```
typedef struct ettruc {
          int * champ1 ;
          int champ2 ;
          int champ3 ;
          struct ettruc * champ4 ; } * Truc ;

void fonction1 (Truc t, int n) {
          t = (Truc)malloc(sizeof(struct ettruc)) ;
          t->champ1 = (int *) malloc(sizeof(int)*n) ;
          t->champ2 = n ;
          t->champ3 = 0 ;
          t->champ4 = NULL ;
}
Truc fonction2 (Truc t, int v) {
          if (t->champ2 == t->champ3) {
                    Truc taux = (Truc)malloc(sizeof(Truc)) ;
                    taux->champ1 = (int *) malloc(sizeof(int)*t->champ2) ;
                    taux->champ2 = t->champ2 ;
                    taux->champ3 = 0 ;
                    taux->champ4 = t ;
                    t = taux ;
          }
          *((t->champ1)+champ3) = v ;
          t->champ3 ++ ;
          return t ;
}
void fonction3 (Truc t) {
          if (t != NULL) {
                    fonction3(t->champ4) ;
                    for (int i = 0; i < t.champ3; i++)
                    printf("%d ",*((t.champ1)+i)) ;
          }
}
int main() {
          Truc t ;
          fonction1(t,2) ; // etape 1
          printf("t : ") ; fonction3(t) ; // etape 2
          t = fonction2(t,10) ; // etape 3
          t = fonction2(t,20) ; t = fonction2(t,0) ; // etape 4
          printf("t : ") ; fonction3(t) ; // etape 5
```

1. Réécrire l'extrait de code *((t->champ1)+i) en considérant que t->champ1 est un tableau et en utilisant les notations sur les tableaux.

On aura : t->champ1[i]

2. Expliquer ce qu'il se passe lors des passages de paramètre pour les étapes 1 et 3 et quels sont les impacts sur la variable t du main.

Etape 1 : Passage par valeur, donc la valeur du t du main est recopiée dans le t de fonction1 ; puis ce t de fonction1 est maj par fonction1 mais comme ce n'est qu'une recopie du t du main, le t du main ne change pas de valeur.

Etape 3 : on a la même chose mais comme il y a un return du t de fonction2 à la fin de l'exécution de fonction2 et que le t du main reçoit cette valeur de retour, du coup, le t du main va être modifie par l'exécution de fonction2

3. En utilisant votre réponse à la question précédente, analyser tout le code en identifiant les erreurs éventuelles et les corrections possibles pour ces erreurs, et le résultat des affichages après correction

Solution : Plusieurs erreurs de syntaxe et de sémantique :

   a. fonction1, 1 erreur de sémantique : il faudrait que la maj du t de fonction1 ait un impact sur le t du main ; donc 2 solutions :
      i. soit on fait un return t ; et on modifie l'appel dans le main par t = fonction1(t,2) ;
      ii. soit on rajoute une * devant le t ds les paramètres et sur la ligne du malloc, on remplace chaque autre t dans le corps par (*t) et on change l'appel ds le main par fonction1(&t,2) ;
   b. fonction2, 2 erreurs de syntaxe : ds le sizeof le type est erroné (il faut struct ettruc) et dans l'expression *((t->champ1)+champ3) l'utilisation de champ3 n'est pas possible (il faut mettre t->champ3)
   c. fonction3, erreurs de syntaxe : t étant un pointeur sur une structure l'accès aux champs se fait avec -> et pas avec . Ce code crée une liste chainée composée de tableaux de 2 cases. Dans la première cellule, on place les valeurs 10, puis 20. Puis comme le tableau de la première cellule est plein, on crée une seconde cellule dans laquelle on met 0. L'affichage se fait ensuite dans l'ordre des valeurs insérées Bonus : la structure de données Truc correspond à ArrayList qu'on implémente ds l'exo 2.

Solution :

- Le premier affichage donne : (la liste est vide) t :
- Le premier affichage donne : (la liste contient 3 valeurs) t :10 20 0

4. Y-a-t-il des fonctions récursives ? Si oui, dire lesquelles et donner leur type (primitive, terminale, transformable terminale) en justifiant votre réponse.

Solution : Il y a 1 fonction récursive : fonction3. Elle est primitive puisqu'elle n'apparait pas dans les arguments de l'appel récursif. Elle n'est pas terminale puisqu'il reste des actions à faire après l'appel récursif (la boucle for). Et elle n'est pas transformable-terminale puisqu'on ne peut pas exhiber une loi associative liant l'appel et ce qui reste à faire après l'appel.

