PROBLÈME I: REPRÉSENTATION DE GRANDS NOMBRES ENTIERS NATURELS

Contexte du problème :

Pour tout langage de programmation, la représentation des nombres par des types prédéfinis est très limitée (en langage **C**, une variable entière v de type **long int** est bornée ainsi : **-2147483648<=v<=2147483647**).

Cependant, plusieurs problèmes (par exemple les systèmes CPS, la cryptographie ou encore les applications de gestion) ont besoin d'utiliser des nombres ayant des valeurs et des précisions qui dépassent largement les limites des types prédéfinis par les langages de programmation.

Pour toutes ces applications, les types de base ne conviennent plus et il faudra définir de nouvelles représentations pour ces nombres.

Le problème suivant s'inscrit dans ce contexte. Il s'intéresse à la représentation des nombres entiers positifs pouvant avoir des valeurs très grandes non spécifiées par les types standard su langage C.

A : Représentation par des chaines de caractères

Dans cette partie, on représentera un nombre positif ou nul par une chaine de caractères contenant ses chiffres.

Rappels

– Une chaine de caractères en **langage C** est un tableau de caractères se terminant par le caractère spécial **'\0'**.
– La longueur d'une chaine de caractères est le nombre de caractères avant **'\0'**.

Notations

– On dira qu'une chaine de caractères S est une **ChaineChiffres** si sa longueur est strictement positive et tout caractère de S est un caractère chiffre (les caractères chiffres sont '0','1','2','3','4','5','6','7','8,'9').

– Soit **N** un nombre entier positif ou nul de **NC** chiffres (NC > 0), la valeur décimale (base 10) de N est: $N=C_{NC-1}C_{NC-2}...C_i...C_1C_0$, ($C_0$ chiffre des unités, C_1 chiffre des dizaines,...), on dit que N est représenté par une **chaineChiffre** S, si S contient les chiffres de N comme suit : S= "$C_{NC-1}C_{NC-2}...C_i...C_1C_0$"

Exemple :

Le nombre N=45009876156430987 de 17 chiffres sera représenté par la **ChaineChiffres** S= "45009876156430987" (S [0]='4 ', S[1]= '5 ', S[2]= '0 ' ,...S[16]= '7 ')

❖ **Question 1 : Chaine de chiffres**

Soit S une chaine de caractères quelconque déjà déclarée et initialisée.

✍ Définir une fonction d'entête **int ChaineChiffres()** qui retourne 1 si S est une **ChaineChiffres** ou 0 (zéro) sinon.

Exemple :

– Si S= **"78500120360007"** alors l'appel à la fonction **ChaineChiffres()** retourne 1.
– Si S= "**856942a1478"** alors l'appel de la fonction **ChaineChiffres()** retourne 0.

❖ **Question 2 : Zéros non significatifs**

On suppose que S est une **ChaineChiffres** déjà déclarée et définie.

✍ Ecrire une fonction d'entête **void supprimer_zeros()** qui supprime les zéros à gauche de la chaine S (les zéros non significatifs dans un nombre)

Exemple :
Si S= "0009760004300", après l'appel de **supprimer_zeros( ), S= "9760004300".**

❖ **Question 3 : Somme de deux chaines de chiffres.**

Il s'agit de faire la somme de deux nombres entiers positifs représentés par leurs **ChaineChiffres**.

Pour ce faire:

- Écrire une fonction de prototype **void additionner(char S1[ ], char S2[ ],char SOM[ ]);** qui place dans la chaine **SOM** la somme des nombres représentés par les **ChaineChiffres S1** et **S2**.

Rappel :

Les valeurs décimales des codes **ASCII** des caractères chiffres sont indiquées dans le tableau qui suit :

Caractère	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
Code ASCII	48	49	50	51	52	53	54	55	56	57

Exemple :

si S1= "129782004977" et S2= "754022234930" alors après l'appel de la fonction **additionner(S1,S2,SOM),** on aura SOM= "883804239907".

Problème II : Distance de Hamming

La distance de Hamming, définie par Richard Hamming permet de quantifier la différence entre deux séquences de symboles. Elle est utilisée en informatique et en télécommunications pour compter le nombre de bits altérés dans la transmission d'un message d'une longueur donnée.

Dans ce problème, on se propose d'implémenter des fonctions pour calculer la distance de **Hamming.**

Question 1 : Distance de Hamming entre deux chaînes de caractères

La distance de **Hamming** entre deux chaînes de caractères de même longueurs est égale au nombre de caractères, à la même position, qui sont différents.

Exemples :

La distance de Hamming entre "sure " et "cure" est 1,

la distance de Hamming entre "aabbcc" et "xaybzc" est 3.

Écrire une fonction d'entête : **int distanceH(char S1[ ], char S2[ ], int M)** qui calcule et retourne la distance de Hamming entre S1 et S2 (Les paramètres S1 et S2 sont deux chaînes de caractères de même longueur M et on suppose que le paramètre M est strictement positif)

Question 2 : Distance de Hamming d'un langage

On appellera **langage,** un tableau de chaînes de caractères toutes de memes longueurs. La distance de Hamming d'un langage est égale au minimum des distance de Hamming entre deux chaînes de ce langage différentes deux à deux.

Exemple :

Si langage={"aabb","xayy","tghy","xgyy"}, sa distance de Hamming est de 1

Ecrire une fonction d'entête : **int distanceH_langage(char langage[NB][L])**, qui retourne la distance de Hamming de son paramètre langage (le paramètre langage est un tableau de NB chaînes de caractères toutes de même longueur L, NB et L sont 2 constantes entières strictement positives déjà définies)

Question 3 : Distance de Hamming entre 2 nombres entiers positifs

La distance de Hamming entre 2 nombres entiers positifs est le nombre de bits distincts dans leurs représentations binaires (voir exemple)

Exemple : la distance de Hamming entre les nombres 7 et 4 est 2 (7 est représenté en binaire sur un octet (8bits) par 00000111 et 4 est représenté en binaire par 00000100)

Question 3-a:

Écrire une fonction d'entête : **void binaire(char bin [] , int N)** qui met dans la chaîne d'adresse bin, la représentation binaire de N (On suppose que 0<=N<256)

Question 3-b:

Écrire une fonction d'entête : **int distanceNombre(int A , int B)** qui calcule et retourne la distance de Hamming entre les nombres A et B (On suppose que 0<=A<256 et 0<=B<256)

Problème III : Fusion de deux tableaux triés

Ecrire le code de la fonction void fusioner(int T1[N1] , int T2[N2] , int T3[N1+N2]) qui permet de fusionner les deux tableaux T1 et T2 de telle sorte à obtenir un tableau T3 trié mais qui ne contient pas des doublons. Les dernières cases de T3 seront égales à -1.

Exemple :

T1 =

6	14	33	51	55	59	63	81	88	89

T2 =

4	19	51	56	63	200

T3 =

4	6	14	19	33	51	55	56	59	63	81	88	89	200	-1	-1

Corrigé

PROBLÈME I: REPRÉSENTATION DE GRANDS NOMBRES ENTIERS NATURELS

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
char* S;
int ChaineChiffres()
{int i;
if (strlen(S)==0) return(0);
for(i=0;i<strlen(S);i++)
if(S[i]<'0' && S[i]>'9')return(0);
return(1);
}

void supprimer_zeros()
{int i;
while(S[0]=='0')
for(i=0;i<strlen(S);i++)S[i]=S[i+1];  }

void add(char a, char b, char c, char* r, char* s)
{*s='0'+ (a+b+c-3*'0')%10;
*r='0'+(a+b+c-3*'0'>=10);}

void CompleterXPar0G(char X[], char Y[])
{char*Z=(char*)malloc(strlen(Y)+1);
 strcpy(Z,""); int i;
 for(i=0;i<strlen(Y)-strlen(X);i++)
   {strcat(Z,"0");}
 strcat(Z,X);
 strcpy(X,Z); }
```

