valeur, et ainsi de suite.

  Exemple:

  Liste chainée de départ :



Après le tri, la liste devient:



- Ecrire la fonction principale **`main()`** qui permet de remplir une liste chaînée par une série de nombres saisis au clavier et se terminant par 0 (le 0 n'est pas compté), et qui trie cette liste en appelant la procédure **`Trier()`** précédente. Le programme doit afficher la liste avant et après le tri.

Réponse:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertTail(liste l,int v){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
void display(liste l){
    liste p;
    p=l;
    while(p!=NULL){
        printf("%d | ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void Trier(liste* l){
    liste p,q,min;int a;
    p=*l;
    while(p!=NULL){
        min=p;
        q=p->suiv;
        while(q!=NULL){
```

```
            if(q->val < min->val){
                min=q;
            }
            q=q->suiv;
        }
        a=p->val;
        p->val=min->val;
        min->val=a;
        p=p->suiv;
    }
}
int main(){
    int v;liste L,S;
    L=NULL;
    do{
        printf("Donner un nombre a inserer (taper 0 pour quiter): ");
        scanf("%d",&v);
        if(v!=0)L=insertTail(L,v);
    }while(v!=0);
    printf("La liste de depart L:\n");
    display(L);
    Trier(&L);
    printf("La liste triee:\n");
    display(L);
}
```

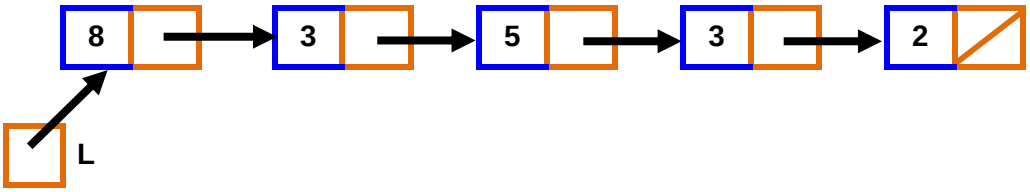
Exercice 254:

- Ecrire une la fonction **`inverserListe()`** qui permet d'inverser une liste chaînée **`L`** passée en paramètre, en manipulant seulement ses pointeurs de liaison.

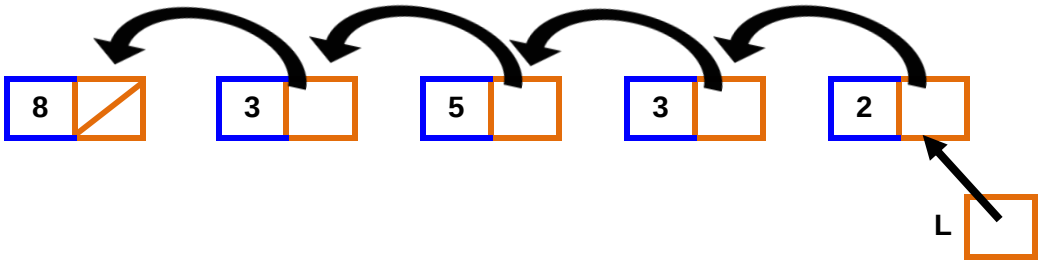
  L'idée est de parcourir la liste chaînée **`L`**, et d'associer, pour chaque élément, à son champ **`suiv`** l'adresse de l'élément qui le précède au lieu de celle de l'élément qui le suit. Toutefois il faut démontrer un peu de technicité pour réussir ceci.

  Exemple:

Liste chainée de départ:



Liste après l'inversement:



- Ecrire la fonction principale **`main()`** qui permet de saisir au clavier **`n`** valeurs entières tout en les mettant dans une liste chaînée (Le nombre **`n`** est donné par l'utilisateur), et qui inverse cette liste en appelant la fonction **`inverserListe()`** précédente. Le programme doit afficher la liste avant et après l'inversement.

Réponse:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste ;
typedef struct node {
     int val;
     liste suiv;
     }node;
liste insertTail(int v,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
        q=l;
        while(q->suiv!=NULL)q=q->suiv;
        q->suiv=p;
    }
    return l;
}
void afficherListe(liste l){
    liste p ;
```

```
    p=l;
    while(p!=NULL){
        printf("%d  ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
liste              inverserListe2(liste              l){//Méthode              de
https://www.theccoder.com/fr/exercices-c/listes-chainees/exercice-3/22-langage-
c/exercices/listes-chainees
    liste p,q,r;
    if(l==NULL)return l;
    else{
        p=l;
        while(p!=NULL){
           r=p->suiv;
           if(p==l)p->suiv=NULL;
           else p->suiv=q;
           q=p;
           p=r;
        }
        l=q;
    return l;
    }
}
liste inverserListe(liste l){//Méthode de Kefali
    liste p,q,r;
    if(l==NULL)return l;
    else{
        r=l;p=l->suiv;
        while(p!=NULL){
           q=p->suiv;
           p->suiv=l;
           l=p;
           p=q;
        }
        r->suiv=NULL;
    return l;
    }
}
int main(){
  int n,i,v;liste L;
    L=NULL;
    printf("Donner le nombre d'elements a saisir: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L=insertTail(v,L);
    }
  printf("La liste de depart:\n");
  afficherListe(L);
  L=inverserListe(L);
```

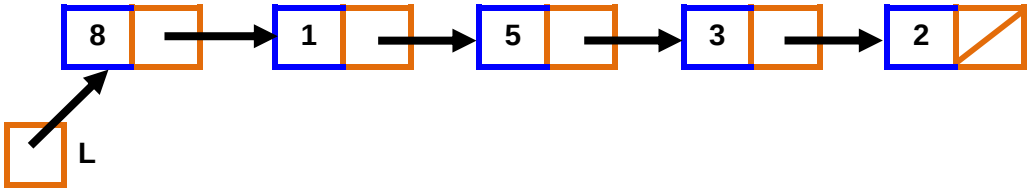
```
  printf("La liste inverse:\n");
  afficherListe(L);
  return 0;
}
```

Exercice 255:

