# Chapitre IV : Les piles

## IV.1. Définition

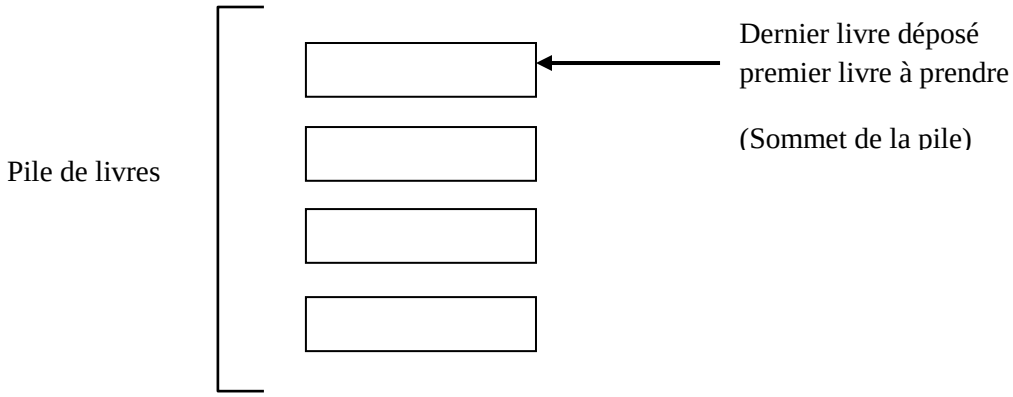
Une pile est une structure de données dynamique à accès séquentiel formée de ***n*** éléments de même type.

Pile = < E1, E2,…, En > ; (**n ≥ 0**)

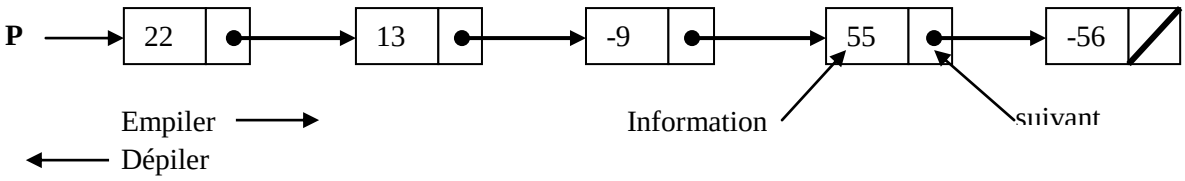
Elle est connue par une stratégie appelé **LIFO** (Last In First Out) qui veut dire « dernier arrivé-premier servii ». En d'autres termes : le dernier élément rajouté à la pile sera le premier à en sortir.

**Exemple**

Si vous déposez des livres l'un sur l'autre vous formez une pile de livres. Le livre qui est alors accessible sur la pile est celui du **sommet.** C'est le dernier que vous avez rajouté à la pile.




## IV .2. Réalisation d'une pile à l'aide d'une liste chaînée




- P est le sommet de la pile et la tête de la liste

## IV.2.1. Déclaration

**Syntaxe** ( en algorithmique)

```
type    type_pile= ↑ nœud  ;
type    nœud = enregistrement
          | info : type_info ;
          | suivant : type_pile ;
          |fin ;
```

Après, on déclare une variable pile:

**var** identificateur_pile : type_pile ; (sommet de pile)

**Exemple**

```
type    pile= ↑ nœud   ;
type    nœud = enregistrement
          | info :entier ;
          | suivant : pile ;
          |fin ;
var   p :pile ;
```

## IV.3. Les opérations sur les piles

## IV.3.1. Créer / détruire une pile

```
Procedure    vider_pile (var p:type_pile) ;
  debut
  |    p ←Nil ;
  |fin;
```

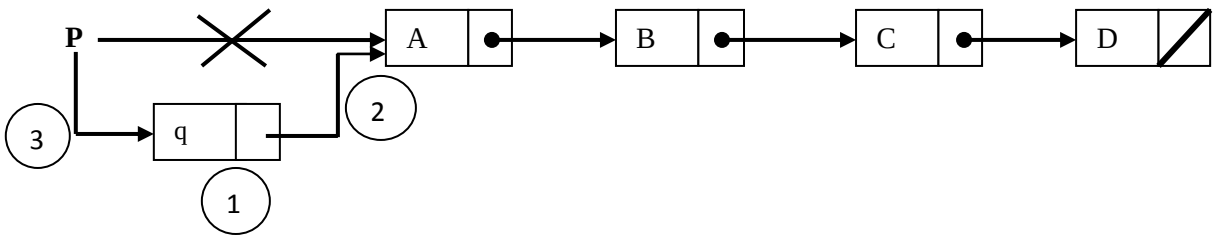
## IV.3.2. Vérifier si la pile est vide

```
fonction   pile_vide ( p:type_pile) :booleen ;
  debut
  |   Si (p =Nil) alors
  |   | pile_vide ← vrai
  |   sinon
  |   |  pile_vide ← faux ;
  |   |finsi ;
  |fin;
```