```
void additionner(char S1[], char S2[], char S3[])
{char r,s;
int max;
if(strlen(S1)>strlen(S2))
   {CompleterXPar0G(S2,S1); max=strlen(S1);}
  else {CompleterXPar0G(S1,S2); max=strlen(S2);}
char*Z=(char*)malloc(max+2);
int i=strlen(S1)-1, k=max+1;
Z[k]='\0';  r='0'; k--;
while(i>=0)
{add(S1[i], S2[i], r, &r, &s);
 Z[k]=s; k--;i--; }
 Z[0]=r;
strcpy(S3,Z);
}
main()
{S=malloc(10);
char S1[10]="6666", S2[20]="8888",S3[40];
additionner(S1,S2,S3);
puts(S1);puts(S2);puts(S3);   getch();  }
```

Problème II : Distance de Hamming

```
#include<stdio.h>
#include<conio.h>
#define L 20
#define NB 100
int distanceH(char S1[], char S2[], int M)
{int s=0,i;
 for(i=0;i<M;i++)if(S1[i]!=S2[i])s++;
 return(s);}
```

```
int distanceH_langage(char langage[NB][L])
{int d=L; int i,j;
for(i=0;i<NB;i++)
for(j=i+1;j<NB;j++)
if(distanceH(langage[i],langage[j],L)<d)
  d=distanceH(langage[i],langage[j],L);
}

void binaire(char *bin ,int N)
{int i=0;
for(i=0;i<8;i++) bin[i]='0';          bin[7]='\0';      i=0;
while(N!=0)
{if((N%2)==0)bin[7-i]='0'; else bin[7-i]='1';
 N=N/2;    i++;
}}

int distanceNombre(int a, int b)
{char ch1[8],ch2[8];    binaire(ch1,a);binaire(ch2,b);
return(distanceH(ch1,ch2,L));  }
```

Problème III : Fusion de deux tableaux triés

```
#define N1 5
#define N2 5
int T[N1+N2];
void FusionerT1etT2dansT(int T1[N1],int T2[N2])
{int i=0,j=0,k=0;   for(i=0;i<N1+N2;i++) T[i]=0;   i=0;
while(i<N1&&j<N2)
{  if(T1[i]<T2[j]){T[k]=T1[i];k++;i++;}
   if(T2[j]<T1[i]){T[k]=T2[j];k++;j++;}
   if(T2[j]==T1[i]){T[k]=T2[j];k++;j++;i++;}
}
while(i<N1)  {T[k]=T1[i];k++;i++;}
while(j<N2)  {T[k]=T2[j];k++;j++;}
}
```

Série- N°2 : Enregistrements et Listes chaînées

Exercice 1:

Dans cet exercice on va gérer les nombres complexes qui sont des couples de réels, une partie réelle et une partie imaginaire.

Déclarer le TYPE complexe.

1) Ecrire une procédure qui permet de lire un nombre complexe c. La variable c doit être passée par adresse pour que la procédure puisse affecter les valeurs lues à c.
2) Ecrire une fonction qui retourne le produit de deux complexes.
3) Ecrire une fonction qui retourne le module d'un nombre complexe.
4) Ecrire une procédure permettant l'affichage des nombres complexes sous la forme "ai+b".

Exercice 2 :

La structure Client contient les informations d'un client d'une banque (nom , numéro de compte et solde). Le nom peut contenir au maximum 31 caractères, le compte (numéro de compte) contient 24 caractères.

struct date {int jour; int mois; int annee;}

struct Client { char nom[32]; char compte[25];

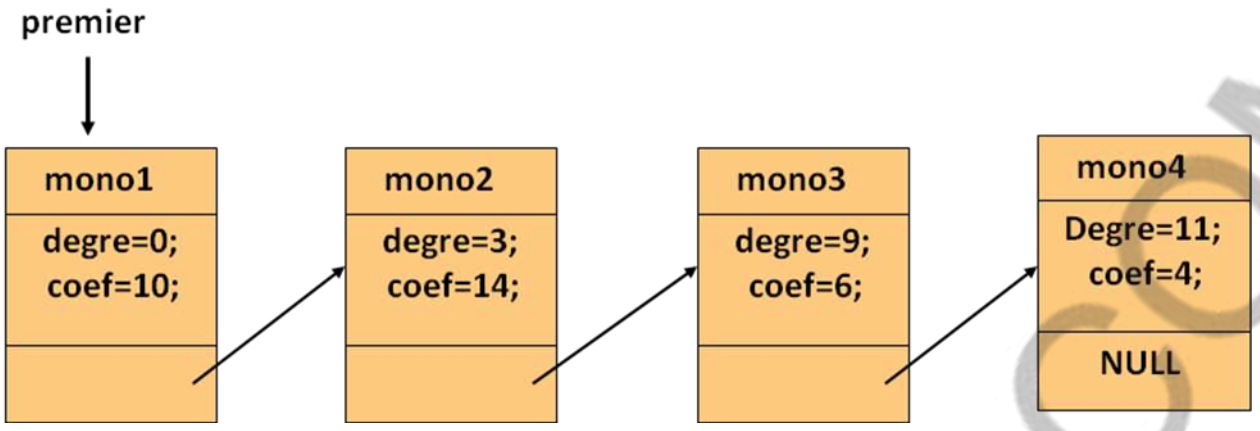
float solde; struct date D ; };

typedef struct Client sc ;

1) pourquoi on réserve toujours un caractère de plus dans une chaine de caractères (25 caractères pour le compte au lieu de 24).
2) Quand est ce qu'une fonction reçoit un pointeur sur une variable (paramètre de la fonction)?
3) Ecrire le corps de la fonction : void afficher_client(sc p) qui affiche les informations contenues dans la variable p de type struct client passée en argument.
4) Ecrire le corps de la fonction : sc Jeune_client(sc A, sc B) qui retourne le plus jeune des deux clients passés en argument.
5) Ecrire le corps de la fonction : void Mise_Ajour_client(sc* A, float nouveauSole) qui affecte au client *A le nouveau solde passé en argument.

Exercice 3 : Listes simplement chaînées

Soit L un polynôme à une seule variable, qu'on peut le représenter par une liste chaînée dont les nœuds sont des monômes, (comme ci-dessous: $P(X)=10+ 14*X^3 + 6*X^9 + 4*X^{11}$).



Soient les déclarations suivantes :

struct donnees_monome{int degre; float coef;};

struct noeud_monome {struct donnees_monome D; struct noeud_monome *suivant;};

typedef struct donnees_monome DM;

typedef struct noeud_monome* mon;

1) **Ecrire une fonction qui permet d'évaluer un polynôme en un point x, par exemple P(0)=10;**
2) **Ecrire une fonction qui permet de calculer fog(x)=f(g(x)), f et g sont deux polynômes, x est un réel donné.**
3) **Ecrire une fonction qui permet de multiplier un polynôme par un scalaire λ (multiplication de la liste par λ)**
4) **Ecrire une fonction qui permet de copier un polynôme A. Elle retourne une copie de A.**
5) **Ecrire une fonction qui permet de simplifier un polynôme A en remplaçant tous les monômes ayant des degrés égaux par un seul monôme qui est leur somme.**
6) **Ecrire une fonction qui permet de trier un polynôme A par ordre croissant des degrés de monômes (on suppose que A est simplifiée).**
7) **Ecrire une fonction qui permet de renvoyer une liste DeriveDuPoly qui est le polynôme dérivé du polynôme A.**
8) **Ecrire une fonction qui permet de renvoyer une liste DeriveDuPolyOrdreN qui est le polynôme dérivé d'ordre n du polynôme A.**
9) **Ecrire une fonction qui permet d'additionner deux polynôme A et B.**
10) **Ecrire une fonction qui permet de multiplier un polynôme A par un autre polynôme B.**

Corrigé du Série- N°2 : Enregistrements et Listes chaînées

Exercice 1:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
struct complexe { float re;
                  float im;};

typedef  struct  complexe  Comp;

void lire_complexe(Comp* c)
{float a,b;
 printf("\n entrer la partie reelle: ");
 scanf("%f",&(c->re));
 printf("\n entrer la partie immaginaire: ");
 scanf("%f",&(c->im));
 }

void somme_complexe(Comp c1, Comp c2, Comp* c3)
{ (*c3).re=c1.re+c2.re;
  (*c3).im=c1.im+c2.im;
}

Comp produit_complexe(Comp c1, Comp c2)
{ Comp c3;
c3.re=c1.re*c2.re-c1.im*c2.im;
c3.im=c1.re*c2.im+c1.im*c2.re;
return(c3);
}

float module(Comp c)
{ return(sqrt(pow(c.re,2)+pow(c.im,2)));}
```

```
void afficher_complexe(Comp c1)
{ printf("\n %2.2f + i.%2.2f",c1.re,c1.im);
}
 main()
{Comp c1,c2,c3;
lire_complexe(&c1);
lire_complexe(&c2);
c3=produit_complexe(c1,c2);
afficher_complexe(c3);
printf("\n Le module de c3 est: %2.2f "
,module(c3));
getch();}
```

