PARTIE VI : PILE ET FILE D'ATTENTE EN LANGAGE C

Les piles et les files d'attente sont des structures de données qu'on peut étudiées sous forme d'un tableau : piles ou files d'attente statiques ou sous forme d'une liste chaînée : piles ou files d'attente dynamiques. (Voir cours).

La structure de données Pile suit la politique de gestion **FILO** (First In Last Out) et la structure de données File d'attente la politique de gestion **FIFO** (First In First Out).

Essayons d'implémenter une pile P statique. Elle est représentée par un tableau Tab plus un indice de parcours qu'on appellera sommet.

```
/* Implémentation*/

#define Maxpile 10    /*On suppose que la pile contient 10 éléments de 0 à 9*/

Typedef  struct  pile  pile ;

Struct   pile    {   int  sommet;

                     Item   Tab[Maxpile];                /* item est un type à préciser*/

                 };

Pile  P;
```

Après avoir implémenté la pile P, on va devoir écrire quelques opérations de base.

Dans ce qui va être écrit, on a convenu que pour dire oui cela équivaut à 1 et non est équivaut à 0, ou le contraire c'est à vous de choisir !

OPERATIONS DE BASE

1. **Initialiser une pile.**

```
Void  InitialiserPile(pile *P)
{
        P.sommet = -1 ; /* C'est une convention*/
}
```

2. Fonction pour dire qu'une pile est vide ou pas.

```
Int  PileVide(pile P)
{
  If P.sommet = = -1
      Return 1;
   Return 0;
}
```

3. Fonction pour dire qu'une pile est pleine ou pas.

```
Int PilePleine(pile P)

{

    If P.sommet = = Maxpile-1

         Return 1;

    Return 0 ;

}
```

4. Procédure pour empiler une valeur à la pile P (ajout ou Push).

```
Void  Push(Pile *P, item valeur)
{
   If PilePleine(P) = = 1
        Printf(" Erreur: Pile pleine!");
    Else
        {
          P.sommet = P.sommet + 1;
          P.Tab[P.sommet] = valeur;
        }
}
```

5. **Procédure pour Dépiler une valeur de la pile P (Suppression ou Pop).**

```
Void Pop(Pile *P, item *valeur)
{
    If PileVide(P) = = 1
        Printf(''Erreur : Pile vide !) ;
    Else
        {
          Valeur = P.Tab[P.sommet] ;
          P.sommet = P.sommet -1 ;
        }
}
```

Essayons maintenant d'implémenter une pile dynamique sous forme d'une liste chaînée.

```
/* Implémentation*/

#include <stdlib.h>

Typedef struct Pile {

                    Item info ;          /*item un type à définir*/

                    Struct Pile *suiv ;  /*Autres littératures utilisent prec au lieu */

                   } Pile ;              /*de suiv tout dépend de l'angle de vue !*/

Pile *P = NULL ;
```

OPERATIONS DE BASE

Une structure de donnée dynamique n'est jamais pleine donc notre pile P l'est aussi !

On utilise Spile comme le nom donné au paramètre formel, on aurait pu l'appeler P simplement. De toute façon c'est nous qui choisissons les noms !

I. **Fonction pile vide.**

```
Int PileVide(Pile *Spile) /*paramètre formel appelé Spile pour Sommet de P*/
{
  If Spile == NULL
     Return 1 ;
   Return 0 ;
}
```

II. Procédure empiler (Push).

```
Void Push(Pile **Spile, item valeur)
{
  Pile *nouveauSommet = malloc(Sizeof *nouveauSommet);

  If nouveauSommet == NULL  /*vérifier s'il est créé*/

  {

        nouveauSommet->info = valeur ;

        nouveauSommet->suiv = *Spile ;

        *Spile = nouveauSommet ;

  }
```

/* Il est prudent de vérifier la création du nouveau sommet mais pas nécessaire !*/

III. Fonction dépiler (Pop).

```
Item Pop(Pile **Spile)
{
   Item res ;
   If VidePile(Spile) = = 0 /* Non elle n'est pas vide*/
    {
        Pile *Pt = (*Spile)->suiv ;
        Res = (*Spile)->info ;
        Free(*Spile), *Spile= NULL ;
        *Spile = Pt;
    }
    return res ;
}
```

NB : Il est plus judicieux d'utiliser une procédure pour afficher un message d'erreur en cas où la pile est vide car on se demande que vat-elle retourner cette fonction dans le cas où la pile est vide !

IV. Procédure dépiler (Pop).

```
Void  Pop(Pile **Spile, item *valeur)
{
  If VidePile(Spile) = = 1
     Printf("Erreur : Pile vide!");
  else
  {
     Pile *Pt = (*Spile)->suiv ;
     Valeur  = (*Spile)->info ;
     Free(*Spile), *Spile= NULL ;
     *Spile = Pt;
  }
}
```

Après avoir étudié les piles du point de vue statique et dynamique, ferons de même avec les files d'attente.

Une file d'attente statique est représentée par un tableau circulaire **Tab** et ses deux indices de parcours **tete** et **queue**, un pour indiquer la tête de la file et l'autre pour indiquer la queue de la file.

