**Université Sultan Moulay Slimane** **Année universitaire 2019/2020**
**Faculté des Sciences et Techniques**
**Béni-Mellal**
**Département d'Informatique**

# Correction TD3
# Structures de données en langage C
# Parcours MIPC

## Exercice 1 :

```
# include<stdio.h>
# include<stdlib.h>
# define N 20

typedef struct elt{
      int info;
      struct elt * suivant ;
} Element;

typedef struct {
      int taille;
      Element *debut;
      Element *fin;
} File;

void InitialiserFile(File *F)  { // fonction initialiser une File ;
            F->taille=0; F->debut=NULL; F->fin=NULL;
}

int fileVide(File F) {  return (F.taille==0); } // fonction file est Vide ?

int Enfiler(File *F, int val)  { // fonction enfiler ;
Element * element;
if ((element=(Element *)malloc(sizeof(Element)))==NULL) return -1;
element->info=val;
element->suivant=NULL;
if (F-> debut==NULL) F->debut=element;
else F->fin->suivant=element;
F->fin=element;
F->taille++;
return 0 ;
}
```

```
int Defiler(File *Fi)   { // fonction defiler ;
 int t;
 Element *supp;
 if (fileVide(*Fi)) { printf("la file est vide \n"); return -1; }
 else {   supp=Fi->debut;
          Fi->debut=Fi->debut->suivant;
          if (Fi->debut==NULL) Fi->fin=NULL;
          Fi->taille--;
          t=supp->info;
          free(supp);
          return t; }
}

int DefilerEnfiler(File *F1,File *F2) {
int x;
x=Defiler(F1);
if (x==-1) return -1;
else return Enfiler(F2, x);
}

void afficher(File Fi)   {  // fonction affichage ;
   while(!fileVide(Fi)) printf(" %d      ", Defiler(&Fi));
   printf("\n");
}

int comparer(File F1, File F2) {
  int x,y;
  x=Defiler(&F1);  y=Defiler(&F2);
  if (x>=y) return 1; else return 0;
}

void trier_vers(File F1,File *F2) { // fonction Trier ;
  int min, m, t, s, val ;
  File F3;
  if (fileVide(F1))  printf("la file est vide \n ") ;
  else     {
           InitialiserFile(&F3);
           while (!fileVide(F1)) {
                DefilerEnfiler(&F1,&F3);
                 if (!fileVide(F1)) {
                                t=1;  m=F1.taille;
                                while(t<=m) {
                                      val=comparer(F1,F3);
                                       if(val==0) { DefilerEnfiler(&F1,&F3); DefilerEnfiler(&F3,&F1);}
                                      else DefilerEnfiler(&F1,&F1);
                                      t++;
                                } //while
              } //if
              DefilerEnfiler(&F3,F2);
            } //while
          }//else
}
```

```
   main(){  // fonction main ;
   File f1, f2;
   int n;
   InitialiserFile(&f1);
   InitialiserFile(&f2);
   printf("entrer les elts de la file et taper un nombre négatif pour terminer : \n");
   while(1) {
            scanf("%d",&n);
      if(n<=0) break; else Enfiler(&f1,n);
   }
  printf("\n**********la file 'avant' le tri est : **********\n");
  afficher(f1) ;
  trier_vers(f1,&f2);
  printf("\n**********la file 'après' le tri est : **********\n");
  afficher(f2) ;
  system("pause");
}
```

## Exercice 2 :

```
#include<stdio.h>
#include<stdlib.h>
#define Max 20

   typedef struct elt {
        int priorite;  // priorite représente la priorité de l'élément de type entier
        char donnee;   // donnée porte une valeur de type char
        struct elt * precedent;
        } Element;

   typedef struct  {
        Element * Sommet;
        int taille;
        } Pile;

void initialisation(Pile * P) { P->Sommet=NULL;  P->taille=0; }

int Pile_vide(Pile P) { return (P.Sommet==NULL);}

int ajouter_ordre_quelconque(Pile *P, int prio, char donnee) {
   Element * element;
   if ((element=(Element *)malloc(sizeof(Element)))==NULL) return -1;
   element->priorite=prio;
   element->donnee=donnee;
   element->precedent=P->Sommet;
   P->Sommet=element;
   P->taille++;
   return 0;
}
```

```
Element *retirer_ordre_quelconque(Pile  *P) {
   Element *Courant, *Maxi, *x;
   int m,i,j,k;
   Element *tab[Max];
   if (Pile_vide(*P)) {printf(" Stack Underflow "); return NULL; };
   Maxi=P->Sommet; P->Sommet=P->Sommet->precedent; P->taille--;
   m=P->taille;  i=1;
   k=0; tab[k]=Maxi;
     while (i<=m) {
        Courant=P->Sommet; P->Sommet=P->Sommet->precedent;P->taille--;
        k++; tab[k]=Courant;
        if (Courant->priorite > Maxi->priorite)  Maxi=Courant;
        i++;
      }
   j=k;
   while (j>=0) {     // retirer les éléments pour arriver à l'élément le plus prioritaire
         x=tab[j];
         if (x->priorite!=Maxi->priorite) ajouter_ordre_quelconque(P, x->priorite, x->donnee);
         j--;
   }
   return Maxi;
}

int Ajouter_ordre(Pile *P, int prio, char donnee) {
    int i,j,trouve;
    Element *tab[Max];
    Element * E, *Courant;

    if ((E=(Element *)malloc(sizeof(Element)))==NULL) return -1;
    E->priorite=prio;
    E->donnee=donnee;
    if (Pile_vide(*P))  {E->precedent=NULL; P->Sommet=E; P->taille++;}
    else   {
            j=-1; trouve=1;
            while ((!Pile_vide(*P)) && (trouve==1)) {
               Courant=P->Sommet; P->Sommet=P->Sommet->precedent;P->taille--;
               if (E->priorite < Courant->priorite) { j++; tab[j]=Courant; }
               else    { trouve=0;
                         Courant->precedent=P->Sommet; //remeetre courant
                         P->Sommet=Courant;
                         P->taille++;
                       }
            } //fin while
            // ajouter element  et puis après le tab
           E->precedent=P->Sommet; P->Sommet=E; P->taille++;
           for(i=j;i>=0;i--) {tab[i]->precedent=P->Sommet; P->Sommet=tab[i]; P->taille++;}
      }  //fin else
   return 0;
}
```