Exercice 2 :

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
struct  date {int jour; int moi; int annee;};

struct Client {char nom[32];     char compte[25];    float solde;     struct date D ; };
typedef struct Client sc  ;
void afficher_Client(sc p)
{printf("\n le nom est: %s ",p.nom);
 printf("\n le telephone est: %s ",p.compte);
 printf("\n le solde est: %f ",p.solde);
 printf("\n Ne(e), le: %d/%d/%d ",   p.D.jour,p.D.moi,p.D.annee); }

sc Jeune_client(sc A, sc B)
{int C1=A.D.annee>B.D.annee;
int C2=(A.D.annee==B.D.annee && A.D.moi>=B.D.moi);
int C3=(A.D.annee==B.D.annee && A.D.moi==B.D.moi);
   C3=C3&&(A.D.jour>=B.D.jour);
if(C1||C2||C3) return(A);
else return(B);}
```

```
void Mise_Ajour_client(sc* A, float nouveauSole)
{A->solde=nouveauSole;}
main()
{sc A={"Mehdi","123456",100000,{4,6,1970}};
 sc B={"Alaoui","656429",50000,{4,12,1970}};
 sc C=Jeune_client(A,B);
 afficher_Client(C);
 Mise_Ajour_client(&A,3300);
 afficher_Client(A);
getch();
}
```

Exercice 3 : Listes simplement chaînées

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
struct donnees_etudiant{char nom[20];    char CIN[20];    int Annee_Naissance;    float MG;};
struct noeud_etudiant
{struct donnees_etudiant D;  struct noeud_etudiant *suivant;};

struct donnees_monome{int degre;  float coef;};
struct noeud_monome
{struct donnees_monome D;
  struct noeud_monome *suivant;};

typedef struct donnees_monome DM;
typedef struct noeud_monome* mon;

mon creer_LV()
{ mon TeteDeListe=NULL;
   return(TeteDeListe);
}

mon ajouterEnTete(mon Tete, DM d)
{    int taille=sizeof(struct noeud_monome);
    mon nouvelleMonome =(mon) malloc(taille);
```

```
nouvelleMonome->D=d;
nouvelleMonome->suivant = Tete;
return nouvelleMonome; }

void afficherDonnees(struct donnees_monome D)
{printf("%.2f*X^%d + ",D.coef, D.degre);}

void afficherListeMonomes(mon Tete)
{   printf("\n\n affichage de la liste :\n");
    mon P=Tete;
    while(P!=NULL)
    {afficherDonnees(P->D);
     P=P->suivant;
    }
}

mon ajouterEnFin(mon TeteActuelle, DM d)
{ int taille=sizeof(struct noeud_monome);
mon nouvelleMonome=(mon)malloc(taille);
nouvelleMonome->D=d;
nouvelleMonome->suivant = NULL;
 if(TeteActuelle == NULL)
    {return nouvelleMonome;}
    else
    { mon temp=TeteActuelle;
    while(temp->suivant != NULL)
     {temp = temp->suivant;}
    temp->suivant = nouvelleMonome;
    return TeteActuelle;
    } }
mon supprimerMonomeEnTete(mon TeteActuelle)
{   if(TeteActuelle == NULL)    return NULL;
    else
    { mon NouvTete = TeteActuelle->suivant;
     free(TeteActuelle);
    return NouvTete;
    } }
```

```
mon supprimerMonomeEnFin(mon TeteActuelle)
{if(TeteActuelle == NULL)       return NULL;
if(TeteActuelle->suivant == NULL)
   { free(TeteActuelle);    return NULL;    }
mon tmp1 = TeteActuelle ;  mon tmp2;
while(tmp1->suivant != NULL)
   {tmp2 = tmp1;  tmp1 = tmp1->suivant;    }
   tmp2->suivant = NULL;
   free(tmp1);
   return TeteActuelle;
}

mon supprimerMonomeEnFin2(mon TeteActuelle)
{if(TeteActuelle == NULL) return NULL;
 if(TeteActuelle->suivant == NULL)
   {free(TeteActuelle); return NULL;}
else{
   mon tmp = TeteActuelle;
   while(tmp->suivant->suivant != NULL)
   {tmp = tmp->suivant;}
   free(tmp->suivant);    tmp->suivant=NULL;    return TeteActuelle;}
}

int lengthList(mon Tete1)
{int n=0;   mon P=Tete1;
while(P!=NULL)
{n++;   P=P->suivant; }
return(n);}

float evalPol(mon Tete, float x)
{float y=0;
mon P=Tete;
while(P!=NULL)
{y=y+P->D.coef*pow(x,P->D.degre);
P=P->suivant;
}
return(y);}
```

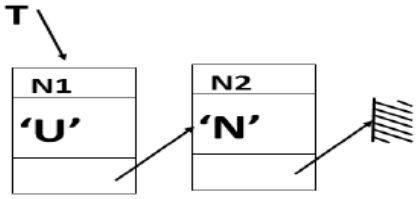
```
float evalPol_f_o_g(mon f, mon g, float x)
{float y=evalPol(g,evalPol(f,x));
return(y);}

void multScalPol(mon Tete, float lambda)
{mon P=Tete;
while(P!=NULL)
{P->D.coef=lambda*P->D.coef;
P=P->suivant;
}}

mon copyPol(mon Tete1)
{mon Tete2=NULL;
 mon P=Tete1;
while(P!=NULL)
{Tete2=ajouterEnFin(Tete2, P->D);
 P=P->suivant;
}
return(Tete2);
}

int comparaison(struct donnees_monome X,
            struct donnees_monome Y)
{if(X.degre!=Y.degre ||X.coef!=Y.coef)    return(1);   else return(0); }

mon simplifierList(mon Tete)
{if(Tete==NULL || Tete->suivant==NULL)return(Tete);
mon p1=Tete;  mon p2;
while(p1->suivant!=NULL)
{p2=p1->suivant;
  while(p2!=NULL)
  { if(p2->D.degre==p1->D.degre)
      {p1->D.coef=p2->D.coef+p1->D.coef;       p2->D.coef=0;       }
  p2=p2->suivant;
  }
 p1=p1->suivant;
}
```

```
p1=Tete;
while(p1!=NULL)
{if(p1->D.coef==0)
  {if(p1->suivant==NULL)Tete=supprimerMonomeEnFin(Tete);
   else {p1->D=p1->suivant->D;
        p2=p1->suivant;
        p1->suivant=p1->suivant->suivant;
        free(p2);
       }
  }
p1=p1->suivant;
}
return(Tete); }

mon DerivList(mon Tete1)
{mon p=Tete1;
while(p!=NULL)
{p->D.degre=p->D.degre-1;  p->D.coef=p->D.degre*p->D.coef;  p=p->suivant;  }
return(Tete1); }

mon DerivOrdreN(mon Tete1 , int n)
{int i;   for(i=0;i<n;i++) Tete1=DerivList(Tete1);
return(Tete1);  }

mon sommePoly(mon A, mon B)
{ mon p1=A;
while(p1->suivant!=NULL)
{ p1=p1->suivant;}
p1->suivant=B;  A=simplifierList(A);  return(A); }

mon MultPoly(mon A, mon B)
{ if(A==NULL || B==NULL) return(NULL);       mon p1=A;  mon p2=B;  mon p3=NULL;
while(p1!=NULL)
{ p2=B;
  while(p2!=NULL)
  {DM X={p2->D.degre+p1->D.degre, p2->D.coef*p1->D.coef};
   p3=ajouterEnFin(p3,X);
```