On veut implémenter une version en langage C des ArrayList 2 contenant des éléments de type TElem. Cette implémentation va se faire sous la forme d'une liste chaînée de tableaux d'éléments (chaque tableau étant de la même taille et cette taille étant définie lors de l'initialisation).

```
typedef struct etAList {
          TElem * tabElements ; // le tableau des elements
          int tailleTab ; // la taille de ce tableau
          int nbElements ; // le nb d'elements significatifs ds le tableau
          struct etAList * celSuivante ; // l'acc`es `a la cellule suivante de la liste
} * ArrayList ;
```

initArrayList : prend en paramètre la taille pour les tableaux et renvoie une ArrayList ne contenant pour l'instant aucun élément

```
ArrayList initArrayList (int n) {
          ArrayList l = (ArrayList)malloc(sizeof(struct etAList)) ;
          l->tabElements = (TElem *) malloc(sizeof(TElem)*n) ;
          l->tailleTab = n ;
          l->nbElements = 0 ;
          l->celSuivante = NULL ;
}
```

addArrayList : prend en param`etres une ArrayList<TElem> et un ´el´ement de type TElem et renvoie l'ArrayList<TElem> pass´ee en param`etre dans laquelle on aura ajouté
en fin de liste l''el´ement pass´e en param`etre

```
ArrayList addArrayList (ArrayList l, TElem v) {
          if (l->tailleTab == l->nbElements) {
                    ArrayList laux = (ArrayList)malloc(sizeof(struct etAList)) ;
                    laux->tabElements = (int *) malloc(sizeof(int)*l->tailleTab) ;
                    laux->tailleTab = l->tailleTab ;
                    laux->nbElements = 0 ;
                    laux->celSuivante = l ;
                    l = laux ;
          }
          *((l->tabElements)+(l->nbElements)) = v ;
          // ou bien l->tabElements[l->nbElements] = v
          l->nbElements ++ ;
          return l ;
}
```

printArrayList : prend en paramètre une ArrayList et affiche son contenu de la premi`ere valeur à la derniere ; on considère qu'il existe une fonction void afficheElement (TElem e) qui se charge de l'affichage d'un élément donné

```
void printArrayList (ArrayList l) {
          if (l != NULL) {
                    printArrayList(l->celSuivante) ;
                    for (int i = 0; i < l->nbElements; i++)
                    afficheElement(l->tabElements[i]) ;
          }
}
```

isEmptyArrayList : prend en param`etre une ArrayList et renvoie true si cette liste est vide et false sinon

```
int isEmptyArrayList(ArrayList l) {
          return (l == NULL || (l->celSuivante == NULL && l->nbElements == 0)) ;
}
```

sizeArrayList : prend en param`etre une ArrayList et renvoie le nombre d''el´ements stock´es dans cette liste (on ne demande pas ici une version tr`es optimis´ee)

```
int sizeArrayList(ArrayList l) {
          if (l == NULL)return 0 ;
          return l->nbElements + sizeArrayList(l->celSuivante) ;
}
```

getArrayList : prend en param`etres une ArrayList et un entier i et renvoie l''el´ement qui est en position i dans la liste (i devra ˆetre ≥ 0 et < `a la taille de la liste) ; la position 0 correspondra au premier ´el´ement de la liste

```
TElem getArrayList(ArrayList l, int i) {
          int posFinTab = sizeArrayList(l) - 1 ;
          // correspondra tjrs a la position du dernier element du tableau de la cellule courante
          while (1) {
                    if (i < posFinTab - l->nbElements +1) {
                    // posFinTab - l->nbElements +1 correspond au d´ebut du tableau
                    // la case i n'est pas ds la cellule courante
                              posFinTab = posFinTab - l->nbElements ;
                              l = l->celSuivante ;
                    }
                    else { // la case i est ds la cellule courante
                              return (l->tabElements[l->nbElements - posFinTab + i -1]);
                    }
          }
}
```