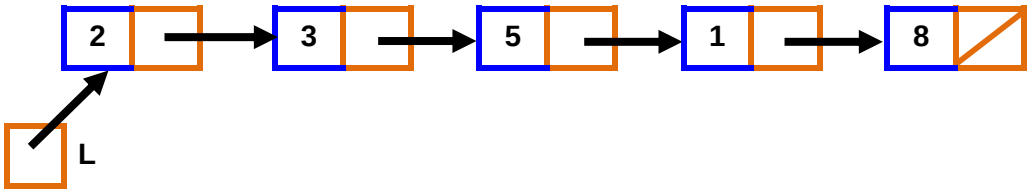
- Ecrire la procédure **`inverserListe()`** qui permet d'inverser une liste chaînée **`L`** passée en paramètres, mais cette fois-ci par permutation des valeurs des éléments (le premier avec le dernier, le deuxième avec l'avant dernier, et ainsi de suite) et non pas par manipulation de pointeurs de liaison.

  Exemple:

  Liste chainée de départ:



  Liste après l'inversement:



- Ecrire la fonction principale **`main()`** qui permet de saisir au clavier **`n`** valeurs entières tout en les mettant dans une liste chaînée (Le nombre **`n`** est donné par l'utilisateur), et qui inverse cette liste en appelant la procédure **`inverserListe()`** précédente. Le programme doit afficher la liste avant et après l'inversement.

Solution:

```
#include <stdio.h>
typedef struct node* liste;

typedef struct node{
               int val;
               liste suiv;
}node;
liste insertTail(int v,liste L){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(L==NULL)L=p;
    else{
        q=L;
        while(q->suiv!=NULL)q=q->suiv;
        q->suiv=p;
    }
    return L;
}
void afficherListe(liste L){
    liste p;
```

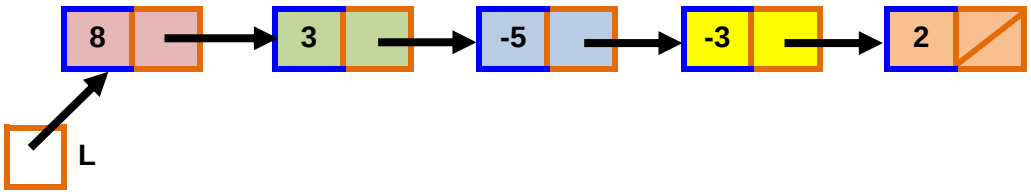
```
    p=L;
    while(p!=NULL){
        printf("%d ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void inverserListe(liste *L){
    int a;
    liste p,t,q;
    if(*L==NULL);
    else{
       t=*L;
       q=NULL;
       while(t!=q && t->suiv!=q){
            p=t;
            while(p->suiv!=q)
                p=p->suiv;
            q=p;
            a=t->val;
            t->val=p->val;
            p->val=a;
            t=t->suiv;
       }
    }
}
int main(){
  int n,i,v;liste L;
    L=NULL;
    printf("Donner le nombre d'elements a saisir: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&v);
        L=insertTail(v,L);
    }
  printf("La liste de depart:\n");
  afficherListe(L);
  inverserListe(&L);
  printf("La liste inverse:\n");
  afficherListe(L);
  return 0;
}
```

Exercice 256:

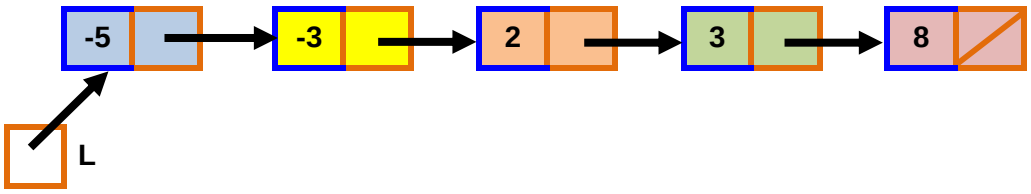
- Ecrire la procédure **`Trier()`** qui prend en paramètres une liste chaînée d'entiers et qui la trie dans l'ordre croissant. Contrairement à l'exercice 248, le tri doit s'effectuer ici par permutation des éléments entiers et pas uniquement permutation de leur valeur. Ici aussi vous devez manipuler les pointeurs de liaison pour permuter un élément avec un autre.

Exemple:

Liste chainée de départ :



Après le tri, la liste devient:



- Ecrire la fonction principale **`main()`** qui permet de remplir une liste chaînée par une série de nombres saisis au clavier et se terminant par 0 (le 0 n'est pas compté), et qui trie cette liste en appelant la procédure **`Trier()`** précédente. Le programme doit afficher la liste avant et après le tri.

Réponse:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertTail(liste l,int v){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
void display(liste l){
    liste p;
    p=l;
    while(p!=NULL){
        printf("%d | ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void Trier(liste* l){
    liste p,q,min,prec,precMin,precP;int a;
    p=*l;
```

```
    int i=0;
    while(p!=NULL){
        min=p;
        prec=p;
        q=p->suiv;
        while(q!=NULL){
            if(q->val < min->val){
                precMin=prec;
                min=q;
            }
            prec=prec->suiv;
            q=q->suiv;
        }
        if(min!=p){
            if(min!=p->suiv){
                q=min->suiv;
                min->suiv=p->suiv;
                p->suiv=q;
                precMin->suiv=p;
                if(p==*l){
                    p=min;
                    *l=p;
                }
                else{
                    precP->suiv=min;
                    p=min;
                }
            }
            else{
                q=min->suiv;
                min->suiv=p;
                p->suiv=q;
                if(p==*l){
                    p=min;
                    *l=p;
                }
                else{
                     precP->suiv=min;
                     p=min;
                }
            }
        }
        precP=p;
        p=p->suiv;
    }
}
int main(){
    int v;liste L,S;
    L=NULL;
    do{
        printf("Donner un nombre a inserer (taper 0 pour quiter): ");
        scanf("%d",&v);
        if(v!=0)L=insertTail(L,v);
```