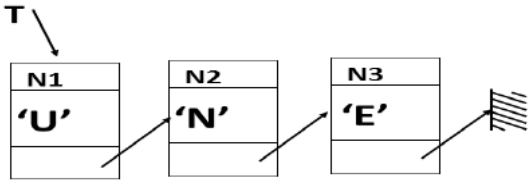
```
  p2=p2->suivant;  }
 p1=p1->suivant;}
 p3=simplifierList(p3);
return(p3);
}

mon rechercherElement(mon T,struct donnees_monome X)
{  mon tmp=T;
   /* Tant que l'on n'est pas au bout de la liste */
   while(tmp != NULL)
   {   if(comparaison(tmp->D,X)==0)
       {//Si l'élément a la valeur recherchée,
        //on renvoie son adresse */
        return tmp;
       }
       tmp = tmp->suivant;
   }
   return NULL;
}

mon TriBulList(mon Tete1)
{int taille=sizeof(struct noeud_monome);
mon nouvelleMonome=(mon)malloc(taille);
mon temp1=Tete1, temp2;
int i;
int n=lengthList(Tete1);
for(i=0;i<n-1;i++)
{temp1=Tete1;  temp2=temp1->suivant;
while(temp2!=NULL)
 {if(temp2->D.degre<temp1->D.degre)
  {nouvelleMonome->D=temp1->D;
   temp1->D=temp2->D;
   temp2->D=nouvelleMonome->D;}
 }
}
}
```

Série- N°3 : Listes simplement chaînées

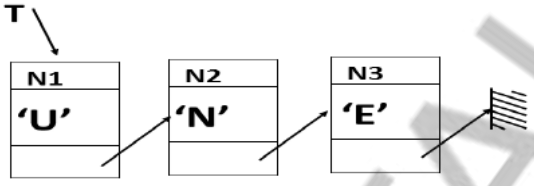
1) **Déclarer le type struct nœud contenant un caractère et l'adresse de son suivant.**
2) **Redéfinir le type struct nœud * par car.**
3) **Ecrire la fonction car ajouterEnFin(car T, char x) qui ajoute le caractère c en fin de la liste T.**
   Par exemple si la lite T est :



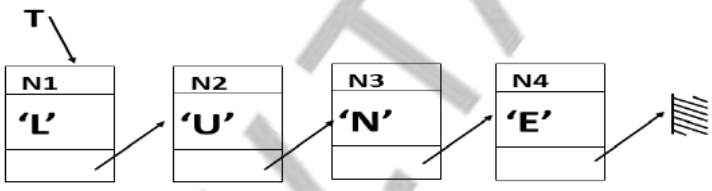
   L'appel de la fonction T= ajouterEnfin(T, 'E') retourne la liste suivante :



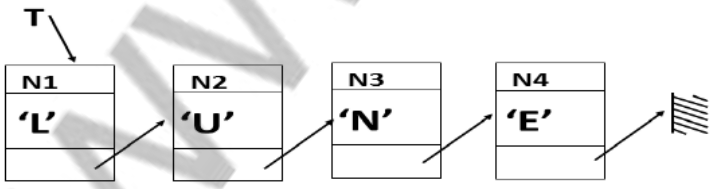
4) **Ecrire la fonction car ajouterEnTete(car T, char x) qui ajoute le caractère c en tête de la liste T.**
   Par exemple si la lite T est :



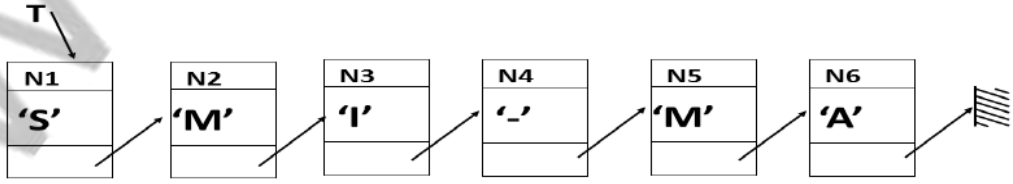
   L'appel de la fonction T= ajouterEnTete(T, 'L') retourne la liste suivante :



5) **Ecrire la fonction int longueur(car T) qui compte le nombre d'éléments de la liste T**
   Par exemple, si T est :



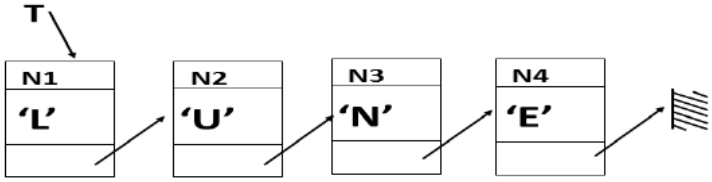
   L'appel de la fonction longueur(T) retourne le nombre 4.
6) **Ecrire la fonction car ConvertChaineToListeChainee(char X[]) qui convertie la chaine de caractères X en une liste chainée. Par exemple, l'appel de la fonction ConvertChaineToListeChainee("SMI-MA") retourne :**



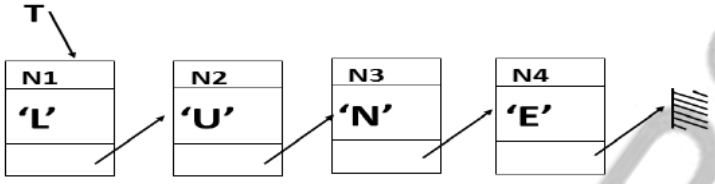
7) Ecrire la fonction char* ConvertListeChaineeToChaine(car T) qui convertie la liste chainée T en une chaine de caractères X. (l'opération inverse de la fonction car ConvertChaineToListeChainee(char X[])

8) Ecrire la fonction car InverseListeChainee (car T) qui retourne la liste chainée représentant l'inverse de T. Par exemple Si T contient les caractères A,B,C,D,E dans cette ordre. La liste retournée contiendra E,D,C,B,A dans cette ordre.

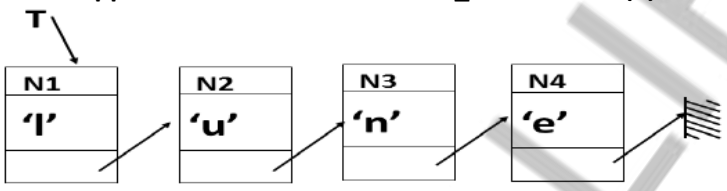
9) Ecrire la fonction int Existe(char x, car T) qui retourne 1 si x est dans T, elle retourne 0 sinon.
Par exemple, si T est la liste suivante, alors Existe('n', T) retourne 0 mais Existe('N', T) retourne 1.



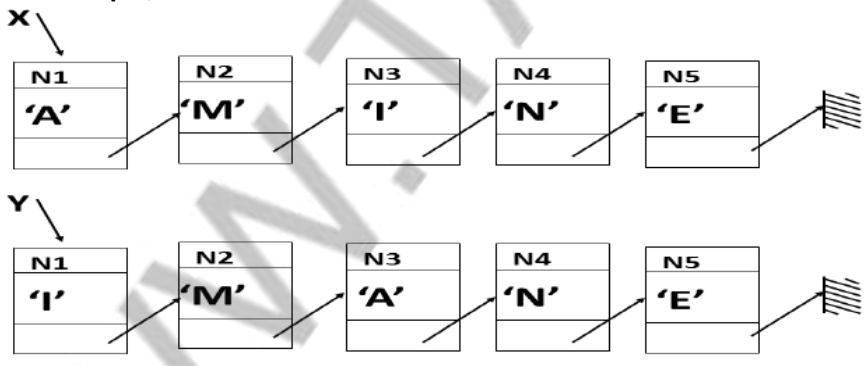
10) Ecrire la fonction void Minuscule_ListeChainee(car T) qui met en minuscule tous les caractères alphabétiques de la liste chainée T. Les autres éléments ne seront pas modifiés.
Par exemple, Si T est :



Alors l'appel de la fonction Minuscule_ListeChainee(T) rend T comme suit :



11) Ecrire la fonction int ComparerListesChaineesOrdreRespecte(car A, car B) qui retourne 1 si tous les éléments de A sont dans B et dans le même ordre et tous les éléments de B sont dans A.
Par exemple, Si X et Y sont :



L'appel de la fonction ComparerListesChaineesOrdreRespecte(X,Y) retourne 0.