removeArrayList : prend en param`etres une ArrayList et un entier i et renvoie l'ArrayList pass´ee en param`etre de laquelle on aura enlev´e l''el´ement qui ´etait en position i (i devra ˆetre ≥ 0 et < `a la taille de la liste). Attention, votre solution devra optimiser en priorit´e le temps d'ex´ecution de la fonction, puis dans un second temps la place m´emoire, tout en ´evitant les fuites m´emoire !

```
ArrayList removeArrayList(ArrayList l, int i) {
          ArrayList lc = l ; // acces cellule courante
          ArrayList lplc = NULL ; // acces cellule pr´ec´edente de la lc
          int posFinTab = sizeArrayList(l) - 1 ; // correspondra tjrs a la position
          // du dernier element du tableau
          // de la cellule courante
          while (1) {
                    if (i < posFinTab - lc->nbElements +1) {
                              // posFinTab - l->nbElements +1 correspond au d´ebut du tableau
                              // la case i n'est pas ds la cellule courante
                              posFinTab = posFinTab - lc->nbElements ;
                              lplc = lc ;
                              lc = lc->celSuivante ;
                    }
                    else {    // la case i est ds la cellule courante
                              if (lc->nbElements == 1) { // un seul element ds la cellule : celui qu'on remove
                                        if (lplc != NULL) lplc->celSuivante = lc->celSuivante ;
                                        else l = lc->celSuivante ;
                                        free(lc->tabElements) ;
                                        free(lc) ;
                                        break ;
                              }
                              else {    // decalage `a faire jusqu'`a la position correspondant `a i
                                        for (int j=lc->nbElements-1;j>=lc->nbElements-posFinTab+i;j--)
                                        lc->tabElements[j-1] = lc->tabElements[j] ;
                                        lc->nbElements -- ;
                                        break ;
                              }
                    }
          }
          return l ;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gibert_module_image_v2.h"
/*-----------------------------------------------------------------------------
Lit et mémorise la matrice à partir des valeurs lues sur l'entrée standard.
Elle reçoit en paramètre les valeurs N et M correspondant respectivement au nombre de lignes et au nombre de colonnes de la matrice représentant
l'image.
Ces valeurs auront été lues et testées par le programme principal.
De plus, cette fonction renverra :
          [0]       si la mémorisation ne peut pas être faite,
          [1]       sinon (traitement effectué).
-----------------------------------------------------------------------------*/
int lire_image(int N, int M, int image[N][M]){
  int i,j,val;
  printf("\n\r[Lecture de la matrice de %d lignes et %d colonnes] : ",N,M);
  if(N<=NB_MAX && N>0 && M<=NB_MAX && M>0){
    for(i=0;i<N;i++){
      for(j=0;j<M;j++){
        scanf("%d",&val);
          image[i][j]=val;
      }
    }
    return 1;
  }
  else return 0;
}

/*-----------------------------------------------------------------------------
Effectue la transformation de l'image suivant le niveau de gris, les valeurs de N (nombre de lignes de la matrice) et M (nombre de colonne de la
matrice) en paramètre.
Elle veillera a enregistrer les nouvelles valeurs dans la matrice mémorisée.
De plus, elle renverra une valeur entière qui vaudra suivant le cas :
          [-1]      si la valeur du niveau est supérieure à 256,
          [0]       si le niveau n'est pas une puissance de 2,
          [1]       sinon (traitement effectué).
-----------------------------------------------------------------------------*/
int afficher_image_codee_bis(int niveau, int N, int M, int image[N][M]){
  int i,j;
  printf("\n\r[Passage vers une image a %d niveau de gris]\n\r",niveau);
  if(niveau<256){
    if((int)log2(niveau)==log2(niveau)){
      for(i=0;i<N;i++){
        for(j=0;j<M;j++){
          image[i][j]=((image[i][j])*niveau)/256;
        }
      }
      affiche_image(N,M,image);
    }
    else return 0;
  }
  else return -1;
  return 1;
}

/*-----------------------------------------------------------------------------
Fonction image_miroir qui affiche l'image en inversant les colonnes.
