Licence Fondamentale Sciences de l'informatique	Niveau : LFSI 1

Correction TD n°3 Atelier de programmation II

Objectifs	Pointeurs, Listes chainées simples
Proposé Par	Grira Hajer

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>  // pour utiliser Sleep pour aider srand() à réinitialiser les valeurs
#include <time.h>     // pour utiliser srand au cas ou rand() génère des nombres identiques

typedef struct  {
                int score;
                struct LesScores * suivant;
              } LesScores;

typedef  LesScores* Pliste;

Pliste creerListe ( int );      // fonction qui renvoie un pointeur sur le début de la liste
void supprimFin ( Pliste ) ;    //  permet de supprimer le dernier élément de la liste si elle contient
plus qu'un élément
void ajoutFin (Pliste) ;          //  permet d'ajouter un entier saisi au clavier à la fin de la liste
Pliste recherche (Pliste, int) ;  // renvoie l'adresse de l'entier en argument, NULL si inexsistant
void affichListe( Pliste );           // afficher les éléments de la liste
Pliste detMin (Pliste) ;        // renvoyer l'adresse du minimum de la liste
int occurrences (Pliste, int) ;   // renvoyer le nombre d'occurrences de l'entier passé en argument
Pliste ExtraitInf  (Pliste *, int );  // renvoie un pointeur sur une nouvelle liste contenant les éléments
                                // cette fonction reçoit l'adresse de la tête de la liste initiale
void main () {
 Pliste t, p, t2;
int x, rep, n;
t=NULL;
do {
      printf("\n 1 ==> Creer Liste generee aleatoirement ");
      printf("\n 2 ==> supprimer un element a la fin de la liste :");
      printf("\n 3 ==> Ajouter un element a la fin de la liste");
      printf("\n 4 ==> Recherche dans la liste ");
      printf("\n 5 ==> Afficher la liste ");
      printf("\n 6 ==> Afficher le minimum de la liste ");
      printf("\n 7 ==> Determiner les occurences d'un entier ");
      printf("\n 8 ==> Extraire une sous liste ");
      printf("\n 0 ==> QUITTER");
      printf("\n\n Saisir Votre Choix : ");
      scanf("%d", &rep);
```

```
      switch (rep) {
      case 1 : do {
                system("cls"); // permet d'effacer l'écran
                printf("\ndonnez nombre éléments au moins un : ");
                scanf("%d",&n);
                } while (n<1);
                t = creerListe ( n);
            break;
      case 2 : supprimFin(t);
            break;
      case 3 :ajoutFin(t);
            break;
      case 4 : system("cls");
            printf("\n Donnez l'entier à chercher :");
            scanf("%d", &x);
            p=recherche(t,x);
            if (p!=NULL) printf("\n %d appartient à la liste", x);
               else printf("\n %d n'appartient pas à la liste", x);
            break;
      case 5 :system("cls");
            affichListe(t);
            break;
      case 6 : p=detMin(t);
            system("cls");
            printf("\n %d est la valeur minimale de la liste", p->score );
            break;
      case 7 :system("cls");
            printf("\n Donnez l'entier  :");
            scanf("%d", &x);
            n=occurrences(t,x);
            printf("\n Le nombre d'occurences de %d dans la liste = %d", x,n);
            break;
      case 8 :system("cls");
            printf("\n Donnez l'entier :");
            scanf("%d", &x);
            t2=ExtraitInf(&t,x);  // &t permet le passage par variable de la tete t au cas ou les nombres à
               affichListe(t2);       // extraire figurent au début de la liste initiale qui voit sa tête modifiée
            break;
       }
   } while (rep>0);
}
Pliste creerListe ( int n) {
Pliste L, prec,ne;
   int i;
   L = (Pliste) malloc(sizeof(LesScores));
   srand(time(NULL));
   L->score=rand()%100;
   L->suivant=NULL;
   prec=L;
```

```
   for(i=0;i<n-1;i++)
   {
      ne = malloc(sizeof(LesScores));
      ne->score=rand()%100;
      ne->suivant = NULL;
      prec->suivant =(struct LesScores*)ne;
      prec=ne;
   }
   return(L);
   }
void affichListe(Pliste t) {
   Pliste p;
   p=t;
   while (p!=NULL) {
   printf("%d ", p->score);
   p= (Pliste)p->suivant;
   }
}

void supprimFin ( Pliste t ) {    //  permet de supprimer le dernier élément de la liste si elle contient
plus qu'un élément
   Pliste p, anc;
   if (t !=NULL) {
      if (t->suivant !=NULL){
         p=t;
         while (p->suivant !=NULL) {
            anc=p;
            p= (Pliste)p->suivant ;
         }
         anc->suivant=NULL;
         free(p);
      }
   }
}
void ajoutFin (Pliste t) {          //  permet d'ajouter un entier saisi au clavier à la fin de la liste
   int x;
   Pliste p, ne;
   if (t) {
      printf("\nDonnez un élément à ajouter : ");
      scanf("%d", &x);
      p=t;
      while (p->suivant!=NULL)
         p=(Pliste)p->suivant;
      ne = (Pliste)malloc(sizeof(LesScores));
      ne->score=x;
      ne->suivant = NULL;
      p->suivant=(struct LesScores*)ne;
   }
}
```

```
Pliste recherche (Pliste t, int x) {  // renvoie l'adresse de l'entier en argument, NULL si inexsistant
   Pliste p;
   p=t;
   while ((p!=NULL) && (p->score !=x))
      p=(Pliste)p->suivant;
   return p;
}

Pliste detMin (Pliste t) {         // renvoyer l'adresse du minimum de la liste
   Pliste p, mini;
   mini=t;
   p=(Pliste)t->suivant;
   while (p!=NULL) {
      if (p->score < mini->score)
         mini=p;
      p=(Pliste)p->suivant;
   }
   return mini;
}
int occurrences (Pliste t, int x) {   // renvoyer le nombre d'occurrences de l'entier passé en argument
   int occ;
   Pliste p;
   p=t;
   occ=0;
   while (p!=NULL) {
      if (p->score==x)
         occ++;
      p=(Pliste)p->suivant;
   }
   return occ;
}
Pliste ExtraitInf  (Pliste *t , int x ){  // renvoie un pointeur sur une nouvelle liste contenant les
éléments
  Pliste tete, p, prec, anc, ne;
   tete=NULL;
   p=*t;
   prec=p;
   while ((p!=NULL) && (p->score >= x)) {
      prec=p;
      p=(Pliste)p->suivant;
   }
   printf("\n ********** debut extraction ");
   if (p!=NULL) {
         ne=p;
         if (p==*t) {
            *t=(Pliste)(*t)->suivant;
            prec=*t;
            p=*t;
         }
```

```
          else
        {
           prec->suivant = p->suivant;
           p=(Pliste)prec->suivant;
        }
        tete=ne;
        tete->suivant=NULL;
        anc=tete;
        while ( p!=NULL) {
           if ((p->score) <x) {
               ne=p;
               if (p==*t) {
                  *t=(Pliste)(*t)->suivant;
                  p=*t;
                  prec=*t;
               }
               else {
                  prec->suivant=p->suivant;
                  p=(Pliste)prec->suivant;
               }
               anc->suivant = (struct LesScores*)ne;
               ne->suivant=NULL;
               anc=ne;
           } else
           {
              prec=p;
              p=(Pliste)p->suivant;
           }
        }
   }
   return (tete);
}
```