12) Ecrire la fonction in ComparerListesChaineesOrdreQuelconque(car A, car B) qui retourne 1 si tous les éléments de A sont dans B et tous les éléments de B sont dans A dans un ordre quelconque.
Pour les deux listes X et Y de la question précédente, L'appel de la fonction ComparerListesChaineesOrdreQuelconque (X,Y) retourne 1.

Corrigé Série- N°3 : Listes simplement chaînées

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
struct noeud
{char c; struct noeud *nxt;};

typedef struct noeud* car;

car ajouterEnFin(car T, char x)
{   car nouvCaractere = (car)malloc(sizeof(struct noeud));
   nouvCaractere->c = x;
   nouvCaractere->nxt = NULL;
    if(T == NULL){return nouvCaractere;}
   else {car temp=T;
        while(temp->nxt != NULL)
          {temp = temp->nxt; }
        temp->nxt = nouvCaractere;
        return T;
       }
}

car ajouterEnTete(car T, char x)
{   car nouvCaractere = (car)malloc(sizeof(struct noeud));
   nouvCaractere->c = x;
   nouvCaractere->nxt = T;
   return nouvCaractere;
}

void afficherListeCaracteres(car Tete)
{   printf("\n\n affichage de la liste :\n");
   car ptrNoeudCourant=Tete;
   while(ptrNoeudCourant!=NULL)
   {printf("%c",ptrNoeudCourant->c);
   ptrNoeudCourant=ptrNoeudCourant->nxt;
   }
}

int longueur(car T)
{int n=0;
car P=T;
while(P!=NULL)
 {n++; P=P->nxt;}
return(n);
}
```

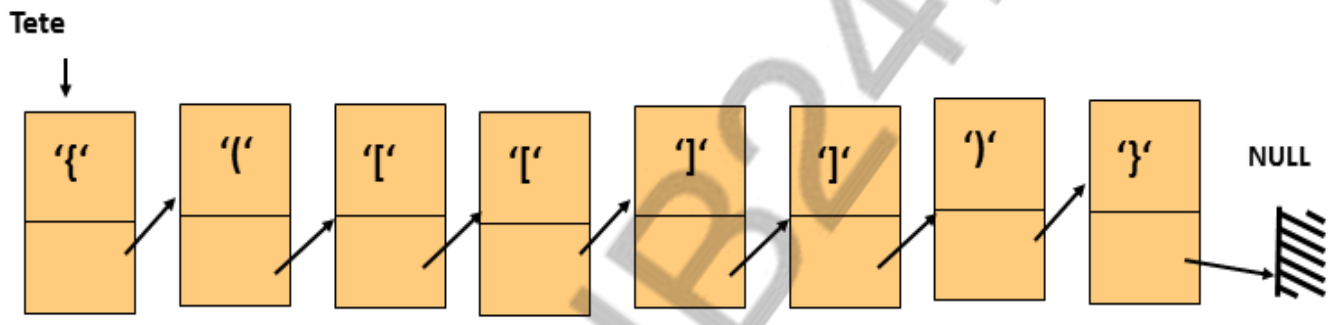
```
car ConvertChaineToListeChainee(char X[])
{int i; car T=NULL;
 for(i=0;i<strlen(X);i++)
    T=ajouterEnFin(T,X[i]);
 return(T);
}

char* ConvertListeChaineeToChaine(car T)
{int n=longueur(T);
 char*X=(char*)malloc((n+1)*sizeof(char));
 car p=T; int i=0;
 while(p!=NULL)
  {X[i]=p->c;
   p=p->nxt; i++;
  }
  X[i]='\0';
  return(X);
}

car InverseListeChainee(car T)
{car New=NULL; car p=T;
 while(p!=NULL)
    {New=ajouterEnTete(New,p->c);
    p=p->nxt;}
 return(New);
}

int Existe(char x, car T)
{car tmp=T;
 while(tmp!=NULL)
  {if(tmp->c==x) return(1);
   tmp=tmp->nxt; }
 return(1);
}

void Miniscule_ListeChainee(car T)
{car tmp=T;
 while(tmp!=NULL)
  {if('A'<=tmp->c && tmp->c<='Z')
   tmp->c = tmp->c+32;
   tmp=tmp->nxt;
  }
}

int ComparerListesChaineesOrdreRespecte(car A, car B)
{car p1=A, p2=B;
 while(p1!=NULL && p2!=NULL)
 {if(p1->c!=p2->c) return(0);
  p1=p1->nxt; p2=p2->nxt;
 }
 if(p1!=NULL || p2!=NULL) return(0);
 else return(1);
}
```

```
int ComparerListesChaineesOrdreQuelconque(car A, car B)
{car p1=A;
 while(p1!=NULL)
 {if(Existe(p1->c, B)==0) return(0);
  p1=p1->nxt;
 }
 p1=B;
 while(p1!=NULL)
 {if(Existe(p1->c, A)==0) return(0);
  p1=p1->nxt;
 }

 return(0);
}

main()
{  car T1,T2;
   car T=NULL;
   T=ajouterEnFin(T,'a');
   T=ajouterEnFin(T,'h');
   T=ajouterEnFin(T,'m');
   T=ajouterEnFin(T,'e');
   T=ajouterEnFin(T,'d');
   afficherListeCaracteres(T);
   getch();
   T1=ConvertChaineToListeChainee("Bonjour");
   afficherListeCaracteres(T1);
   T2=InverseListeChainee(T1);
   afficherListeCaracteres(T2);
   getch();
}
```

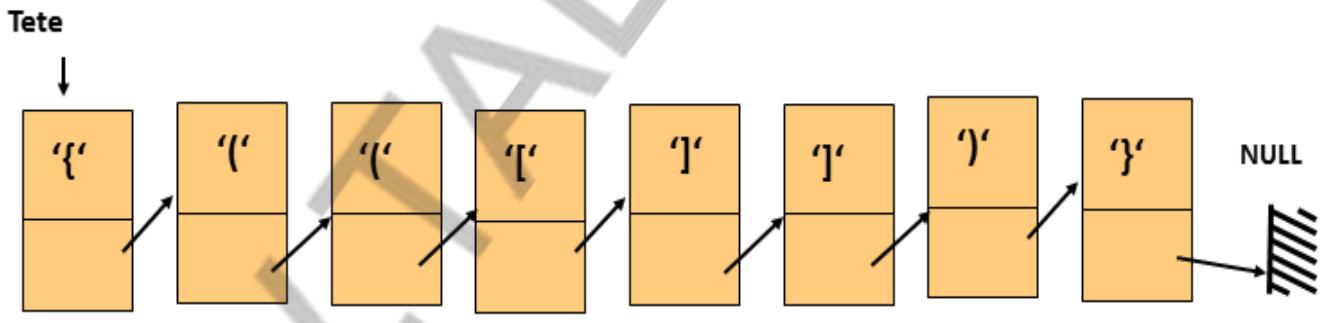
Série- N°4 : Listes chaînées (Suite)-Piles et Files

1) Quelle est la différence structurelle entre une liste doublement chainée et une liste simplement chainée
2) Quels sont les avantages d'une liste doublement chainée par rapport à une liste simplement chainée
3) En utilisant les fonctions développées dans le cours, écrire un programme (une version par tableau et une version par listes chainées) permettant de vérifier si une expression E (sous forme de chaine de caractères) est valide ou non. On se limitera à la vérification des caractères (délimiteurs) suivants : '(', '{', '[', '}', ']' et ')'.

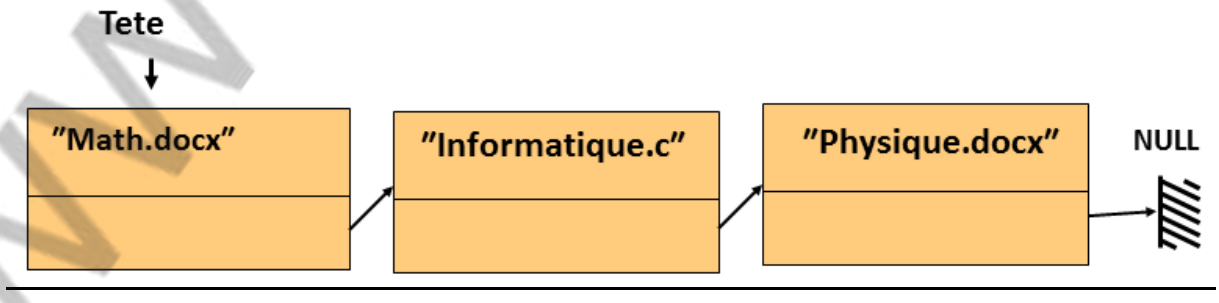
   Exemples :
   L'expression E= "int A={(3+T[6*M[4]]), 123}" qui sera représentée par la liste chainée ci-dessous est valide



   L'expression E= "int A={(3+T(6*M[4]]), 123}" qui sera représentée par la liste chainée ci-dessous n'est pas valide



4) En utilisant les fonctions développées dans le cours, écrire un programme (une version par tableau et une version par listes chainées) permettant de gérer l'impression des fichiers par une imprimante.
   Exemple :



Corrigé Série- N°4 : Listes chaînées (Suite)-Piles et Files

1) **Quelle est la différence structurelle entre une liste doublement chainée et une liste simplement chainée ?**
   Dans une liste simplement chainée, chaque nœud contient un pointeur sur le nœud suivant, mais dans une liste doublement chainée chaque nœud contient un pointeur sur le nœud suivant et le nœud précédent.
2) **Quels sont les avantages d'une liste doublement chainée par rapport à une liste simplement chainée ?**
   Dans une LDC (Listes Doublement Chainées) on peut accéder directement au dernier élément, sans parcourir toute la liste comme on le fait dans une LSC (liste simplement chainée). Donc l'ajout et la suppression du dernier élément sont moins complexe (sont plus rapides).

   On peut parcourir une LDC dans les 2 sens (on peut donc revenir en arrière). Par exemple, On peut vérifier si une liste est symétrique.

3)

