Université Sultan Moulay Slimane **Année universitaire 2015/2016**
Faculté des Sciences et Techniques
Béni-Mellal
Département d'Informatique

Correction TP1
Structures de données en langage C
Parcours MIPC

Exercice 1 : *pile et élément avec priorité*

```
#include<stdio.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
         } element;

   typedef element Pile[Max]; /* les variables globales */
          Pile p;
          int Sommet;

   void initialisation(void) { Sommet=-1; }

   int Pile_vide(void) { return (Sommet<0);}

   int Pile_pleine(void) { return (Sommet>=Max-1);}

   void ajouter_ordre_quelc(element x) {
   if (Pile_pleine()) printf(" Stack overflow : dépassement de la capacité");
   else {
       Sommet=Sommet+1;
       p[Sommet]=x;
      }
   }

   int recherche_pile_non_vide() { // fonction pour chercher l'indice l'élément le plus prioritaire
          int max=0,pos;
          int indice=Sommet;
          element x;
          while (indice>=0){
             x=p[indice];
             if (x.priorite > max) {max=x.priorite; pos=indice;}
             indice--;
          }
```

```
        return pos;
   }

   element retirer_ordre_quelc() {
   element x,t;
   int indice,i,j;
   element tab[Max];
   if (Pile_vide()) printf(" Stack Underflow ");
   else {
    indice=recherche_pile_non_vide();
    j=-1;
    while (Sommet!=indice) { // retirer les éléments pour arriver à l'élément le plus prioritaire
        x=p[Sommet]; Sommet=Sommet-1;
        j++;
        tab[j]=x;
        }
       t=p[Sommet];       // ici Sommet = indice ;  indice de l'élément le plus prioritaire
      Sommet=Sommet-1;
     for(i=j;i>=0;i--) ajouter_ordre_quelc(tab[i]);  //rendre les éléments retirés dans le même
                                                     //ordre avant l'extraction.
     return t;
    }
}

void ajouter_ordre(element x) {
    element t,y;
    int i,j,trouve;
    element tab[Max];
    if (Pile_pleine()) printf(" Stack overflow : dépassement de la capacité");
    else {
        if (Pile_vide()) {Sommet=Sommet+1; p[Sommet]=x;}
        else {
            j=-1; trouve=1;
            while ((!Pile_vide()) && (trouve==1)){
            y=p[Sommet];
            if (x.priorite<y.priorite) {
                            j++;
                            tab[j]=y;
                            Sommet=Sommet-1;
                            }
            else trouve=0;
            } //fin while
            Sommet=Sommet+1;p[Sommet]=x;    //ajout element x
            for(i=j;i>=0;i--) {Sommet=Sommet+1;p[Sommet]=tab[i];}
        }  //fin else
   }
}
```

Exercice 2 : Gérer une vaisselle sale

```
#include<stdio.h>
#define Max_File 5
#define Max_Assiettes 10

typedef struct {
        int PA[Max_Assiettes];
        int Nbre_Assiettes;
        } Pile_Assiettes;

typedef Pile_Assiettes File[Max_File];

int ar;
File F;

int Pile_vide(Pile_Assiettes P) {
return(P.Nbre_Assiettes==-1);
}

int Pile_pleine(Pile_Assiettes P) {
return(P.Nbre_Assiettes>=Max_Assiettes-1);
}

int Depiler(Pile_Assiettes *P) {
int x;
if (Pile_vide(*P)) { printf("pile est vide \n"); return -1 ;}
else {
x=(*P).PA[(*P).Nbre_Assiettes];
(*P).Nbre_Assiettes=(*P).Nbre_Assiettes-1;
return x;
}
}

int Empiler(Pile_Assiettes *P,int x) {
if (Pile_pleine(*P)) return -1;
else{
    (*P).Nbre_Assiettes=(*P).Nbre_Assiettes+1;
    (*P).PA[(*P).Nbre_Assiettes]=x;
    return 0;
   }
}

void Pile_initialisation(Pile_Assiettes *P) {
(*P).Nbre_Assiettes=-1;
}
```

```
Creer_pile(Pile_Assiettes *p , int n) {
int i;
Pile_initialisation(p);
for (i=1;i<=n;i++) Empiler(p,i);
}

Pile_Assiettes Creation_Pile(){
Pile_Assiettes P;
Pile_initialisation(&P);
return P;
}

void Initialisation_file () {ar=-1;}

void File_push(Pile_Assiettes elt) {
if (ar>=Max_File-1) printf("la file est pleine \n");
else {ar++; F[ar]=elt;}
}

int File_vide() { return (ar==-1);}

int File_pleine() { return (ar>=Max_File-1);}

int Retirer_assiette() {
int assiette,i;
if (File_vide()) printf("la file est vide \n ");
else {
     assiette=Depiler(&F[0]);
     if (Pile_vide(F[0])) {
                    for (i=1; i<=ar; i++) F[i-1]=F[i];
                    ar=ar-1;
                    }
     return assiette;
     }
}

int completer_file(int x) {
int r, nb_ass, k, j;
Pile_Assiettes P;
   r=0;k=0;
   while ((r<=ar) && (k<x)) {
          nb_ass=F[r].Nbre_Assiettes+1;
        while (!Pile_pleine(F[r])&& (k<x)) {nb_ass++; Empiler(&F[r],nb_ass);k++;}
        r++;
   }
   if (k>=x) return 0;
   else
```

```
    {
    while (!File_pleine()&&(k<x)) {
         P=Creation_Pile();nb_ass=0;File_push(P);
         while (!Pile_pleine(F[ar])&& (k<x)) {nb_ass++; Empiler(&F[ar],nb_ass);k++;}
    }
    if (k>=x) return 0; else return (x-k);
    }
}

Afficher() {
int i;
if (File_vide()) printf("la file est vide \n ");
else
     for(i=0; i<=ar; i++) printf("pile : %d Nbre d'assiettes %d\n",i,F[i].Nbre_Assiettes+1);
}

main(){
int t;
Pile_Assiettes p1, p2, p3, p4, p5;
Initialisation_file ();
Creer_pile(&p1 , 6);
Creer_pile(&p2 , 4);
Creer_pile(&p3 , 9);

File_push(p1);
File_push(p2);
File_push(p3);

Afficher();
printf("************\n");
printf("*****ajout *******\n");
t=completer_file(30);
printf("*****%d assiettes restantes *******\n", t);
Afficher();
system("pause");
}
```