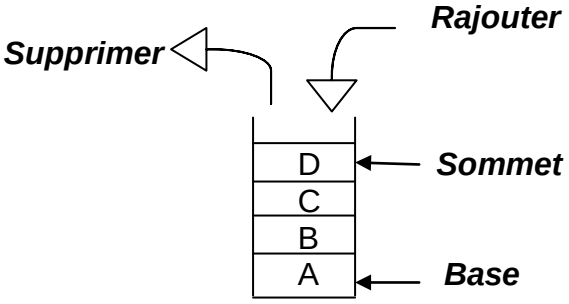
# Chapitre 6 : Les PILES et les FILES

## 6.1 Les PILES :

### 6.1.1 Définition :

Une pile est une structure de donnée contenant une lise d'éléments telle qu'un élément ne peut lui être retiré ou ajouté que par une seule extrémité appelé ***sommet*** de la pile. Une pile est utilisée pour stocker ***temporairement*** des données.

***Exemple :*** pile d'assiettes, pile de dossiers …




Un élément ne peut être rajouté ou supprimé qu'au sommet de la pile. Le dernier élément ajouté sera le premier retiré. C'est donc une structure de type ***LIFO*** *(« Last in, first out » = « dernier arrivé premier sorti »)*.
Les éléments sont retirés dans l'ordre inverse de celui de leur introduction.

### 6.1.2 Les opérations sur les piles :

Les opérations sur les piles sont les suivantes :

- ***Ajout*** d'un élément : l'action consistant à ajouter un nouvel élément au sommet de la pile s'appelle ***empiler*****,** puis mettre à jour ce sommet.
- ***Suppression*** d'un élément : l'action consistant à retirer un élément, celui qui est au sommet, s'appelle ***désempiler***, à condition que la ***pile ne soit pas vide***.
- ***Consultation*** : consulter le sommet de la pile en recopiant dans une variable l'élément du sommet sans affecter ni le contenu ni la position du sommet.

La manipulation des opérations sur les piles peut dépendre du type de la représentation de la pile : contiguë ou chaînée.

### 6.1.3 Représentation contiguë et chainée :

#### 6.1.3.1 Implémentation d'une pile par un tableau

On peut représenter une pile par un tableau (contiguë), la déclaration de la pile doit alors tenir compte de la taille maximale. C'est-à-dire si le tableau est plein, il n'est pas possible de rajouter d'éléments.

Il est donc nécessaire avant de rajouter un élément dans la pile de tester si la ***pile n'est pas pleine***.

| A | B | C | D | ….. | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | | | | | | taille max |
| Base | | | Sommet | | | | | | |

### 6.1.3.2 Implémentation d'une pile par une liste chainée

Dans une pile représentée par une liste la suppression et l'insertion se font en tête de la liste.

### 6.1.3.3 Opérations de manipulation des deux représentations

**Représentation contigüe**

**1) Déclaration**

```
#define max 100
typedef int typelem;
typedef struct { typelem T[max];
                 int sommet;
               } pile;

pile p;
```

**2) Initialisation**

```
pile  initpile ( )
{ pile p ;
  p.sommet=-1;
  return(p) ;
}
```

**3) Test si la pile est vide**

```
 int  pilevide(pile p)
{ if (p.sommet==-1) return(1) ;
 else return(0);
}
```

**4) Test si la pile est pleine**

```
int pilepleine (pile p)
{ if (p.sommet==max-1) return(1) ;
  else return(0) ;
}
```

**5) Consultation du sommet de la pile**

```
typelem  sommetpile(pile p)
{ typelem  x ;
   x= p.T[p.sommet];
   return(x) ;
}
```

**6) Ajout d'un element**

```
void empiler(pile *p, typelem x)
{  (*p).sommet++;
   (*p).T[(*p).sommet] = x;
}
```

**Représentation chaînée**

**1) Déclaration**

```
typedef int typelem ;
typedef struct no *pile;
 typedef struct no
        { typelem valeur ;
          pile svt ;
        } nœud ;
 pile p;
```

**2) Initialisation**

```
 pile  initpile ( )
{ pile p ;
  p=NULL;
  return(p) ;
}
```

**3) Test si la pile est vide**

```
 int  pilevide(pile p)
{ if (p==NULL) return(1);
  else return(0);
}
```

N'existe pas

**5) Consultation du sommet de la pile**

```
typelem  sommetpile(pile p)
{ typelem  x ;
   x=p->valeur ;
   return(x) ;
 }
```