```
#include<stdio.h>
#include<conio.h>
#define max 100

struct pile
{char vec[max];
 int sommet; };

struct pile p;

void raz()
{p.sommet=-1;}

int estVide()//vide
{if (p.sommet!=-1)
 return 0;
 else return 1;
}

void depiler()
 {  p.sommet--;  }
char dernier()
 {   return(p.vec[p.sommet]);  }

void empiler(char x)
 { p.sommet++; p.vec[p.sommet]=x;  }

void affichPile()
 {int i;printf("\n aff pile: ");
 for(i=0;i<=p.sommet;i++) printf(" %c ",p.vec[i]); }

int isDelimiter(char c)
{if(c=='{' ||c=='}' ||c=='[' ||c==']' ||c=='(' ||c==')' ) return 1;
else return(0);   }
```

```
main()
{
char ch[100]="int A={(3+T[6*M[4]]), 123}";
 int correct=1 , i=0;
 raz();
 //printf("entrer votre expression: \n");
 //gets(ch);
  //puts(ch);
 while(ch[i]!='\0')
   {
   if (isDelimiter(ch[i])==1)
   {
   switch(ch[i])
   {case '{' :empiler(ch[i]);  break;
   case '(' :empiler(ch[i]);  break;
   case '[' :empiler(ch[i]);  break;
   case '}' :if ( dernier()!='{' )
            {correct=0; break;}
            else {depiler(); break;}
   case ')' :if ( dernier()!='(' )
            {correct=0; break;}
            else {depiler(); break;}
   case ']' :if ( dernier()!='[' )
            {correct=0; break;}
            else {depiler(); break;}
   }
   }
   //printf("\n     %c",dernier());
   i++;
   //affichPile();
   };
 if (estVide()&&correct==1)
 printf("l'expression est valide \n");
 else
 printf("l'expression n'est pas valide \n");
getch();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
struct caractere
{char c; struct caractere *nxt;};

typedef struct caractere* pc;

void afficherListeCaracteres(pc Tete)
{   printf("\n\n affichage de la liste :\n");
   pc ptrNoeudCourant=Tete;
   while(ptrNoeudCourant!=NULL)
   {printf("%c",ptrNoeudCourant->c);
   ptrNoeudCourant=ptrNoeudCourant->nxt;
```

```
   }
}

pc Empiler(pc TeteActuelle, char x)
{   pc nouvCaractere = (pc)malloc(sizeof(caractere));
    nouvCaractere->c = x;
    nouvCaractere->nxt = NULL;
     if(TeteActuelle == NULL)
       {return nouvCaractere;}
    else
      {pc temp=TeteActuelle;
       while(temp->nxt != NULL)
         {temp = temp->nxt; }
          temp->nxt = nouvCaractere;
       return TeteActuelle;
       }
}

pc Depiler(pc TeteActuelle)
{   if(TeteActuelle == NULL) return NULL;
    if(TeteActuelle->nxt == NULL)
    {free(TeteActuelle);return NULL;}
     pc tmp = TeteActuelle;
     pc ptmp = TeteActuelle;
     while(tmp->nxt != NULL)
     {ptmp = tmp;
      tmp = tmp->nxt;}
     ptmp->nxt = NULL;
     free(tmp);
     return TeteActuelle; }

pc Enfiler(pc TeteActuelle, char x)
{Empiler(TeteActuelle,x);}

pc Defiler(pc TeteActuelle)
{   if(TeteActuelle != NULL)
    {pc aRenvoyer = TeteActuelle->nxt;
     free(TeteActuelle);
     return aRenvoyer;
    }
    else
    {return NULL;}
}
char dernier(pc T)
  {if(T != NULL)
    {pc tmp = T;
     while(tmp->nxt!=NULL) tmp=tmp->nxt;
     return tmp->c;
    }
    else
    {return '#';}   }
int isDelimiter(char c)
{if(c=='{' ||c=='}' ||c=='[' ||c==']' ||c=='(' ||c==')' ) return 1;   else return(0);   }
```

```
main()
{
char ch[100]="int A={(3+T[6*M[4]]), 123}";
 int correct=1 , i=0;
 pc T=NULL;
 //printf("entrer votre expression: \n");
 //gets(ch);
 while(ch[i]!='\0')
   {
   if (isDelimiter(ch[i])==1)
   {
     switch(ch[i])
      {case '{' :T=Empiler(T,ch[i]);  break;
       case '(' :T=Empiler(T,ch[i]);  break;
       case '[' :T=Empiler(T,ch[i]);  break;
       case '}' :if ( dernier(T)!='{' )
              {correct=0; break;}
            else {T=Depiler(T); break;}
       case ')' :if ( dernier(T)!='(' )
                 {correct=0; break;}
              else {Depiler(T); break;}
       case ']' :if ( dernier(T)!='[' )
              {correct=0; break;}
              else {T=Depiler(T); break;}
       default: break;
   }
   afficherListeCaracteres(T);
   }
   i++;
   };
 if (T==NULL&&correct==1)
 printf("l'expression est valide \n");
 else
 printf("l'expression n'est pas valide \n");
getch();
}
```

4)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>

struct noeud
{char nom[20]; struct noeud *nxt;};

typedef struct noeud* pe;

void afficherFile(pe Tete)
{   printf("\n\n affichage de la liste :\n");
   pe ptrNoeudCourant=Tete;
   while(ptrNoeudCourant!=NULL)
```

```
   {printf("%s\n",ptrNoeudCourant->nom);
   ptrNoeudCourant=ptrNoeudCourant->nxt;
   }
}

pe Enfiler(pe TeteActuelle, char nomF[])
{   pe nouvCaractere = (pe)malloc(sizeof(noeud));
   strcpy(nouvCaractere->nom , nomF);
   nouvCaractere->nxt = NULL;
    if(TeteActuelle == NULL)
      {return nouvCaractere;}
   else
     {pe temp=TeteActuelle;
      while(temp->nxt != NULL)
        {temp = temp->nxt; }
         temp->nxt = nouvCaractere;
     return TeteActuelle;
     }
}

pe Defiler(pe TeteActuelle)
{   if(TeteActuelle != NULL)
   {pe aRenvoyer = TeteActuelle->nxt;
    free(TeteActuelle);
    return aRenvoyer;
   }
   else
   {return NULL;}
}
main()
{pe T=NULL;
 int code, Encore=1; char NomFichier[20];
 while(Encore==1)
  {printf("\n Pour ajouter un fichier entrer 1");
   printf("\n Pour supprimer un fichier entrer -1");
   printf("\n Pour afficher la file entrer 2");
   printf("\n Pour Quiter entrer 0\n");
   scanf("%d",&code);
   switch(code)
    {case 1 : printf("\n Entrer le nom du fichier:");
              fflush(stdin);gets(NomFichier);
              T=Enfiler(T,NomFichier);  break;
     case -1: T=Defiler(T); break;
     case 2 : afficherFile(T); break;
     case 0 : Encore=0;
    }
  }
getch();
}
```

