Chapitre 4 : Les Piles

I. DEFINITIONS

I.1 Définition d'une Pile : Une pile est une collection d'éléments de même type dont le seul élément directement accessible est le *dernier introduit* dans la pile.

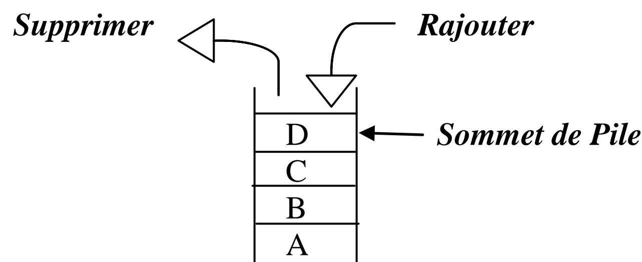
Le dernier élément introduit s'appelle ***sommet de pile*** et joue un rôle particulier, puisque tout accès à la pile se fait par cet élément. En effet, l'accès à un élément quelconque de la pile nécessite le retrait de tous les éléments qui le précèdent. L'accès aux éléments de la pile se fait donc de manière séquentielle. Les éléments sont retirés dans l'ordre inverse de celui de leur introduction.

Une pile est une structure de donnée de type ***LIFO***
(« Last In, First Out » = « dernier arrivé premier sorti »).

Remarque : l'état de la pile ne doit pas être changé suite à une opération de consultation : il faut remettre en place tous les éléments retirés.

Exemples : pile d'assiettes, pile de dossiers, …

Schématiquement, une pile est représentée comme suit :



I.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 sans affecter ni le contenu ni la position du sommet.

I.3. Utilisation des Piles dans les Applications Informatiques

Les piles sont des structures de données utilisées dans divers domaines de l'informatique. Parmi lesquels, nous citons :

- ***Par un compilateur, lors des appels de fonctions*** : Avant de se brancher vers la fonction appelée, le compilateur sauvegarde l'adresse de retour et les variables du programme appelant, afin de les restituer au retour de la fonction. De plus, les retours se font dans le sens inverse des appels, ce qui coïncide bien avec la structure LIFO de la pile.
- ***Par un compilateur, pour l'évaluation des expressions arithmétiques :*** ce dernier point est détaillé dans ce qui suit.
- ***Par un système d'exploitation, pour la gestion des interruptions :*** pour la *s*auvegarde et la restitution de contexte de processus.
- ***Dans la théorie des langages :*** pour l'implémentation des automates à piles[1].

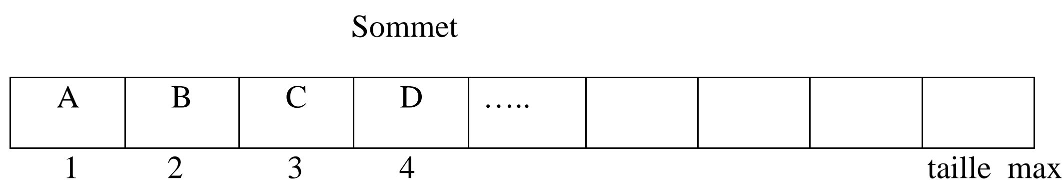
II. REPRESENTATION DES PILES EN MEMOIRE :

Il existe deux représentations (implémentations) des Piles en mémoire : contiguë et chaînée. La manipulation des opérations sur les piles dépend du type de la représentation.

II.1 Représentation d'une Pile par un Tableau

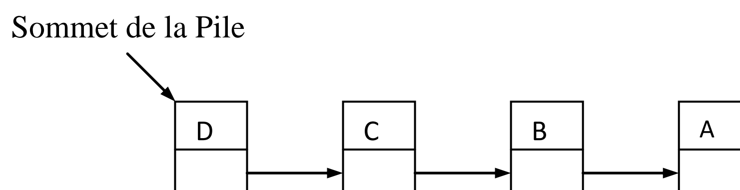
On peut représenter une pile par un tableau (forme contiguë), alloué d'une manière statique ou dynamique. La déclaration de la pile doit alors tenir compte de la taille maximale, c'est-à-dire si le tableau est plein, il n'est pas possible de rajouter des éléments.

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



II.2 Représentation d'une Pile par une Liste Chainée

Une pile peut être représentée par une liste chainée où l'ajout ou le retrait d'un élément se fait toujours par la tête de la liste, appelée dans ce cas *sommet de pile*.



III. PRIMITIVES DE MANIPULATION DES PILES

[1] Les automates a pile sont les reconnaisseurs des langages algébriques.

Ces primitives correspondent à des fonctions de base de manipulation de la pile. Leur écriture dépend de l'implémentation de la pile (contigüe ou chainée) et du type des éléments de la pile.

InitPile : permet d'initialiser la pile

PileVide : permet de vérifier si la pile est vide.