**6) Ajout d'un element**

```
void empiler(pile *p, typelem x)
{ pile temp = créer_noeud() ;
  temp ->valeur = x ;
  temp->svt=*p ; *p=temp ;
}
```

### 7) Suppression d'un élément

```
void désempiler(pile *p, typelem *x)
{
   *x = (*p).T[(*p).sommet];
   (*p).sommet - -;
}
```

```
void désempiler(pile *p, typelem *x)
{ pile temp ;
   *x = (*p) ->valeur  ;
   temp=*p ;
   *p=(*p)->svt; free(temp) ;
 }
```

**Remarque :**
Si le mode de représentation n'est pas spécifié on utilisera les fonctions précédemment définies dans leur généralité comme suit :

*p=initpile() ;*
*empiler(&p,x) ;*
*désempiler(&p,&x) ;*
*x=sommetpile(p) ;*
*if (!pilevide(p)) …*

p étant déclaré : *pile p ;* *Les fonctions sont alors considérées comme prédéfinies.*

### 6.1.4 Transformation des expressions :

#### 6.1.4.1 Présentation du problème

Une utilisation courante des piles est l'élaboration par le compilateur d'une forme intermédiaire de l'expression. Après l'analyse syntaxique et lexicale, l'expression est traduite en une forme intermédiaire plus facilement évaluable.

Soit l'expression : A + B, son évaluation ne peut être faite immédiatement lors de la rencontre d'un opérateur car le 2ème opérande n'est pas encore connu par la machine. Par contre si l'expression pouvait être écrite sous la forme AB+ alors elle serait directement évaluable car les deux opérandes sont connus avant l'opérateur.
La notation **< *Opérande*> < *Opérateur*> < *Opérande*>** est dite INFIXE.
La transformation en autre représentation plus facilement évaluable est dite POSTFIXE ou PLONAISE SUFFIXE a qui a la forme :

**< *Opérande Gauche* > < *Opérande Droit* > < *Opérateur*>**

#### 6.1.4.2 Transformation des expressions Infixées en Postfixées

***Exemple :***

(A+B)*3 ⟶ AB+3*
A+B*3 ⟶ AB3*+
(A≤B) and (not C) ⟶ AB≤C not and
(A+B)-(C*D)/E ⟶ AB+CD*E/-

Pour transformer une expression Q Infixée en expression Postfixée il faut lui appliquer un certain nombre de règles.
Q étant donnée, nous utilisons deux piles :
- une pile P (des opérateurs) pour les traitements intermédiaires,
- et une pile R qui contiendra le résultat final en fin de traitement.

***Algorithme de transformation :***

1- Ecrire l'expression Q avec ***')'*** à sa fin
2- Initialiser la pile P avec ***'('***
3- **Début** boucle : *lire* un élément de Q (progresser de gauche à droite)
    4- Si ***opérande*** le mettre dans la pile R
    5- Si ***'('*** la mettre dans la pile p
    6- Si ***opérateur Opt***
        - désempiler P et mettre dans R tous les opérateurs de priorité **>** à ***Opt*** jusqu'à '('
        - empiler ***Opt*** dans P
    7- Si ***')'*** - désempiler p et mettre dans R tous les opérateurs jusqu'à ***'('***
        - supprimer ***'('*** de P
    8- Recommencer 3, 4, 5, 6 et 7 jusqu'à la fin de Q
9- Fin boucle
**10- Fin.**

***Exemple d'application :***

| Q | P | R |
|---|---|---|
| A | ( | A |
| + | ( | A |
| B | (+ | AB |
| - | (+ | AB |
| C | (+ - | ABC |
| * | (+ - | ABC |
| D | (+ - * | ABCD |
| ) | (+ - * | ABCD * - + |