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define max_lignes 100
#define max_colones 20
struct file
{char vec[max_lignes][max_colones];
 int sommet;};

struct file f;

void raz()
{f.sommet=-1;}

void defiler()
  {int i;
   for(i=0;i<=f.sommet;i++)
      strcpy(f.vec[i],f.vec[i+1]);
   f.sommet--;
  }

void enfiler(char nom[])
  {if(f.sommet>=max_lignes-1)
    printf("\n Depassement de capacite de la file");
   else {f.sommet++; strcpy(f.vec[f.sommet],nom);} }

void affichFile()
  {int i;printf("\n aff file: \n");
  for(i=0;i<=f.sommet;i++) puts(f.vec[i]);
  }

main()
{raz();
 int code, Encore=1; char NomFichier[20];
 while(Encore==1)
   {printf("\n Pour ajouter un fichier entrer 1");
    printf("\n Pour supprimer un fichier entrer -1");
    printf("\n Pour afficher la file entrer 2");
    printf("\n Pour Quiter entrer 0\n");
    scanf("%d",&code);
    switch(code)
     {case 1 : printf("\n Entrer le nom du fichier:");
               fflush(stdin);gets(NomFichier);
               enfiler(NomFichier);  break;
      case -1: defiler(); break;
      case 2 : affichFile(); break;
      case 0 : Encore=0;
     }
   }
getch();
}
```

Série- N°5 : Arbres

Dans cette partie, on se propose de représenter des ensembles finis **d'entiers strictement positifs triés par ordre croissant** par des listes chaînées définies en **langage C** comme suit

```
typedef struct ens
        { int nombre; //un nombre entier strictement positif élément de l'ensemble
        struct ens *suiv; // l'adresse de l'élément suivant
        } ensembleListe;
```

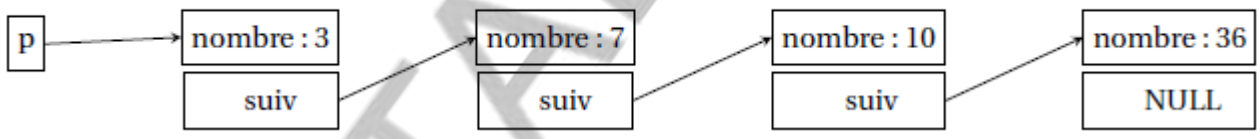
Appellation

On appellera **"Ensemble Liste d'adresse p"** une liste chaînée d'éléments de type **ensembleListe** (définie plus haut) et possédant les propriétés suivantes :

- Le premier élément a l'adresse **p.**
- Le dernier élément a dans son champ **suiv** la valeur **NULL**
- Pour tout élément d'adresse **el** de la liste chaînée, tel que (**el->suiv!=NULL**), on a (**0< el->nombre** < ((**el->suiv**) ->**nombre**) (liste triée par ordre croissant de nombres)

Exemple

L'ensemble {3, 7, 10, 36} sera représenté par **l'Ensemble Liste d'adresse p** comme suit :



Insertion d'un élément dans la liste chaînée triée

Soit la déclaration globale suivante : **ensembleListe *p ;**

On suppose avoir définie et inséré des éléments dans l' **"Ensemble Liste d'adresse p"** (**p** est déclaré plus haut).

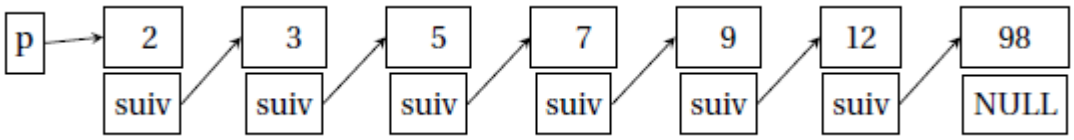
☞ Écrire une fonction d'entête : **void inserer(int val)** qui permet d'insérer **à sa place** l'élément de type **ensembleListe** dans l' **"Ensemble Liste d'adresse p"** pour que la liste reste toujours triée par ordre croissant. Cet élément a dans son champ **nombre**, la valeur **val** (paramètre de la fonction), en plus, on suppose que : (**val>p->nombre**)(voir rappel, remarque et exemple)

Rappel : L'appel de la fonction de la bibliothèque du **langage C malloc(n)** (n étant un entier positif), permet d'allouer n octets dans la mémoire dynamique et retourne l'adresse mémoire du block alloué. La fonction **malloc** est définie dans le fichier de la bibliothèque **stdlib.h**

Remarque

- Si le paramètre **val** est la valeur du champ **nombre** d'un élément qui existe déjà dans la liste, aucun élément ne sera inséré.

Exemple : Soit l' **"Ensemble Liste d'adresse p"** suivant :



1) Quelle est la différence ~~structurelle~~ entre un tas et un arbre binaire de recherche ?

Soit la déclaration suivante :

typedef struct element

{ int contenu; struct element* gauche; struct element* droit; } noeud ;

Déterminez les fonctions suivantes en langage C :

2) int Taille(struct element* arbre) qui retourne le nombre de nœuds de l'arbre

3) int Hauteur_arb (struct element* arb) : qui calcule et retourne la hauteur d'un arbre

4) int Maximum (struct element* arb): qui retourne le maximum de l'arbre arb

5) float Moyenne(struct element* arb): qui retourne la moyenne de l'arbre arb

6) int EstFeuille (struct element* X) qui vérifie si X est une feuille ou non

7) int NombreDeFeuilles(struct element* arb) : qui retourne le nombre de feuilles d'un arbre

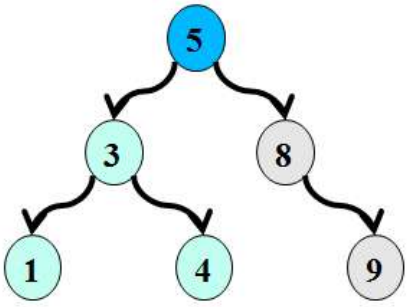
8) int EstComplet(struct element* arbre) qui vérifie si l'arbre est complet ou non

9) existe(arbre,nd): qui vérifie si le nœud nd appartient à l'arbre

10) int Nombre_noeuds_internes(struct element* arbre) qui permet de compter le nombre de nœuds internes d'un arbre (c-à-dire autres que les feuilles).

11) Définir la fonction int est_abr_bin_recherche(struct element* arb) qui vérifie si l'arbre arb est un arbre binare de recherche ou non et retourne 1 si oui ou 0 si non

12) Soit l'arbre binaire suivant :



Donner le résultat d'exécution des algorithmes NGD, GND, GDN et

ParcoursEnLargeur de cet arbre

13) Quelles sont les avantages de la méthode de tri par tas (tri maximier) ?

14) Implémenter le tri maximier en langage C.

15) Implémenter une fonction permettant d'insérer une valeur à la bonne place dans un arbre binaire de recherche

16) Implémenter une fonction permettant de supprimer une valeur précise d'un arbre binaire

Série- N°5 : Corrigé

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
typedef struct element
{  int contenu;    struct element* gauche;
  struct element* droit;  } noeud ;

noeud* CreerNoeud(int valeur)
{ noeud*N=(noeud*)malloc(sizeof(noeud));
  N->contenu=valeur;
  N->gauche = NULL;  N->droit = NULL;
  return N;}
noeud* ajouterNoeud(noeud* Arbre, char chemin[], int valeur)
{ int i;
  noeud* p=CreerNoeud(valeur);
  if(Arbre==NULL) return (p);
  noeud* temp=Arbre;
  for(i=0;((i<strlen(chemin)-1)&& (temp!=NULL));i++)
  {if(chemin[i]=='-') temp=temp->gauche;
     else temp=temp->droit;  }
  if(chemin[strlen(chemin)-1]=='-') temp->gauche=p;
  else temp->droit=p;
  return(Arbre);  }
```

```
int Nbr=0; float S=0;
void Moyenne(noeud* Arbre)
{if(Arbre!=NULL)
 {S+=Arbre->contenu; Nbr++;   Moyenne(Arbre->gauche);  Moyenne(Arbre->droit); }}

void ParcoursEnlargeur(noeud* Arbre)
   { if(Arbre!=NULL)
    {int n,i;    noeud** T=(noeud**)malloc(100*sizeof(noeud*));
    T[0]=Arbre; n=1;
    while(n>0)
      {printf("%d\n",T[0]->contenu);
      if (T[0]->gauche!=NULL)
         {T[n]=T[0]->gauche; n++;}
      if (T[0]->droit!=NULL)
         {T[n]=T[0]->droit; n++;
         }
      for(i=1;i<n;i++) T[i-1]=T[i];
      n--;      }    }  }
int max(int x,int y)
   {if(x<y) return(y);    else return(x);}

int hauteur(noeud* arbre)
   {if(arbre==NULL)       return(0);
   noeud* X=arbre->gauche;    noeud* Y=arbre->droit;    return(1+max(hauteur(X),hauteur(Y)));    }
int feuille(noeud* nd)
   {if(nd==NULL)return(0);    else if((nd->gauche==NULL) && (nd->droit==NULL))        return(1);
       else return(0);    }
```

```
int N=0;
void Nombre_de_Feuilles_NGD(noeud* Arbre)
{if(Arbre!=NULL)
 {if(feuille(Arbre)==1)N++;
  Nombre_de_Feuilles_NGD(Arbre->gauche);
  Nombre_de_Feuilles_NGD(Arbre->droit);
  }
}
int EstArbreBinaireComplet(noeud* A)
   {if((A==NULL) || (feuille(A)==1))
      return(1);
   else if(hauteur(A->gauche)==hauteur(A->droit))
       {int cond1=EstArbreBinaireComplet(A->gauche);
        int cond2=EstArbreBinaireComplet(A->droit);
        return(cond1 && cond2);
        }
   return(0);
   }
int Min;
void Minimum(noeud* Arbre)
{if(Arbre!=NULL)
 {if(Arbre->contenu<Min)Min=Arbre->contenu;
  Minimum(Arbre->gauche);
  Minimum(Arbre->droit);
 }
}
```