```
    }while(v!=0);
    printf("La liste de depart L:\n");
    display(L);
    Trier(&L);
    printf("La liste triee:\n");
    display(L);
}
```

Exercice 257:

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. La suite de Fibonacci (F_n) est définie par récurrence par :

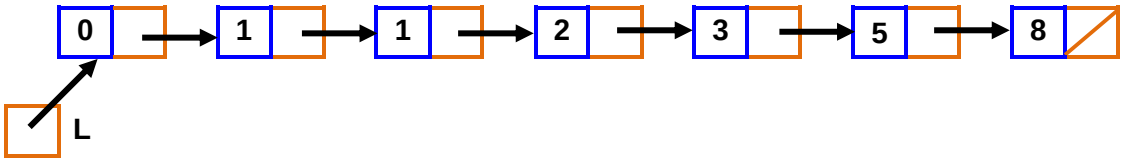
$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \quad \text{pour } n \geq 2;$$

Dans cet exercice, on souhaite établir un programme permettant de vérifier si une suite numérique stockée dans une liste chaînée est la suite de Fibonacci. Ainsi, il vous ai demandé de:

- Ecrire la fonction **`Fibonacci()`** qui prend en entrée une liste chaînée représentant une suite numérique et qui renvoie True si cette suite est la suite de Fibonacci et renvoie False dans le cas contraire.

  Exemple:

  La liste chaînée suivante correspond à la suite de Fibonacci d'ordre 6:



- Ecrire la fonction principale **`main()`** qui permet de saisir au clavier les termes d'une suite numérique en les stockant dans une liste chaînée (la saisi se termine par l'introduction d'une valeur négative), et qui nous informe si cette suite est la suite de Fibonacci ou non.

Réponse:

```
#include <stdio.h>
typedef struct node* liste;
typedef struct node{
                int val;
                liste suiv;
}node;
liste insertTail(int v,liste L){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(L==NULL)L=p;
    else{
        q=L;
        while(q->suiv!=NULL)q=q->suiv;
        q->suiv=p;
    }
    return L;
}
void afficherListe(liste L){
```

```
    liste p;
    p=L;
    while(p!=NULL){
        printf("%d ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
int Fibonacci(liste l){
    liste p,q,r;int fibo;
    if(l==NULL)return 0;
    else if(l->suiv==NULL){
            if(l->val==0)
                return 1;
            else return 0;
    }
    else{
        r=l;
        q=l->suiv;
        p=q->suiv;
        if(p==NULL){
            if(q->val==1)
                return 1;
            else return 0;
        }
        else{
            fibo=1;
            while(p!=NULL && fibo==1){
                if(p->val==q->val+r->val){
                    r=q;
                    q=p;
                    p=p->suiv;
                }
                else fibo=0;
            }
            return fibo;
        }
    }
}
int main(){
  int n,i,v;liste L;
  L=NULL;
  do{
      printf("Donner un nombre a inserer (taper un nombre negatif pour quiter):
");
      scanf("%d",&v);
      if(v>=0)L=insertTail(v,L);
  }while(v>=0);
  printf("La liste de depart:\n");
  afficherListe(L);
  if(Fibonacci(L)==1)printf("Cette liste represente la suite de Fibonacci");
  else printf("Cette liste ne represente pas la suite de Fibonacci");
  return 0;
```

```
}
```

Exercice 258:

- Ecrire la procédure **`TriParInsertion()`** qui prend en paramètres une liste chaînée d'entiers et qui la trie dans l'ordre croissant. La méthode à suivre dans cet exercice est le tri par insertion et pas par permutation (ni des éléments ni de leur valeur). Ainsi, l'élément contenant la plus petite valeur est enlevé de la liste et inséré de nouveau en première position. Ensuite l'élément avec la deuxième plus petite valeur est enlevé aussi et inséré après le premier, et ainsi de suite. Le processus se réitère Jusqu'à ce que tous les éléments de la liste sont enlevés et insérés de nouveau dans l'ordre.
- Ecrire la fonction principale **`main()`** qui permet de remplir une liste chaînée par une série de nombres saisis au clavier et se terminant par 0 (le 0 n'est pas compté), et qui trie cette liste en appelant la procédure **`TriParInsertion()`** précédente. Le programme doit afficher la liste avant et après le tri.

Réponse:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* liste;
typedef struct node{
               int val;
               liste suiv;
}node;
liste insertTail(liste l,int v){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
       q=l;
       while(q->suiv!=NULL)q=q->suiv;
       q->suiv=p;
    }
    return l;
}
void display(liste l){
    liste p;
    p=l;
    while(p!=NULL){
        printf("%d | ",p->val);
        p=p->suiv;
    }
    printf("\n");
}
void TriParInsertion(liste* l){
    liste p,q,min,prec,precMin,precP;int a;
    p=*l;
    int i=0;
    while(p!=NULL){
        min=p;
        prec=p;
        q=p->suiv;
```

```
        while(q!=NULL){
            if(q->val < min->val){
                precMin=prec;
                min=q;
            }
            prec=prec->suiv;
            q=q->suiv;
        }
        if(min!=p){
            precMin->suiv=min->suiv;
            min->suiv=p;
            if(p==*l){
                *l=min;
                precP=min;
            }
            else{
                precP->suiv=min;
                precP=min;
            }
        }
        else{
          precP=p;
          p=p->suiv;
        }
    }
}
int main(){
    int v;liste L,S;
    L=NULL;
    do{
        printf("Donner un nombre a inserer (taper 0 pour quiter): ");
        scanf("%d",&v);
        if(v!=0)L=insertTail(L,v);
    }while(v!=0);
    printf("La liste de depart L:\n");
    display(L);
    TriParInsertion(&L);
    printf("La liste triee:\n");
    display(L);
}
```

Exercice 259:

Soit la suite U définie par:

$$U_1 = 1$$

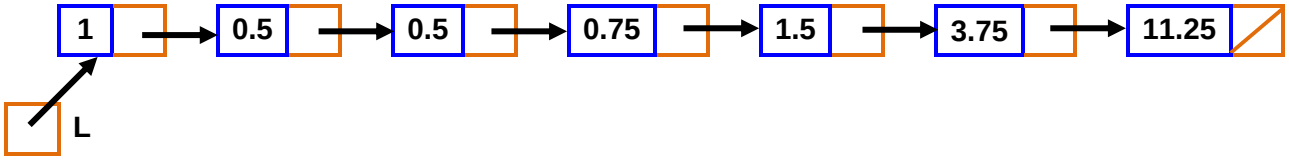
$$U_n = U_{n-1} \times \frac{n-1}{2}, \quad pour\ \mathrm{n} \geq 2$$

On voudrais établir un programme permettant de vérifier si une suite numérique représentée par une liste chaînée est notre suite U. Ainsi, il vous ai demandé de:

- Ecrire la fonction **`saisirSuite()`** qui permet de saisir au clavier les termes d'une suite numérique en les stockant dans une liste chaînée (la saisi se termine par l'introduction d'une valeur négative) et de renvoyer cette liste comme résultat.
- Ecrire la fonction **`estU()`** qui prend en entrée une liste chaînée représentant une suite numérique et qui renvoie True si cette suite est la suite U et renvoie False dans le cas contraire.

  Exemple:

  La liste chaînée suivante correspond à la suite U d'ordre 6:



- Ecrire la fonction principale **`main()`** qui n'a comme rôle que d'appeler les deux fonctions précédentes et d'afficher le résultat.

Remarques:

- Le programme pourra contenir d'autres modules auxiliaires comme par exemple une fonction d'insertion en queue, une procédure d'affichage, etc.
- Tous les sous-programmes doivent être récursifs; en d'autres termes le programme ne doit contenir aucune boucle.

Réponse:

```
#include <stdio.h>
typedef struct node* liste;
typedef struct node{
               float val;
               liste suiv;
}node;
liste insertTail(float v,liste L){
    liste p,q;
    if(L==NULL){
        p=malloc(sizeof(node));
        p->val=v;
        p->suiv=NULL;
        L=p;
    }
    else{
         L->suiv=insertTail(v,L->suiv);
    }
    return L;
}
liste saisirSuite(liste L){
```

```
    float v;
    printf("Donner un nombre a inserer (taper un nombre negatif pour quiter):
");
    scanf("%f",&v);
    if(v>=0){
        L=insertTail(v,L);
        return saisirSuite(L);
    }

    return L;
}
void afficherListe(liste L){
    if(L!=NULL){
        printf("%.2f ",L->val);
        afficherListe(L->suiv);
    }
}
int estU(liste L,liste q,int num){
    if(L==NULL){
        if(q==NULL)
            return 0;
        else return 1;
    }
    else{
        if(q==NULL){
            if(L->val==1)
                return estU(L->suiv,L,num+1);
            else return 0;
        }
        else{
            if(L->val==q->val*num/2)
                return estU(L->suiv,L,num+1);
            else return 0;
        }
    }
}
int main(){
  liste L;
  L=NULL;
  L=saisirSuite(L);
  printf("La liste de depart:\n");
  afficherListe(L);
  if(estU(L,NULL,0)==1)printf("\nCette liste correspond a la suite U");
  else printf("\nCette liste ne correspond pas a la suite U");
  return 0;
}
```

Exercice 260:

On définit une matrice creuse comme étant une matrice dont plus que la moitié des éléments sont nuls.

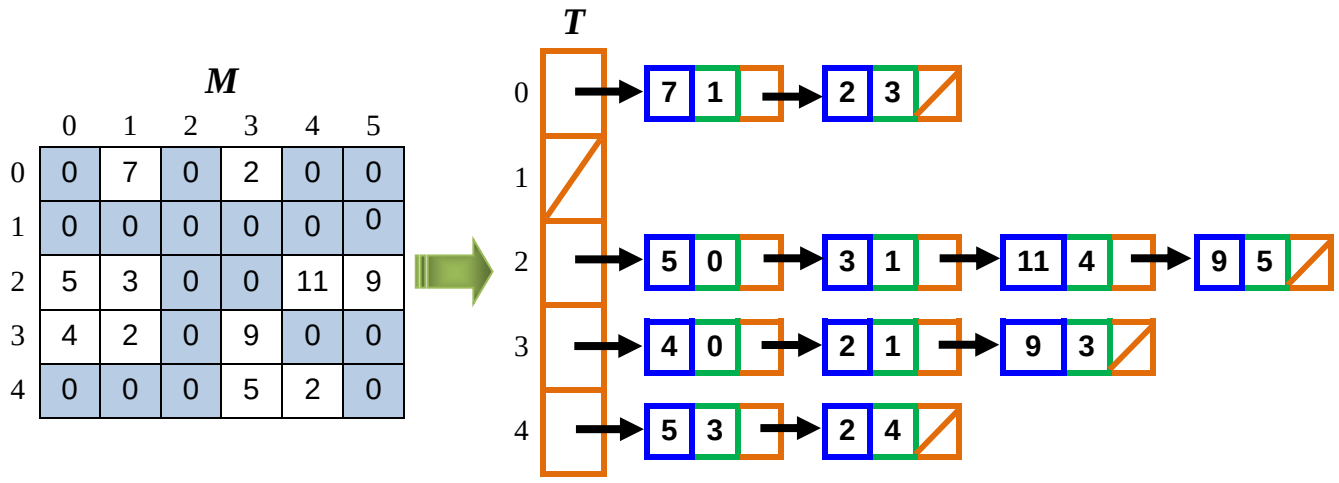
En effet, on peut représenter une matrice creuse en ne tenant compte que des éléments non nuls afin de minimiser l'espace occupé par ce type de matrice. Il est ainsi commun de représenter une matrice creuse sous forme d'un tableau de listes chaînées, de sorte que la $i^{ème}$ liste chaînée contient les éléments non nuls de la ligne i de la matrice.

Chaque élément d'une liste chaînée contient la valeur de l'élément et le numéro de la colonne où il se trouve.