### 6.1.4.3 Evaluation des expressions Postfixées

***Algorithme :***

1- Ecrire l'expression R avec ***')'*** à sa fin
2- **Début** boucle : *lire* un élément de R (progresser de gauche à droite)
    3- Si ***opérande*** le mettre dans la pile p
    4- Si ***opérateur Opt***
        - désempiler les 2 premiers opérandes ***Op1*** et ***Op2*** de P
        - évaluer ***Op2 Opt Op1*** (faire attention à l'ordre)
        - empiler le résultat de l'évaluation intermédiaire dans P
    5- Recommencer 2, 3, et 4 jusqu'à la fin de R
6- Fin boucle
**7- Fin.**

***Exemple :***

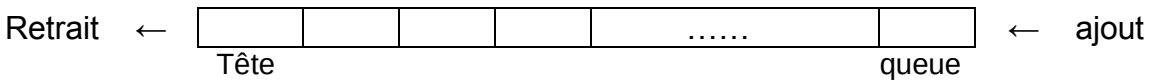
| | |
|---|---|
| R | P |
| A | A |
| B | AB |
| C | ABC |
| D | ABCD |
| * | AB(C*D) |
| - | A+(B - (C*D)) |
| + | (A+(B – (C*D))) |
| ) | |

***Remarque*** : Les parenthèses, contenues dans la pile P ne servent qu'à montrer dans quel ordre sont effectuées les opérations.

## 6.2 Les FILES :

### 6.2.1 Définition

La notion de file est très courante dans notre vie pratique. Sa principale caractéristique est le « 1er arrivé 1er servi » ou ***FiFo « first in first out »***.




Une file permet deux accès : un accès appelé tête désigne le 1er élément à supprimer et un accès appelé queue désigne le dernier élément ajouté.
Comme pour les piles une file peut être représenté de 2 manières : ***contiguë*** ou ***chaînée.***

### 6.2.2 Opérations de manipulation des deux représentations

**Représentation contigüe**

**Représentation chaînée**

**1) Déclaration**

```
#define max 100
typedef int typelem;
typedef struct { typelem T[max];
                 int queue, tete;
               } file;

file F;
```

```
typedef int typelem ;
typedef struct no *file;
typedef struct no
    { typelem valeur ;
      file svt ;
    } nœud ;
file F;
```

**2) Initialisation**

```
file  initfile ( )
{ file F ;
  F.queue=-1;  F.tete=-1;
  return(F) ;
}
```

```
file  initfile ( )
 { file F ;
   F=NULL;
   return(F) ;
 }
```

### 3) Test si la file est vide

```
 int  filevide(file F)
{ if (F.tete==-1) return(1) ;
 else return(0);
}
```

```
 int  filevide(file F)
{ if (F==NULL) return(1);
   else return(0);
 }
```

### 4) Test si la file est pleine

```
int filepleine (file F)
{ if (F.queue==max) return(1) ;
  else return(0) ;
}
```

N'existe pas

### 5) Consultation de la tête de file

```
typelem  sommetfile(file F)
{ typelem  x ;
   x= F.T[F.tete];
   return(x) ;
}
```

```
typelem  sommetfile(file F)
  { typelem  x ;
    x=F->valeur ;
    return(x) ;
  }
```

### 6) Consultation de la queue de file

```
typelem  queuefile(file F)
{
   typelem  x ;
   x= F.T[F.queue];
   return(x) ;
}
```

```
typelem  queuefile(file F)
{ typelem  x ;
  file  queue=dernierelem(F);
   x=queue->valeur ;
   return(x) ;
}
```

### 7) Ajout d'un element

```
void emfiler(file *F, typelem x)
{  (*F).queue++;
   (*F).T[(*F).queue] = x;
   If (filevide(*F)) (*F).tete++;
}
```

```
void emfiler(file *F, typelem x)
{ file queue, temp=créer_noeud() ;
  temp ->valeur = x ;
  temp->svt=NULL ;
  if (filevide(*F)) *F=temp;
  else {queue=dernierelem(*F);
         queue->svt=temp;
         queue=temp;  }
  }
```

### 8) Suppression d'un élément

```
void désemfiler(file *F, typelem *x)
{  int i ;
   *x = (*F).T[(*F).tete];
   for(i=(*F).tete;i<(*F).queue;i++)
       (*F).T[i]=(*F).T[i+1];
   (*F).queue--;
}
```

```
void désemfiler(file *F, typelem *x)
{ file temp ;
   *x = (*F) ->valeur  ;
    temp=*F;
   *F=(*F)->svt;
   free(temp) ;
```