PilePleine : permet de vérifier si la pile est pleine (dans le cas d'une représentation contiguë).

SommetPile : retourne l'élément du sommet dans une variable, sans le désempiler.

Empiler : permet d'ajouter un élément à la pile (au dessus de l'élément du sommet).

Désempiler : permet de retirer (supprimer) un élément de la pile (celui du sommet) et retourne sa valeur dans une variable.

III.1 Opérations de Manipulation des Piles dans les deux Représentations

Dans ce cours, on considère une pile d'éléments de type TypeElement. On écrira les primitives dans les deux types d'implémentation (contigüe et chainée).

A) Représentation Contigüe (Cas d'Allocation Statique)

1) Déclaration

```
constante   max 100

type Pile = Enregistrement

                T : tableau [max] TypeElement ;

                Sommet : entier ;

            FinEnre

variable p : Pile ;
```

2) <u>Initialisation</u>

```
Fonction  InitPile ( ) : Pile

Variable p : Pile

Debut

   p.Sommet ←0;

   retourner (p);

Fin
```

3) Test si la pile est vide

```
Fonction PileVide (E/ p : Pile) : booleen
  Debut  si (p.sommet=0) alors retourner (vrai) ;
                                   sinon retourner (faux) ;
         fsi
  Fin
```

4) Test si la pile est pleine

```
Fonction PilePleine (E/ p : Pile) : booleen
Debut
  si (p.sommet=max) alors retourner (vrai) ;
                              sinon retourner (faux) ;
  fsi
Fin
```

5) Consultation du sommet de la pile

```
Fonction SommetPile(E/ p : Pile) : TypeElement
variable x : TypeElement ;
Debut
  x← p.T[p.sommet];
  retourner (x) ;
Fin
```

6) Ajout d'un element

```
Procedure Empiler (E/S p: Pile, E/ x : TypeElement)
Debut
  p.sommet ←p.sommet+1 ;
  p.T[p.sommet]←x ;
Fin
```

7) Suppression d'un élément

```
Procedure Désempiler (E/S p: Pile, E/S x : TypeElement)
Debut
    x← p.T[p.sommet] ;
    p.sommet ←p.sommet-1 ;
Fin
```

B) Représentation Chainée

1) Déclaration

```
type Element = Enregistrement
                    info : TypeElement ;
                    suivant : ^ Element;
               FinEnreg
type Pile : ^Element ;
variable p : Pile;
```

2) Initialisation

```
Fonction InitPile ( ) :  Pile
Debut
  retourner (nil)
Fin
```

3) Test si la pile est vide

```
Fonction PileVide (E/ p: Pile) : booleen
Debut
  si p=nil alors retourner (vrai)
          sinon retourner (faux) ;
  fsi
Fin
```

4) Consultation du sommet de la pile

```
Fonction SommetPile (E/ p : Pile) : TypeElement
variable   x : TypeElement
 Debut
   x← ^p.info;
   retourner (x) ;
Fin
```

5) Ajout d'un element

```
Procedure Empiler (E/S p : Pile , E/ x : TypeElement)
variable temp : Pile ;
Debut
   temp←Allouer(TailleDe(Pile)) ;
   ^temp.inf←x
   ^temp.suivant←p ;
   p←temp ;
Fin
```

6) Suppression d'un élément

```
Procedure Desempiler (E/S p : Pile , E/S x : TypeElement)
variable temp : Pile ;
Debut
   x← ^p.info ;
   temp←p ;
   p←^p.suivant ;
   liberer (temp)
Fin
```

Exemple 1

Ecrire une fonction qui lit des entiers et les stocke dans une pile. L'arrêt se fera dès la rencontre de la valeur –1.

```
Fonction StockerPile( ) : Pile
Variable p : Pile; x : entier;
Debut
   P←InitPile();
   Lire(x);
   Tantque( x≠-1) faire
                              Empiler(p, x) ;
                              Lire(x) ;
                         fait ;
    Retourner (p) ;
Fin;
```

Exemple 2

Soit une pile d'entiers P et une valeur val donnée. Ecrire une fonction qui recherche la valeur val dans la pile.

```
Fonction Recherche(E/ P : Pile ; E/ val :entier ) : booléen
Variable R : Pile ; trouv : booléen ;
Debut
  R←InitPile() ;
  trouv←faux ;
  Tantque((PileVide(P)=faux)et(trouv=faux))
  faire
          Si (SommetPile(P)≠val) alors     Debut
                                           Desempiler(P,x) ;
                                           Empiler(R, x) ;
                                           Fin ;
                                  Sinon B←vrai ;
          Fsi

  fait ;
  Tantque (PileVide(R)=faux) faire
                                  Desempiler(R, x) ;
                                  Empiler(p, x) ;
                              fait ;
  Retourner (trouv) ;
Fin;
```

IV. UTILISATION DES PILES DANS L'EVALUATION DES EXPRESSIONS ARITHMETIQUES

Une utilisation courante des piles est l'élaboration par le compilateur d'une forme intermédiaire de l'expression à évaluer. Après l'analyse lexicale et syntaxique, 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.