Quant à la file d'attente dynamique, elle est représentée sous forme d'une liste chaînée avec sa politique de gestion particulière.

```
/*Implémentation d'une file d'attente F statique*/
#define Maxfile 10              /* on suppose qu'il y ait  10 éléments*/
Typedef   struct  fila  fila ;  /*fila pour ne pas confondre avec le mot clé FILE Fichier*/
 Struct  fila     {             /* quoique FILE est écrit en majuscule */
                    Int tete ;
                    Int queue ;
                    Item  Tab[Maxfile] ; /* item type à définir*/
                  } ;
fila  F ;
```

QUELQUES OPERATIONS DE BASE

A. Procédure d'initialisation de la file d'attente.

```
Void  InitialiserFile(fila *F)
{
  F.tete = 0 ;
  F.queue = -1 ;        /*C'est une convention*/
}
```

B. Fonction pour vérifier si la file d'attente est vide.

```
Int VideFile(fila F)
{
   If ((F.tete = = 0) && (F.queue = = -1))
      Return 1;
    Return 0;
}
```

C. Fonction pour vérifier si la file d'attente est pleine.

```
Int pleineVide(fila F)
{
   If (((F.tete == 0) && (F.queue == Maxfile-1)) || ((F.tete == F.queue+1) && (F.queue != -1)))
      Return 1 ;
    Return 0 ;
}
```

D. Procédure pour enfiler une valeur à la file d'attente (Enqueue).

```
Void  EnqueueFile(fila  *F, item valeur)
{
  If FilePleine(F) == 1
      Printf(‘’ Erreur : file d’attente pleine !’’) ;
   Else
      {
         F.Queue = (F.Queue+1) % Maxfile ;
         F.Tab[F.queue] = valeur ;
      }
}
```

E. Procédure pour défiler une valeur de la file d'attente (Dequeue).

```
void  dequeueFile(fila *F, item *valeur)
{
   If FileVide(F) == 1
      Printf(''Erreur : File d'attente vide !'') ;
   Else
     {
        Valeur = F.Tab[F.tete] ;
        If F.tete == F.queue   /* cela veut dire que la file contient un elem*/
          {
            F.tete = 0 ;          /* donc elle devient vide*/
            F.queue = -1 ;
          }
        else /* La file contient plus d'un élément*/
             F.Tete = (F.Tete+1) % Maxfile;
     }
}
```

Sans tarder, on va étudier les files d'attente dynamiques sous forme de listes chaînées.

```
/*Implémentation d'une file d'attente F*/
    #include <stdlib.h>
    Typedef     struct  fila    {
                                Item  info ;
                                Struct  fila   *suiv ;
                                } fila;
    Fila   *F = NULL ;
```

NB : On remarque que c'est pratiquement la même implémentation qu'une pile et c'est leur politique de gestion qui fait la différence.

i. Fonction pour défiler une valeur de la file d'attente (En tête de file).

```
Item DequeueFile(fila **T_file)        /* T_file comme tête de file*/
{
   Item res ;
   If (*T_file != NULL)  /* la file d'attente n'est pas vide*/
    {
        Fila *tete = (*T_file)->suiv ;
        res = (*T_file)->info ;
        free(*T_file), *T_file = NULL ;
        *T_file = tete ;
     }
   Return res ;
}
```

Remarque : Il est plus judicieux d'utiliser une procédure pour régler le problème de la file d'attente vide !

ii. **Procédure pour défiler une valeur de la file d'attente.**

```
Void DequeueFile(fila **T_file, item *valeur) ;
{
  If (*T_file = = NULL)
    Printf(''Erreur : File d'attente vide !'') ;
  Else
    {
      Fila  *tete = (*T_file)->suiv ;
      Valeur = (*T_file)->info ;
      Free(*T_file), *T_file = NULL ;
      *T_file = tete ;
    }
}
```

iii. Procédure d'enfiler une valeur à la file d'attente (en queue de la file).

```
Void EnqueueFile(fila **T_file, item valeur)
{
   Fila *nouveaunode = malloc(Sizeof *nouveaunode) ;
   If  (nouveaunode != NULL) /* pour vérifier sa création*/
   {
        Nouveaunode->suiv = NULL ; /* Car il sera le dernier*/
        Nouveaunode->info = valeur ;
        If (*T_tete == NULL)   /* ie elle est vide*/
            *T_file = nouveaunode ;   /* ie elle devient singleton*/
        Else   /* ie elle n'est pas vide il faut la parcourir jusqu'à la fin «Q_file »*/
          {
                Fila   *Q_file = *T-file ;          /*Q_file comme Queue de la file*/
               While (Q_file->suiv   !=  NULL)
                     Q_file  =  Q_file->suiv ;
               Q_file->suiv  =  nouveaunode ;
            }
         }
        }
```

BIBLIOGRAPHIE

1- Langage C : Cours et références
Pierre NERZIC Mars 2003

2- Langage C : Support de cours
Patrick CORDE Mars 2006

3- Data structures and programming design
Robert L. Kruse

4- Algorithmique, structures de données et langage C
J.M. ENJALBERT Janvier 2005

5- An introduction to methodical programming
W. Findbay and D. Wattes

6- Apprendre à programmer
Algorithmes et conception objet.
C. Dabancourt