-----------------------------------------------------------------------------*/
void image_miroir(int N, int M, int image[N][M]){
  int i,j,temp;
  printf("\n\r[Transformation de l'image en image miroir]\n\r");
  for(i=0;i<N;i++){
    for(j=0;j<M/2;j++){
      temp=image[i][N-1-j];
      image[i][N-1-j]=image[i][j];
      image[i][j]=temp;
    }
  }
  affiche_image(N,M,image);
}


/*-----------------------------------------------------------------------------
Sous-programme supplémentaire permettant simplement d'afficher la matrice
-----------------------------------------------------------------------------*/
void affiche_image(int N, int M, int image[N][M]){
  int i,j;
  printf("Affichage de l'image\n\r");
  for(i=0;i<N;i++){
    for(j=0;j<M;j++){
      printf("%5d",image[i][j]);
    }
    printf("\n\r");
  }
  printf("Affichage termine\n\r");
}

/*-----------------------------------------------------------------------------
Fonction rotation_image qui affiche l'image après une rotation de 90° ou 180°.
L'angle de rotation étant passé en paramètre.
-----------------------------------------------------------------------------*/
void rotation_image(int N, int M, int angle, int image[N][M]){
  int i,j,temp;

  affiche_image(N,M,image);
  printf("\n\r[Rotation de %d° de l'image]\n\r",angle);
  switch(angle){
    case 180 :
      for(i=0;i<N/2;i++){
        for(j=i;j<N-i-1;j++){
          temp=image[i][j];
          image[i][j]=image[j][N-1-i];
          image[j][N-1-i]=image[N-1-i][N-1-j];
          image[N-1-i][N-1-j]=image[N-1-j][i];
          image[N-1-j][i]=temp;
        }
      }
    case 90 :
      for(i=0;i<N/2;i++){
        for(j=i;j<N-i-1;j++){
          temp=image[i][j];
          image[i][j]=image[j][N-1-i];
          image[j][N-1-i]=image[N-1-i][N-1-j];
```

```
          image[N-1-i][N-1-j]=image[N-1-j][i];
          image[N-1-j][i]=temp;
        }
      }
      affiche_image(N,M,image);
      break;
    default:
      printf("Erreur : Valeur d'angle inconnue\n\r");
      break;
  }
}

/*-------------------------------------------------------------------------------
Fonction histogramme qui calcule et affiche l'histogramme d'une image.
Ici un histogramme correspond à la fréquence avec laquelle chaque niveau de gris est présent dans la totalité de l'image.
Cette fréquence est ramenée à un pourcentage.
-------------------------------------------------------------------------------*/
void histogramme(int N, int M, int niveau, int image[N][M]){
  int i,j,tab_histogramme[niveau];
  printf("\n\r[Histogramme]\n\r");
  for(i=0;i<niveau;i++) tab_histogramme[i]=0;
  for(i=0;i<N;i++) for(j=0;j<M;j++) tab_histogramme[image[i][j]]+=1;
  for(i=0;i<niveau;i++) if(tab_histogramme[i]!=0) printf("Fréquence de %d dans l'image : %d pourcent\n\r",i,(tab_histogramme[i]*100)/(N*M));
  printf("Toutes autres fréquences dans l'image : 0 pourcent\n\r");
}


/*-------------------------------------------------------------------------------
Fonction qui copie les valeurs d'une matrice dans une autre matrice
-------------------------------------------------------------------------------*/
void copie(int N, int M, int image[N][M], int image_copie[N][M]){
  int i,j;
  for(i=0;i<N;i++) for(j=0;j<M;j++) image_copie[i][j]=image[i][j];
  printf("\n\rCopie : OK\n\r");
}


/*-------------------------------------------------------------------------------
Fonction affiche les différentes images selon le format suivant :
image1 | image_miroir1
image2 | image_miroir2
-------------------------------------------------------------------------------*/
void affichage_formate(int N1, int M1, int N2, int M2, int image1[N1][M1],int image_miroir1[N1][M1],int image2[N2][M2],int
image_miroir2[N2][M2]){
  int i,j;
  printf("\n\r[Affichage formate]\n\r");
  for(i=0;i<N1;i++){
    for(j=0;j<((M2/2)-(M1/2));j++) printf("%5c",' ');
    for(j=0;j<M1;j++) printf("%5d",image1[i][j]);
    for(j=0;j<((M2/2)-(M1/2));j++) printf("%5c",' ');
    printf("%5c%5c%5c",' ','*',' ');
    for(j=0;j<((M2/2)-(M1/2));j++) printf("%5c",' ');
    for(j=0;j<M1;j++) printf("%5d",image_miroir1[i][j]);
    printf("\n\r");
  }
  printf("\n\r");
  for(i=0;i<(M2*2)+3;i++) printf("%5c",'*');
  printf("\n\r\n\r");
  for(i=0;i<N2;i++){
    for(j=0;j<M2;j++) printf("%5d",image2[i][j]);
    printf("%5c%5c%5c",' ','*',' ');
    for(j=0;j<M2;j++) printf("%5d",image_miroir2[i][j]);
    printf("\n\r");
  }
}
```