L'autre représentation plus facilement évaluable est dite POSTFIXE ou POLONAISE SUFFIXE. Elle a la forme :

< Opérande Gauche > < Opérande Droit > < Opérateur>

Exemples de Transformation:

Forme Infixée	Forme Postfixée
25 -11	25 11-
25 + 11*5	25 11 5 * +
(25 + 11) *5	25 11 + 5 *

L'évaluation des expressions n'est pas directe puisque dès la rencontre d'un opérateur on ne sait pas quelle opération doit-on évaluer d'abord ? Ceci dépend des priorités des opérateurs dans le cas d'une expression non parenthésée ou de la position des parenthèses dans l'expression dans le cas d'une expression parenthésée.

IV.1 Algorithme de Transformation d'une Forme Infixée en une Forme Postfixée

L'algorithme utilise **deux piles P et R**. Il reçoit comme donnée l'expression arithmétique déjà analysée, rangée dans un vecteur T de taille *Taille_Expression*. Il utilise les fonctions suivantes supposées définies.

Operateur(x) : détermine si x est un opérateur : + (addition), - (soustraction), *(multiplication), / (division) et % (reste de la division entière).

Operande(x) : détermine si x est un opérande. Un opérande peut être une suite de symboles représentant un entier ou un réel.

Priorite(x) : retourne la priorité de l'opérateur x.

Priorité(+)=Priorité(-) < Priorité(*)=Priorité(/)=Priorité(%)

Operation (x1, x2, operateur) : effectue l'opération (x1 operateur x2) et retourne le résultat.

Algorithme de Transformation

```
Debut

P←InitPile();  R←InitPile(); ;

Pour i ←1 à Taille_Expression

Faire

   Si (Operande (T[i])  alors   Empiler (R, T[i]) ; fsi ;

   Si (T[i]=  '('  ) alors   Empiler (P, T[i]) ; fsi ;

   Si (Operateur (T[i]) alors

            Tant que  (non PileVide(P) et Operateur(SommetPile(P)) et

                          (Priorite (T[i]) <= Priorite (SommetPile(P)))

                 Faire   Desempiler (P, x) ;  Empiler   (R, x) ; Fait

            Empiler (P, T[i] );   /* à la sortie de la boucle */

  Fsi ;

   Si  (T[i] = ')'  )  alors

                Tant que  (non PileVide(P) et (SommetPile (P) ≠ '(' )

                     Faire  Desempiler (P, x) ;  Empiler   (R, x) ; fait

                Desempiler (P, x) ;

  fsi

Fait

Tant que  (non PileVide(R))

                     Faire Desempiler (R, x) ;  Empiler   (P, x) ; fait

Fin
```

A la fin de cet algorithme, la pile P contiendra la forme postfixée de l'expression.
Exemple : Soit l'expression A + B – C * D

Action	Etat de la Pile R	Etat de la Pile P
Lire (A)	A	
Lire (+)	A	+
Lire (B)	AB	+

Lire (-) Lire (C) Lire (*) Lire (D) Fin de l'Expression	AB+ AB+C AB+C AB+CD	- - -* -* -*DC+BA

A présent, la forme postfixée contenue dans la pile P peut être alors évaluée en utilisant l'algorithme ci-dessous.

IV.2 Algorithme d'Evaluation d'une Forme Postfixée

L'algorithme utilise **deux piles P et R**. P contient la forme postfixée de l'expression et R étant initialement vide.

Algorithme d'Evaluation

```
Debut
  R ←InitPile(); ;
  Tant que  (non PileVide(P))
        Faire
           Desempiler (P, x) ;
           si (Operande (x))  alors Empiler (R, x) ;
                              sinon  /* c'est un opérateur */
                                faire   Desempiler (R, x1) ;
                                        Desempiler (R, x2) ;
                                        Res  = Operation (x2, x1, x) ;
                                        Empiler (R, Res) ;
                                fait
           fsi
        fait
Fin
```

Le résultat Final se trouvera dans la pile R et la pile P devient vide.

Exemple : Soit l'expression A + B – C * D. Sa forme postfixée contenue dans la pile P est : -*DC+BA. Son évaluation se fait selon les étapes suivantes :

Action	Etat de la Pile P	Etat de la Pile R
Initialement	-*DC+BA	
Depiler (A)	-*DC+B	A
Depiler(B)	-*DC+	AB
Depiler(+)	-*DC	(A+B)
Depiler(C)	-*D	(A+B)C
Depiler(D)	-*	(A+B)CD
Depiler(*)	-	(A+B)(C*D)
Depiler(-)		((A+B)-(C*D))

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