Structures de données en C
TD N°2

Exercice 1

```
/**********(Q1) ************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
//-------------------------------------------------------------------------------------------------------------------
//(a) La structure d'un élément de la liste simplement chaînée:
typedef struct ElementRepere{
  int valeur;
  struct ElementRepere *suivant;
}Element;
//-------------------------------------------------------------------------------------------------------------------
//(b) La structure de contrôle de la liste :
typedef struct ListeRepere{
  Element *tete;
  Element *queue;
  int nef;   //le nombre final d'éléments que contient la liste
} Liste;
//-------------------------------------------------------------------------------------------------------------------
//(c) La fonction CreerElement( ) :
Element* CreerElement( ){
 Element* El;
 El=(Element*)malloc(sizeof(Element));
 El->valeur=0;  /* initialisation de sa valeur à 0 */
 El->suivant=NULL;  /* initialisation de son suivant à NULL */
 return(El);
}
//-------------------------------------------------------------------------------------------------------------------
//(d) La fonction CreerListe( ) qui crée une liste vide:
Liste* CreerListe( ){
 Liste* Li;      /
 Li= (Liste*)malloc(sizeof(Liste));
 Li->tete=NULL; //Initialisation
 Li->queue=NULL;//Initialisation
 Li->nef=0;//Initialisation
 return(Li);
}
//-------------------------------------------------------------------------------------------------------------------
//(e) la fonction CreerListeOrdre( ) qui crée une liste en respectant l'ordre de lecture:
/*Méthode 1: la boucle for*/
void CreerListeOrdre(Liste *Li){
  int vaf;/*valeur a ajouter*/
  int nb, i;
  Element *El;
```

```
  /*saisie des données*/
  printf("Taper le nombre d'éléments: ");
  scanf("%d", &nb);
  for(i=1; i<=nb; i++){
     printf("Saisir un nombre: ");
     scanf("%d", &vaf);
     El=CreerElement();/*appel a la fonction qui crée un élément vide*/
     if(Li == NULL || El == NULL){
       exit(EXIT_FAILURE);  //Erreur
     }
     El->valeur=vaf; /*mettre une valeur dans le nouveau élément */

     if(Li->tete==NULL){   /*cas de la liste vide*/
          Li->tete=El;
          Li->queue=El;
          Li->nef++;
     }

     else{ /*cas de la liste non vide*/
          Li->queue->suivant=El;
          Li->queue=El;
          Li->nef++;
     }
  } //fin de for
}//fin de la fonction
//-------------------------------------------------------------------------------------------------------------------------
//(f) La fonction CreerListeCroissante( ) qui crée une liste en rangeant les entiers par ordre croissant
/*Méthode 2: la boucle do-while*/
void CreerListeCroissante(Liste *Li){
  int val;/* valeur à ajouter */
  Element *El, *courant, *pred;
  do{//faire tant que le nombre saisi est différent de 0
    /*la saisie des données*/
     printf("Saisir un nombre non nul: ");
     scanf("%d", &val);

     if(val != 0){ /*éliminer 0 pour ne pas le considérer comme élément de la liste*/
       El=CreerElement();/*appel à la fonction qui crée un élément vide*/
       if(Li == NULL || El == NULL){
         exit(EXIT_FAILURE);  //Erreur
       }
       El->valeur=val; /*mettre une valeur dans le nouveau élément */

     /*cas de la liste vide*/
       if(Li->tete==NULL){
         Li->tete=El;
         Li->queue=El;
         Li->nef++;
       }/*Fin de if*/
```

```
//*cas d'une/liste contenant plus éléments et la valeur du 1er élément est supérieur à la valeur à insérer
      else if(Li->tete->valeur >= val){/*ajout au début*/
        El->suivant=Li->tete;
        Li->tete=El;
        Li->nef++;
      }/*Fin de else if*/

 //cas d'une/liste contenant plus éléments et la valeur du 1er élément est inférieur à la valeur à insérer
      else{
        courant=Li->tete;
        pred=NULL;
        while((courant!=NULL)&&(courant->valeur < val)){
          pred=courant;
          courant=courant->suivant;
        }/*Fin de while*/
        El->suivant=courant;
        pred->suivant=El;
       //si on a fait l'insertion à la fin
        if(courant == NULL){
          Li->queue=El;
        }
        Li->nef++;
      }//fin de else
    }/*fin de if*/
}while((val) != 0);/*Fin de do- while */
}/*Fin de la fonction*/

/************************(Q2) ************************************************************/
//(a) La fonction AjoutFin( ):
void AjoutFin(Liste *Li, int vaf){
  Element *El;
  El=CreerElement();
  if(Li == NULL || El == NULL){
      exit(EXIT_FAILURE);  //Erreur
  }
  El->valeur=vaf;
  if(Li->tete==NULL){   /*cas de la liste vide*/
     Li->tete=El; /
     Li->queue=El;
     Li->nef++;
     /*Li->nef=Li->nef+1; */
  }
  else{ /*cas de la liste non vide*/
     Li->queue->suivant=El;
     Li->queue=El;
     Li->nef++;
  }
}
//-------------------------------------------------------------------------------------------------------------------------
```

```
//(b) La fonction Concatenation( ):
Liste *Concatenation(Liste *Li1, Liste *Li2){
 Liste *Li;
 Element *courant;

 Li=CreerListe();//pour contenir le résultat
 if(Li==NULL){
      exit(EXIT_FAILURE);
 }

 /*Parcours de liste 1*/
 courant=Li1->tete;
 while(courant!=NULL){
    AjoutFin(Li,courant->valeur);
    courant=courant->suivant;
 }

 /*Parcours de liste 2*/
 courant=Li2->tete;
 while(courant!=NULL){
    AjoutFin(Li,courant->valeur);
    courant=courant->suivant;
}
return Li;
}
//-----------------------------------------------------------------------------------------------------------------------
//(c) La fonction RechercheElem( ):
int RechercheElem(Liste* Li, int vr){
  int Var;
  Element* ptr;
  if(Li == NULL){
    exit(EXIT_FAILURE);
  }
  if(Li-> tete== NULL){
    Var=0;
  }
  else{
    ptr = Li -> tete;
    while((ptr->suivant !=NULL) && (ptr->valeur != vr)){
        ptr = ptr->suivant;
    }
    if(ptr->valeur==vr){
          Var=1;
    }
    else{
          Var=0;
    }
  } //fin de else
return(Var);
}//fin de la fonction
```