element.h LIFO

```
typedef int ELEMENT;

void affiche_ELEMENT(ELEMENT elm);
void affect_ELEMENT(ELEMENT *elm1,ELEMENT elm2);
int compare_ELEMENT(ELEMENT elm1,ELEMENT elm2);
ELEMENT saisir_ELEMENT();
```

element.c LIFO

```
#include <stdio.h>
#include "element.h"
/*-------------------------------------------------------------------
Affiche l'élément
-------------------------------------------------------------------*/
void affiche_ELEMENT(ELEMENT elm){
          printf("%d ",elm);
}
/*-------------------------------------------------------------------
Affecte un élément a un autre élément
-------------------------------------------------------------------*/
void affect_ELEMENT(ELEMENT *elm1,ELEMENT elm2){
          *elm1=elm2;
}
/*-------------------------------------------------------------------
Compare un élément à un autre élément
-------------------------------------------------------------------*/
int compare_ELEMENT(ELEMENT elm1,ELEMENT elm2){
          return (elm1==elm2);
}
/*-------------------------------------------------------------------
Permet de saisir un élement
-------------------------------------------------------------------*/
ELEMENT saisir_ELEMENT(){
          ELEMENT elm;
          scanf("%d",&elm);
          return elm;
}
```

pile_dynamique.h

```
#include "element.h"
#include <stdlib.h>

typedef struct cel {
          ELEMENT elm;
          struct cel *suiv;
}cel;

typedef struct cel* PILE;

PILE init_PILE();
void affiche_PILE(PILE p);
int PILE_estVide(PILE p);
PILE emPILE(PILE p,ELEMENT x);
PILE dePILE(PILE p,ELEMENT *x);
PILE saisir_PILE();
```

pile_dynamique.c

```
#include <stdio.h>
#include "pile_dynamique.h"
/*--------------------------------------------------------------------
Initialise une pile en respectant le prototype : PILE init_PILE()
--------------------------------------------------------------------*/
PILE init_PILE(){
          PILE p=NULL;
          return p;
}
/*--------------------------------------------------------------------
Affiche tous les ELEMENTs d'une pile donnee en parametre en respectant le prototype : void affiche_PILE(PILE)
--------------------------------------------------------------------*/
void affiche_PILE(PILE p){
          while(!PILE_estVide(p)){
                    affiche_ELEMENT(p->elm);
                    p=p->suiv;
          }
          printf("\n\r");
}
/*--------------------------------------------------------------------
Permet de tester si une pile donnee en parametre est vide en respectant le prototype : int PILE_estVide(PILE)
--------------------------------------------------------------------*/
int PILE_estVide(PILE p){
          return (p==NULL);
}
/*--------------------------------------------------------------------
Empile un ELEMENT (donne en parametre) dans une pile (donnee en parametre) en respectant le prototype : PILE emPILE(PILE, ELEMENT)
--------------------------------------------------------------------*/
PILE emPILE(PILE p,ELEMENT x){
          PILE paux = malloc(sizeof(PILE));
          if(paux!=NULL){
                    affect_ELEMENT((&paux->elm),x);
                    paux->suiv=p;
                    p=paux;
          }
          return p;
}
/*--------------------------------------------------------------------
Depile une pile (donnee en parametre), cette fonction doit aussi renvoyer l'ELEMENT qui etait en tete de pile en respectant le prototype : PILE
dePILE(PILE,ELEMENT *).
--------------------------------------------------------------------*/
PILE dePILE(PILE p,ELEMENT *x){
          if (!PILE_estVide(p)){
                    PILE paux=p;
                    affect_ELEMENT(x,paux->elm);
                    p=p->suiv;
                    free(paux);
          }
          return p;
}
/*--------------------------------------------------------------------
Permet de saisir une pile en demandant a l'utilisateur d'entrer les ELEMENT un par un et en les inserant dans la pile en respectant le prototype
: PILE saisir_PILE().
--------------------------------------------------------------------*/
PILE saisir_PILE(){
          PILE p=init_PILE();
          ELEMENT x;
          char test;
          int i=1;
          do{
                    printf("\n\rSaisir une valeur [%d] : ",i++);
                    x=saisir_ELEMENT();
                    p=emPILE(p,x);
                    printf("Voulez-vous continuer ? [Y/N] : ");
                    scanf(" %c",&test);
          }while(test=='Y');
          return p;
}
```