1) Décrire la/les structure(s) de données modélisant ce problème.
2) Ecrire la fonction **`estCreuse()`** qui prend en entrée une matrice **`M`** de taille **`n × m`** et qui teste si **`M`** est creuse ou pas. La fonction retourne alors **`True`** si **`M`** est creuse et retourne **`False`** sinon.
3) Ecrire la fonction **`insertTail()`** qui permet d'insérer un élément composé du couple (**`valeur`**, **`numéro de colonne`**) dans une liste chaînée.
4) Ecrire la procédure **`transformer()`** qui permet de transformer une matrice creuse **`M`** de taille **`n × m`** en tableau de listes chaînées **`T`** comme défini précédemment.
5) Ecrire la procédure **`afficherMatrice()`** qui affiche une matrice dont les éléments sont stockés dans un tableau de listes chaînées **`T`** passé en paramètre.
6) Ecrire la fonction principale **`main()`** qui permet de saisir une matrice **`M`** de 5 lignes et de 6 colonnes, de la transformer en tableau de listes chaînées **`T`**, et d'afficher la matrice à partir du tableau **`T`**. La transformation n'est possible que If La matrice est creuse.
   Vous devez utiliser les sous-programmes précédents.

Exemple:

Matrice creuse **Tableau de listes chaînées correspondant**



Réponse:

```
#include <stdio.h>
#include <stdlib.h>
#define n 5
#define m 6
//déclaration de types
typedef int mat[n][m];
typedef struct node* liste ;
typedef struct node {
     int val;
     int col;
     liste suiv;
}node;
typedef liste tab[n];
//les sous-programmes
int estCreuse(mat M){
    int nbNuls=0,i,j;
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
```

```
        if(M[i][j]==0)nbNuls++;
    if(nbNuls>(n*m)/2)return 1;
    else return 0;
}
liste insertTail(int v,int col,liste l){
    liste p,q;
    p=malloc(sizeof(node));
    p->val=v;
    p->col=col;
    p->suiv=NULL;
    if(l==NULL)l=p;
    else{
        q=l;
        while(q->suiv!=NULL)q=q->suiv;
        q->suiv=p;
    }
    return l;
}
void transformer(mat M,tab t){
    int i,j;
    for(i=0;i<n;i++){
        t[i]=NULL;
        for(j=0;j<m;j++)
            if(M[i][j]!=0)t[i]=insertTail(M[i][j],j,t[i]);
    }
}
void afficheMatrice(tab t){
    liste p;int i,j;
    printf("Les elements de la matrice:\n");
    for(i=0;i<n;i++){
        p=t[i];
        for(j=0;j<m;j++){
            if(p!=NULL && p->col==j){
                    printf("%d ",p->val);
                    p=p->suiv;
            }
            else printf("0 ");
        }
        printf("\n");
    }
}
int main(){
  int i,j;mat M;
  printf("Veuillez remplir la matrice svp\n");
  for(i=0;i<n;i++)
  for(j=0;j<m;j++){
    printf("Element [%d,%d]: ",i,j);
    scanf("%d",&M[i][j]);
  }
  if(estCreuse(M)==0)printf("La matrice n'est pas creuse");
  else{
    tab T;
    transformer(M,T);
```

```
    afficheMatrice(T);
  }
  return 0;
}
```

Exercice 261: problème de Joséphus

En mathématiques et en informatique, le problème de Josèphus ou problème de Caligula est un problème d'élimination, conduisant à l'obtention d'un unique survivant. Il a été énoncé sous différentes formes, mais sa première formulation est due à Flavius Josèphus.

Josephus Flavius était un célèbre historien juif du premier siècle de notre ère. Durant une guerre il fut pris au piège dans une cave avec son groupe de 40 soldats, entouré par les troupes ennemies. La légende raconte que le groupe encerclé préféra se suicider plutôt que d'être capturé. Ainsi Josephus et ses soldats formèrent un cercle et décidèrent de se tuer mutuellement et successivement, de manière à ce qu'une personne tue la troisième personne sur sa gauche, que la personne à droite du mort tue à son tour la troisième personne sur sa gauche, ainsi de suite Jusqu'à ce qu'il ne reste qu'un seul survivant. Restant seul, ce dernier est censé se suicider lui-même. Josephus, qui ne souhaitait pas mourir, trouva rapidement la place sûre, c'est-à-dire la place de la dernière personne debout, sans que quiconque ne reste pour le tuer. AinIf il resta en vie et put par la suite raconter cette légende. Trouver cette place sûre est maintenant appelé le problème de Josephus.

Durant cet exercice nous implémenterons un programme qui simulera une version généralisée du problème de Josephus de la manière suivante : étant donnés **`n`** soldats, placés en cercle aux positions [1 ; **`n`**] avec 1 comme position de départ, il faut retirer chaque **`k`**^{ième} soldat à gauche Jusqu'à ce que tous les soldats sauf le dernier (le survivant) soient retirés.

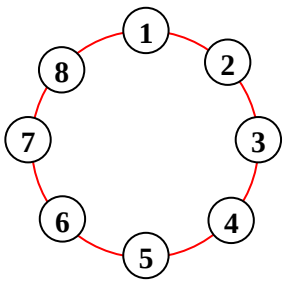
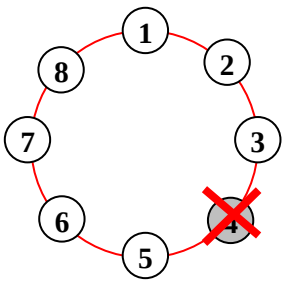
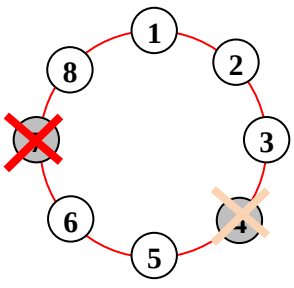
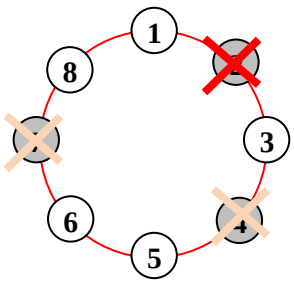
Pour mieux comprendre le processus, veuillez voir l'exemple suivant:

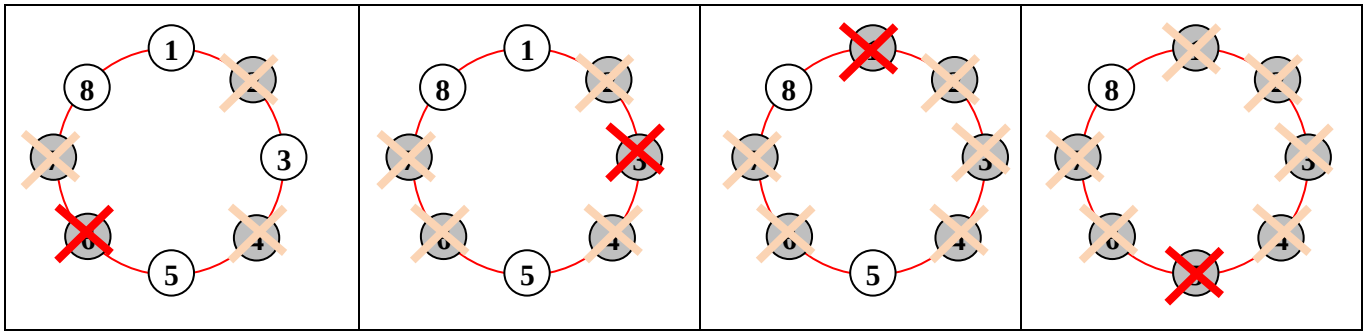
Exemple:

Afin de faciliter la simulation, nous considérons dans cet exemple **`n`** = 8 et **`k`** = 3.

Le soldat 1 tue le troisième à main gauche (qui est le soldat numéro 4). Le soldat placé à main droite du mort, en l'occurrence le 3, prend la relève et tue à son tour le soldat 7. Le voisin droit de ce dernier, qui est le soldat 6, continue le meurtre en tuant le troisième à sa gauche: le numéro 2. Et ainsi de suite.

On considère les morts comme retirés du cercle. On ne compte donc plus que les vivants. Et de fil en aiguille, on arrive à un seul survivant, ici le soldat 8.

Itération 1	Itération 2	Itération 3	Itération 4
			
Itération 5	**Itération 6**	**Itération 7**	**Itération 8**



Le programme que vous devez développer devra prendre comme entrées les nombres **n** et **k**, respectivement le nombre de soldats et la distance (dans l'exemple ci dessus: **n** = 8 et **k** = 3), et produire comme sortie l'ordre dans lequel les soldats seront tués, le soldat restant étant finalement le survivant.

Ainsi, dans l'exemple l'ordre de meurtre est le suivant: 4, 7, 2, 6, 3, 1, 5. Donc le survivant est le soldat numéro 8.

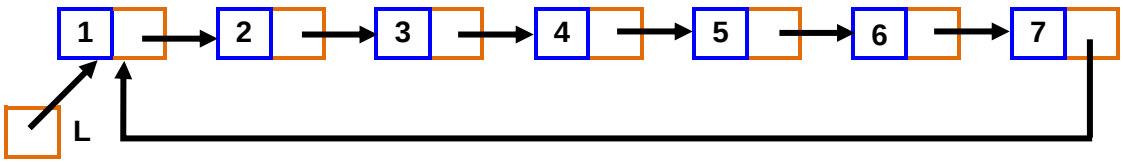
Considérant que la cercle de soldats peut être modélisé par une liste chaînée circulaire, et que chaque soldat est définie par un numéro et un nom, Il vous ai demandé de:

1) Donner la structure de données modélisant le cercle de soldats.
2) Ecrire la fonction **`insererSoldat()`** permettant d'insérer un soldat dans la liste correspondante au cercle de soldats. La liste, le numéro et le nom du soldat sont passées en paramètres.
3) Ecrire la procédure **`afficherSoldats()`** qui prend en entrée un cercle de soldats et qui affiche tous les soldats dans le cercle.
4) Ecrire la fonction **`tuerSoldat()`** qui permet de tuer un soldat (de le retirer du cercle). Cette fonction prend entrée un pointeur sur le soldat qui précède le soldat qui va être tué, et renvoie le cercle après le changement.
5) Ecrire la fonction **`Josephus()`** qui implémente le processus de Joséphus sur une liste de candidats passée en paramètre. Cette fonction prend en entrée le cercle de soldats et la distance **k**, tue et affiche le soldat tué à chaque itération, et renvoie finalement le cercle qui ne contient que le soldat survivant.
6) Ecrire la fonction principale **`main()`** qui permet de saisir au clavier le nombre de soldats **n** et la distance **k**, lit les informations des **n** soldats tout en les insérant dans la liste représentant le cercle de soldats, et qui applique finalement le processus de Josephus et cherche le soldat survivant en appelant la fonction **`Josephus()`**.

   Le programme doit afficher la liste de soldats avant et après l'application du processus de Josephus.

Notice:

La liste chaînée circulaire est une liste chaînée où tous les nœuds sont connectés pour former un cercle. Le suivant du dernier nœud n'est pas alors NULL mais le premier nœud. La figure suivante illustre un exemple:



Dans une liste chaînée circulaire, tout nœud peut être un point de départ. Nous pouvons parcourir toute la liste en partant de n'importe quel point. Il suffit de s'arrêter lorsque le premier nœud visité est à nouveau visité.