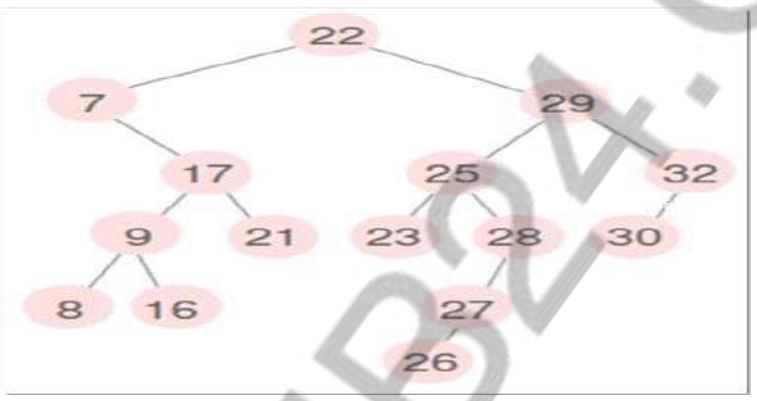
```
int MinArbre(noeud* A)
{Min=99999; Minimum(A); return(Min);  }
int Max;
void Maximum(noeud* Arbre)
{if(Arbre!=NULL)
 {if(Arbre->contenu>Max)
     Max=Arbre->contenu;
Maximum (Arbre->gauche);
Maximum(Arbre->droit); }}

int MaxArbre(noeud* A)
{Max=-1;
Maximum(A);
return(Max); }

int ABR(noeud* A)
   { if (A==NULL or feuille(A)==1)return 1;
   else if ((A->gauche==NULL)&&(A->droit!=NULL))
         {if(A->contenu<MinArbre(A->droit))  return (ABR(A->droit));      else return(0);}
  else if ((A->droit==NULL)&&(A->gauche!=NULL))
           {if(A->contenu>MaxArbre(A->gauche)) return(ABR(A->gauche));
              else return(0); }
   else {int cond1=(A->contenu>MaxArbre(A->gauche));
         int cond2=(A->contenu<MinArbre(A->droit));
         int cond3=ABR(A->gauche);
         int cond4=ABR(A->droit);
         return (cond1 && cond2 && cond3 && cond4);
       } }
```

Série- N°6 : Arbres(suite) et graphes

1) **Quel est l'avantage principal de l'algorithme de tri par tas par rapport à l'algorithme Tri à Bulles ?**

2) **Expliquer le principe de traitement de collision des clés de hachage à travers un exemple.**

3) **Soit l'arbre A suivant :**

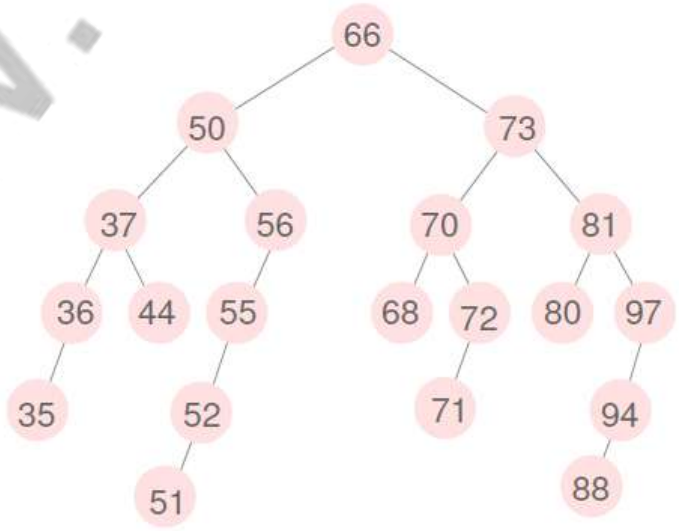


   a) **A est-il un arbre binaire ? Est-il un tas ?**

   b) **A est-il un arbre binaire complet ? équilibré ?**

   c) **A est-il un arbre binaire de recherche ? si oui, insérer les valeurs suivantes dans A de telle sorte qu'elle reste arbre binaire de recherche : 1, 18, 20.**

4) **Soit l'arbre X suivant :**



Pour supprimer un nœud N d'un arbre binaire de recherche X en en le gardant de recherche, on distingue 3 cas :

- Si N n'as pas de fils, N sera supprimé directement.

- Si N possède un seul fils alors il sera remplacé par ce fils.

- Si N possède deux fils alors N sera remplacé soit par le plus grand nœud de son sous arbre gauche ou bien par le plus petit nœud de son sous arbre droit.

Supprimer les nœuds suivants : 35, 52 et 73 de l'arbre ci-dessus.

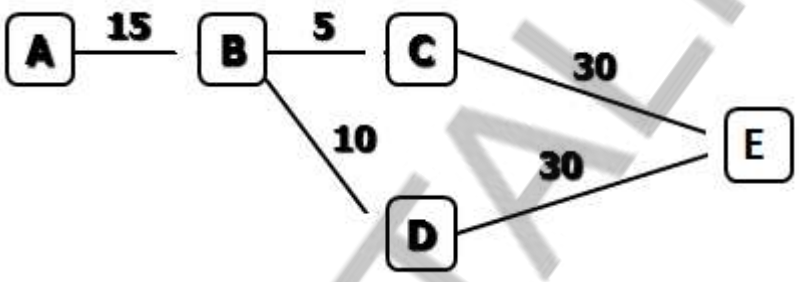
++

++++++++++++++++++++++++++++++++++ **Graphes** ++++++++++++++++++++++++++++++++++

1) Tracer (forme de graphe : nœuds et arrêtes) le graphe correspondant à la matrice d'adjacence suivante :

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

2) Donner la matrice de coûts associé au graphe ci-dessous.



3) Donner une *Chaine Eulérienne pour le graphe de la question 2)*

4) Donner les résultats d'exécution des algorithmes de parcours en largeur et en profondeur sur le graphe précédent

5) En utilisant l'algorithme de Dijkstra, colorer en rouge le chemin le plus court entre la ville N° 0 et la ville N° 6.