file_dynamique.h

```
#include "element.h"
#include <stdlib.h>

typedef struct cel {
          ELEMENT elm;
          struct cel *suiv;
}cel;

typedef struct MA_FILE{
          cel *tete;
          cel *queue;
}MA_FILE;

MA_FILE INIT_FILE();
void AFFICHE_FILE(MA_FILE p);
int FILE_EST_VIDE(MA_FILE p);
MA_FILE ENFILER(MA_FILE p,ELEMENT x);
MA_FILE DEFILER(MA_FILE p,ELEMENT *x);
MA_FILE SAISIR_FILE();
```

file_dynamique.c

```
#include <stdio.h>
#include "file_dynamique.h"

/*--------------------------------------------------------------------
Initialise une MA_FILE en respectant le prototype : MA_FILE INIT_FILE()
--------------------------------------------------------------------*/

MA_FILE INIT_FILE(){
          MA_FILE p;
          p.tete=NULL;
          p.queue=NULL;
          return p;
}

/*--------------------------------------------------------------------
Affiche tous les ELEMENTs d'une MA_FILE donnee en parametre en respectant le prototype : void AFFICHE_FILE(MA_FILE)
--------------------------------------------------------------------*/

void AFFICHE_FILE(MA_FILE p){
          while(!FILE_EST_VIDE(p)){
                    affiche_ELEMENT(p.tete->elm);
                    p.tete=p.tete->suiv;
          }
          printf("\n\r");
}

/*--------------------------------------------------------------------
Permet de tester si une MA_FILE donnee en parametre est vide en respectant le prototype : int FILE_EST_VIDE(MA_FILE)
--------------------------------------------------------------------*/

int FILE_EST_VIDE(MA_FILE p){
          return (p.tete==NULL);
}

/*--------------------------------------------------------------------
Enfile un ELEMENT (donne en parametre) dans une MA_FILE (donnee en parametre) en respectant le prototype : MA_FILE emPILE(MA_FILE, ELEMENT)
--------------------------------------------------------------------*/

MA_FILE ENFILER(MA_FILE p,ELEMENT x){
          MA_FILE paux;
          paux.tete = malloc(sizeof(cel));
          if(!FILE_EST_VIDE(paux))
          {
                    affect_ELEMENT((&paux.tete->elm),x);
                    paux.tete->suiv=NULL;
                    if(p.tete==NULL)p.tete=paux.tete;
                    else p.queue->suiv=paux.tete;
                    p.queue=paux.tete;
          }
          return p;
}

/*--------------------------------------------------------------------
Defile une MA_FILE (donnee en parametre), cette fonction doit aussi renvoyer l'ELEMENT qui etait en tete de MA_FILE en respectant le prototype :
MA_FILE DEFILER(MA_FILE,ELEMENT *).
--------------------------------------------------------------------*/

MA_FILE DEFILER(MA_FILE p,ELEMENT *x){
          if (!FILE_EST_VIDE(p))
          {
                    MA_FILE paux=p;
                    affect_ELEMENT(x,paux.tete->elm);
                    paux.tete=p.tete;
                    p.tete=p.tete->suiv;
                    if(p.tete==NULL)p.queue=NULL;
                    free(paux.tete);
          }
          return p;
}

/*--------------------------------------------------------------------
Permet de saisir une MA_FILE en demandant a l'utilisateur d'entrer les ELEMENT un par un et en les inserant dans la MA_FILE en respectant le
prototype : MA_FILE SAISIR_FILE().
--------------------------------------------------------------------*/

MA_FILE SAISIR_FILE(){
          MA_FILE p=INIT_FILE();
          ELEMENT x;
          char test;
          int i=1;
          do{
                    printf("\n\rSaisir une valeur [%d] : ",i++);
                    x=saisir_ELEMENT();
                    p=ENFILER(p,x);
                    printf("Voulez-vous continuer ? [Y/N] : ");
                    scanf(" %c",&test);
          }while(test=='Y');
          return p;
}
```