Réponse:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct soldat *cercle;
typedef struct soldat{
    int num;
    char nom[30];
    cercle suiv;
}soldat;
cercle insererSoldat(cercle C,int num,char nom[30]){
    cercle p,q;int i;
    p=malloc(sizeof(soldat));
    p->num=num;
    i=0;
    while(nom[i]!='\0'){
        p->nom[i]=nom[i];
        i++;
    }
    p->nom[i]='\0';
    if(C==NULL){
            C=p;
            p->suiv=C;
    }
    else{
        q=C;
        while(q->suiv!=C){
            q=q->suiv;
        }
        q->suiv=p;
        p->suiv=C;
    }
    return C;
}
void afficherSoldats(cercle C,char* message){
    cercle p;
    printf("\n%s:\n",message);
    p=C;
    do{
       printf("\t(%d, %s)\n",p->num,p->nom);
       p=p->suiv;
    }while(p!=C);
}
cercle tuerSoldat(cercle C,cercle q){
    cercle p;
    p=q->suiv;
    printf("\tLe soldat: (%d, %s) est tue\n",p->num,p->nom);
    q->suiv=p->suiv;
    if(p==C)C=C->suiv;
    free(p);
    return C;
}
cercle Josephus(cercle C,int k){
```

```
    cercle p,q;int i,nb;
    p=C;
    printf("L'ordre de meurtre:\n");
    while(p->suiv!=p){
        i=0;
        while(i<k){
            q=p;
            p=p->suiv;
            i++;
        }
        C=tuerSoldat(C,q);
        p=q;
    }
    return C;
}
int main(){
    cercle C;int n,k,i,num;char nom[30];char poub;
    C=NULL;
    printf("Donner le nombre de soldats: ");
    scanf("%d",&n);
    printf("Donner la distance k: ");
    scanf("%d",&k);
    for(i=1;i<=n;i++){
        printf("Donner les informations du soldat %d\n",i);
        printf("\tNum: ");scanf("%d",&num);
        scanf("%c",&poub);
        printf("\tNom: ");
        gets(nom);
        C=insererSoldat(C,num,nom);
    }
    afficherSoldats(C,"Les soldats avant le meurtre");
    C=Josephus(C,k);
    afficherSoldats(C,"Les soldats apres le meurtre");
    printf("le survivant est le soldat: (%d, %s)",C->num,C->nom);
}
```

Exercice 262 (devoir 2019):

1) Description et travail demandé:

Une matrice binaire est une matrice contenant que des 0 et des 1.

Soit **M** une matrice binaire de **n** lignes et de **m** colonnes. On appelle **Objet** un ensemble de 1 voisins. Par exemple la matrice suivante comporte 4 objets, représenté chacun par une couleur.

	0	1	2	3	4
0	1	1	0	1	0
1	1	0	0	0	1
2	0	0	0	0	0
3	1	1	0	1	1
4	1	1	0	0	1

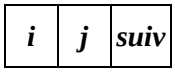
- Le premier objet comporte les cases de coordonnées **(0,0) (0,1) (1,0)**
- Le deuxième objet comporte les cases de coordonnées **(0,3) (1,4)**
- Le troisième objet comporte les cases de coordonnées **(3,0) (3,1) (4,0) (4,1)**
- Le quatrième objet comporte les cases de coordonnées **(3,3) (3,4) (4,4)**

Ce qui est demandé est de concevoir et d'implémenter un programme en langage C permettant de remplir aléatoirement une matrice binaire, de l'afficher, et d'en extraire les différents objets existants tout en stockant les coordonnées des cases composant chaque objet dans une liste chainée. Le programme doit finalement afficher les différents objets extraits.

2) Démarche à suivre:

Les étapes à suivre pour résoudre le problème sont les suivants :

- Déclarer 2 constantes globales nommées **n** et **m** représentant respectivement le nombre de lignes et le nombre de colonnes. Pour vos tests, prenez des petites valeurs (par exemple **n** = 4 et **m** = 5).
- Donner la déclaration des types globales **liste** et **element** exprimant la structure qui représente un élément de la liste chaînée. Chaque élément est un enregistrement composé de 3 cases: deux entiers **i** et **j** décrivant respectivement le numéro de ligne et le numéro de colonne de chaque élément, et une case **suiv** qui s'agit d'un pointeur sur **element** (type **liste**). Il a donc la forme suivante:



- Ecrire la procédure **void remplirMatrice(int M[n][m])** qui permet de remplir une matrice binaire passée en paramètre aléatoirement.

  Remarque:

  pour générer un nombre aléatoirement en langage C, on utilise la fonction prédéfinie **rand()** du fichier d'entête **<windows.h>**. Cette utilisation doit être précédée par l'initialisation de **rand()** à l'aide de l'instruction **srand(time(NULL))**.

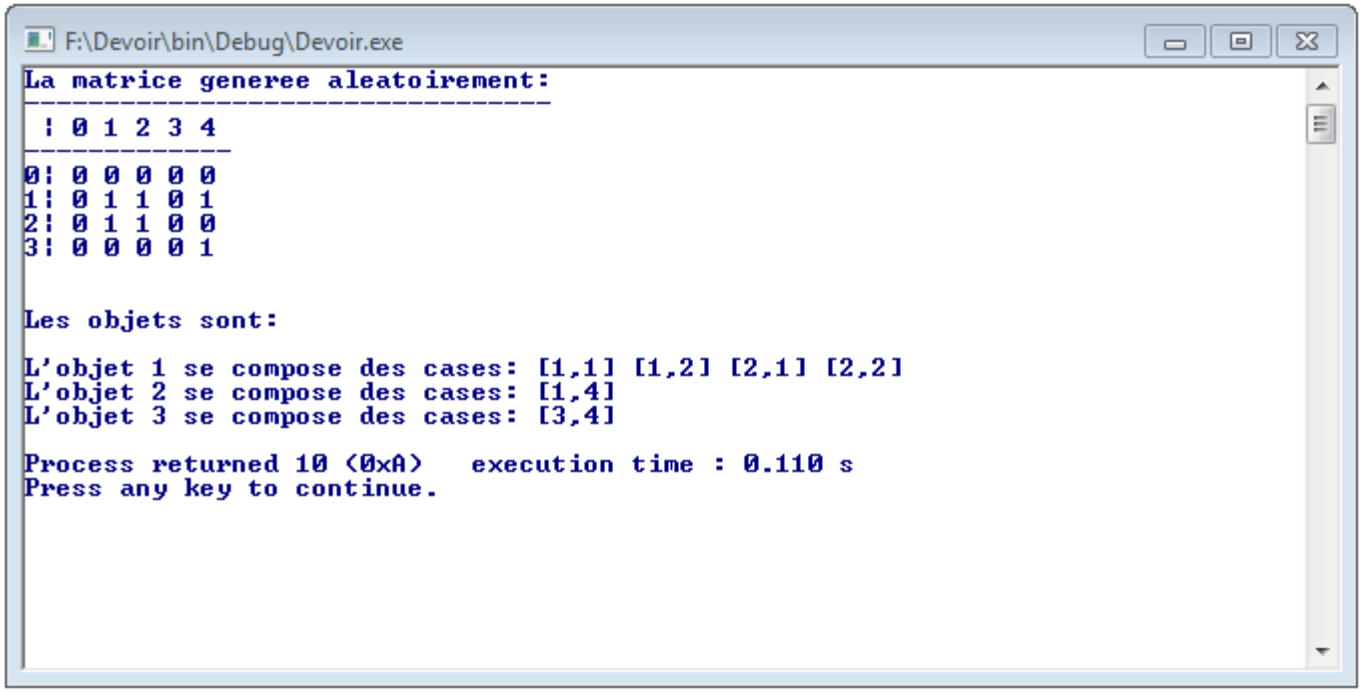
  Par exemple, les deux instructions suivantes permettent de générer aléatoirement un nombre binaire (0 ou 1) et de le stocker dans la variable **nb**:

```
srand(time(NULL));
int nb = rand() % 2;
```

- Ecrire la procédure **void afficherMatrice(int M[n][m])** qui affiche une matrice d'entiers passée comme paramètre.
- Ecrire la fonction **liste insertTail(liste tete,int i,int j)** qui insère un élément, composé de paire **(i,j)**, dans une liste chaînée dont la tête est passée comme paramètre.
- Ecrire la fonction récursive **liste extraireObjet(int M[n][m],int i,int j, liste tete)** qui, à partir d'une matrice binaire et des coordonnées **(i,j)** d'une case contenant 1, renvoie une liste chaînée composée des coordonnées de toutes les cases qui forment un objet incluant la case **[i,j]**. L'insertion dans la liste chaînée doit se faire à l'aide de la fonction **insertTail()**.
- Ecrire la procédure **void afficherObjet(liste tete)** qui affiche les coordonnées des cases formant un objet représenté par une liste chainée, dont la tête est passée en paramètre.
- Ecrire la procédure **void extraireTousLesObjets(int M[n][m])** qui permet d'extraire tous les objets contenus dans une matrice binaire passée en paramètre et de les afficher. Cette fonction doit utiliser les deux sous-programmes **extraireObjet()** et **afficherObjet()** précédents.
- Ecrire le corps de la fonction principale **main()** qui permet de générer une matrice binaire aléatoirement, de l'afficher, et puis d'extraire et afficher tous ses objets.
  - La génération d'une matrice aléatoirement doit se faire à l'aide de la procédure **remplirMatrice()**.
  - L'affichage de la matrice se fait à l'aide de la procédure **afficherMatrice()**.
  - L'extraction et l'affichage des objets doit être effectués en appelant la procédure **extraireTousLesObjets()**.

3) Exemple d'exécution:

Un exemple du résultat attendu du programme demandé est illustré par la figure suivante :



Réponse:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#define n 4
#define m 5
typedef struct node* liste;
typedef struct node{
    int i,j;
    liste suiv;
}node;
void remplirMatrice(int M[n][m]){
    int i,j;
     srand(time(NULL));
    for(i=0;i<n;i++)
        for(j=0;j<m;j++){
            M[i][j]=rand()%2;
        }
}
void afficherMatrice(int M[n][m]){
    printf("La matrice generee aleatoirement:\n");
    printf("---------------------------------\n | ");
    int i,j;
    for(j=0;j<m;j++)printf("%d ",j);
    printf("\n-------------\n");
    for(i=0;i<n;i++){
        printf("%d| ",i);
        for(j=0;j<m;j++)
            printf("%d ",M[i][j]);
        printf("\n");
    }
}
liste insertTail(liste tete,int i,int j){
    liste p=malloc(sizeof(node));
```

```
    p->i=i;
    p->j=j;
    if(tete==NULL)
        tete=p;
    else{
        liste q=tete;
        while(q->suiv!=NULL)    q=q->suiv;
        q->suiv=p;
    }
    p->suiv=NULL;
    return tete;
}
liste extraireObjet(int M[n][m],int i,int j,liste tete){
    tete=insertTail(tete,i,j);
    M[i][j]=0;
    if(i>0 && j>0 && M[i-1][j-1]==1)tete=extraireObjet(M,i-1,j-1,tete);
    if(i>0 && M[i-1][j]==1)tete=extraireObjet(M,i-1,j,tete);
    if(i>0 && j<m-1 && M[i-1][j+1]==1)tete=extraireObjet(M,i-1,j+1,tete);
    if(j>0 && M[i][j-1]==1)tete=extraireObjet(M,i,j-1,tete);
    if(j<m-1 && M[i][j+1]==1)tete=extraireObjet(M,i,j+1,tete);
    if(i<n-1 && j>0 && M[i+1][j-1]==1)tete=extraireObjet(M,i+1,j-1,tete);
    if(i<n-1 && M[i+1][j]==1)tete=extraireObjet(M,i+1,j,tete);
    if(i<n-1 && j<m-1 && M[i+1][j+1]==1)tete=extraireObjet(M,i+1,j+1,tete);
return tete;
}
void afficherObjet(liste tete){
    while(tete!=NULL){
        printf("[%d,%d] ",tete->i,tete->j);
        tete=tete->suiv;
    }
   printf("\n");
}
void extraireTousLesObjets(int M[n][m]){
    printf("\n\nLes objets sont:\n\n");
    int i,j,nb;
    nb=1;
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
            if(M[i][j]==1){
                liste tete=NULL;
                tete=extraireObjet(M,i,j,tete);
                printf("L'objet %d se compose des cases: ",nb);
                afficherObjet(tete);
                nb++;
            }
}
main(){
    system("color F1");
    int M[n][m];
    remplirMatrice(M);
    afficherMatrice(M);
    extraireTousLesObjets(M);
}
```