## IV.3.3. Prendre la valeur du sommet de la pile

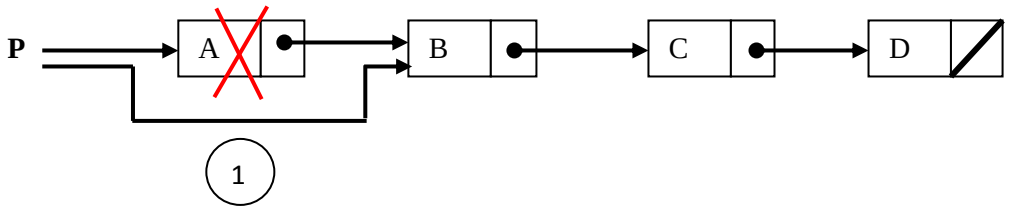
```
fonction   sommet ( p:type_pile) : type_info ;
  debut
  |   Si (pile_vide(p)=faux) alors
  |   | sommet ← p↑.info
  |   sinon
  |   |  ecrire ('pile vide') ;
  |   |finsi ;
  |fin;
```

## IV.3.4. Ajouter une valeur au sommet de la pile



```
procedure   empiler ( var p:type_pile ; E : type_info) ;
 var   q :type_pile ;
   debut
   |        allouer(q) ;
   |        avec q↑ faire
   |        | info ←E;
   |        | suivant ←p ;
   |        |finavec ;
   |       p ← q ;
   |fin;
```

## IV.3.5. Supprimer un élément de la pile (le sommet)




```
procedure   depiler ( var p:type_pile) ;
   debut
   |   Si (pile_vide(p)=faux) alors
   |   | p ← p↑.suivant
   |   sinon
   |   |  ecrire ('pile vide')  ;
   |   |finsi ;
   |fin;
```

**Exercice**

Ecrire un algorithme qui remplit une pile de 10 valeurs entières puis de chercher et d'afficher le min dans cette pile.

```
algorithme     chercher_min ;
type     pile= ↑ nœud   ;
type     nœud  = enregistrement
          |  info :entier ;
          |  suivant : pile ;
          |fin ;
var   p :pile ; E, Min, i : entier ;
debut
|         vider_pile (p) ;
|         pour   i     allantde     1     jusqu'à   10   faire
|         |     lire (E) ;
|         |     empiler (p,E) ;
|         | finpour;
|         Min ← sommet(p) ;
|         tantque    (pile_vide(p)=faux)   faire
|         |    depiler (p) ;
|         |     Si (Min> sommet(p)) alors
|         |     |   Min← sommet(p) ;
|         |     |finsi ;
|         | fintantque;
|          ecrire (Min) ;
|fin.
```

## IV.4. Les applications des piles

### IV.4.1. Calcul arithmétique

Une application courante des piles se fait dans le calcul arithmétique: l'ordre dans la pile permet d'éviter l'usage des parenthèses. La notation ***postfixée*** consiste à placer les opérandes devant l'opérateur. La notation ***infixée*** (parenthésée) consiste à entourer les opérateurs par leurs opérandes. Les parenthèses sont nécessaires uniquement en notation infixée. Certaines règles permettent d'en réduire le nombre (priorité de la multiplication par rapport à l'addition), en cas d'opérations unaires représentées par un caractère spécial (-, !,...).
Les notations ***préfixée*** et ***postfixée*** sont d'un emploi plus facile puisqu'on sait immédiatement combien d'opérandes il faut rechercher.

Détaillons, ci-dessous, la saisie et l'évaluation d'une expression postfixée:

La notation usuelle, comme (3 + 5) * 2, est dite infixée. Son défaut est de nécessiter l'utilisation de parenthèses pour éviter toute ambiguïté (ici, avec 3 + (5 * 2)). Pour éviter le parenthésage, il est possible de transformer une expression infixée en une expression postfixée en faisant "glisser" les opérateurs arithmétiques à la suite des expressions auxquelles ils s'appliquent.

**Exemple**

- (3 + 5) * 2 s'écrira en notation postfixée : 3 5 + 2 * alors que 3 + (5 * 2) s'écrira: 3 5 2 * +
- A * B/C. En notation postfixée est: AB * C/.

On voit que la multiplication vient immédiatement après ses deux opérandes A et B. Imaginons maintenant que A * B est calculé et stocké dans T. Alors la division / vient juste après les deux arguments T et C.

***Forme infixée***: A/B ** C + D * E - A * C

***Forme postfixée***: ABC ** /DE * + AC * -

**Exemple**

Considérons l'expression postfixée suivante:
6 5 2 3 + 8 * + 3 + ****Pseudo-code de l'algorithme :***

1. Initialiser la pile à vide;
2. While (ce n'st pas la fin de l'expression postfixée)
3. {
4. Prendre l'item prochain de l'expression postfixée;
5. Si (item est une valeur)
6. empiler;
7. Sinon si (item operateur binaire)
8. {
9. dépiler dans x;
10. dépiler dans y;
11. effectuer y operateur x;
12. empiler le résultat obtenu;
13. }
14. sinon si (item opérateur unaire)
15. {
16. dépiler dans x;
17. effectuer opérateur(x);
18. empiler le résultat obtenu;
19. }
20. }