

## Chapitre 6 : Les Piles

### 1. Définition

On appelle pile un ensemble de données, de taille variables, éventuellement nul, sur lequel on peut effectuer les opérations suivantes :

**créer pile** : elle permet de créer une pile vide

**vide** : elle permet de voir si une pile est vide ou non ?

**empiler** : elle permet d'ajouter un élément de type T à la pile

**dépiler** : elle permet de supprimer un élément de la pile

**dernier** : elle retourne(ou rend) le dernier élément empilé et non encore dépilé.

En générale, les opérations applicables sur une structure de donnée sont divisées en trois catégories :

**Les opérations de création** (souvent une seule opération) : elles permettent de créer une structure SD.

Illustration : SD pile : créer pile

**Les opérations de consultation** : elles permettent de fournir des renseignements sur une SD

Illustration : SD pile : vide et dernier. Une opération de consultation laisse la SD inchangée.

**Les opérations de modification** : elles agissent sur la SD. Illustration : empiler et dépiler.

### OPERATIONS ILLEGALES :

Certaines opérations définies sur une SD exigent des préconditions: on ne peut pas appliquer systématiquement ces opérations sur une SD.

Illustration :

dépiler exige que la pile soit non vide de même

Idem pour l'opération dernier.

### 2. Propriétés

Les opérations définies sur la SD pile obéissent aux propriétés suivantes :

(P1) creer\_pile permet la création d'une pile vide.

(P2) si on ajoute un élément (en utilisant empiler) à une pile alors la pile résultante est non vide.

(P3) un empilement suivi immédiatement d'un dépilement laisse la pile initiale inchangée.

(P4) On récupère les éléments d'une pile dans l'ordre inverse ou on les a mis (empiler).

(P5) Un dépilement suivi immédiatement de l'empilement de l'élément dépilé laisse la pile inchangée.

**REMARQUE** : Ces propriétés permettent de cerner la sémantique (comportement ou rôle) des opérations applicables sur la SD pile.

**En conclusion** : la SD pile obéit à la loi LIFO (Last In, First Out) ou encore DRPS (Dernier Rentré, Premier Sortie).

### 3. Représentation physique d'une Structure de donnée

Pour pouvoir matérialiser (concrétiser, réaliser ou implémenter) une SD on distingue deux types de représentation :

**Représentation chaînée** : les éléments d'une SD sont placés à des endroits quelconques (bien entendu dans la MC) mais chaînés (ou reliés). Idée : pointeur.

**Représentation contiguë** : les éléments d'un SD sont rangés (placés) dans un espace contiguë. Idée : tableau.

### 3.1 Représentation chaînée de la SD pile

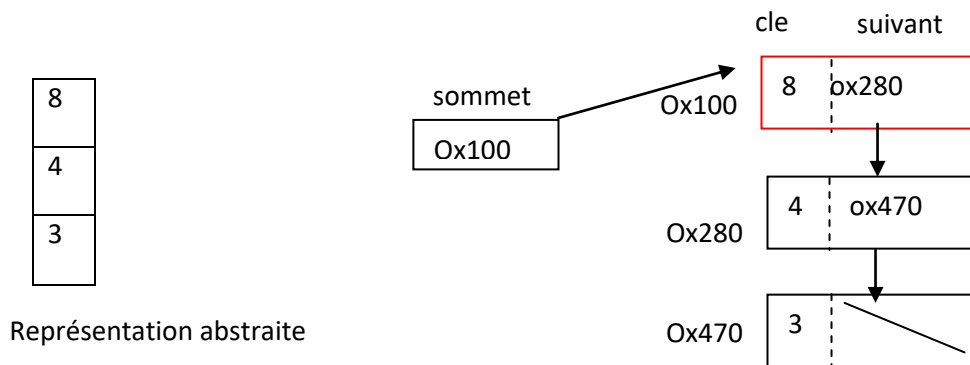
Utilisation des pointeurs pour relier explicitement les éléments formant la SD.

La représentation chaînée comporte plusieurs nœuds. Chaque nœud regroupe les champs suivants :

Le champ « cle » permet de mémoriser un élément de la pile.

Le champ « suivant » permet de pointer sur l'élément précédemment empilé.

La variable « sommet » permet de pointer ou de repérer l'élément au sommet de la pile.



### 3.2 Représentation contiguë

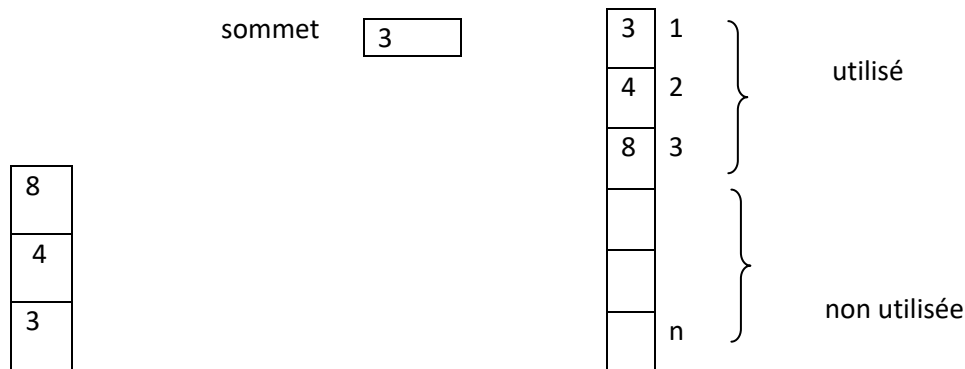
Pour pouvoir représenter la structure de données Pile d'une façon contiguë, on fait appel à la notion du tableau.

Puisque le nombre d'éléments d'un tableau doit être fixé avant l'utilisation, on aura deux parties :

**Partie utilisée** : elle est comprise entre 1 et sommet.

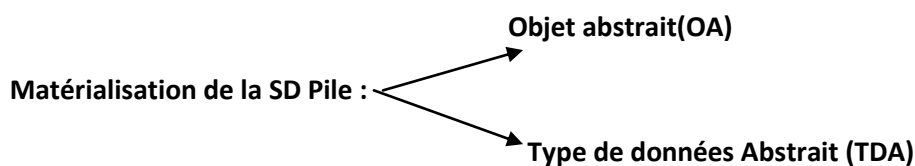
**Partie non utilisée** : elle est comprise entre sommet+1 et n.

Avec sommet est indice compris entre 0 et n.



Représentation abstraite

## 4. Matérialisation de la SD Pile



- Objet abstrait(OA) → un seul exemplaire (ici une seule pile) → unique → implicite.
- Type de données Abstrait (TDA) → plusieurs exemplaires → il faut mentionner ou rendre explicite l'exemplaire courant.

## 4.1 Structure de données Pile comme Objet Abstrait

L'interface pile regroupe des services exportées par cette interface. Chaque service correspond à une opération applicable sur la SD (ici la SD Pile). Et il est fourni sous forme d'un sous-programme

Opérations applicable sur la SD Pile définies

- Opération de création :(procédure ou fonction)
- Opération de modification : procédure
- Opération de consultation : fonction

<pre> /*représentation physique*/ <b>Types</b> Cellule = Struct cle : entier Suiv : ^Cellule FinStruct <b>Var</b> sommet : ^Cellule statique  <b>Procédure</b> creer_pile ( ) <b>Début</b>     sommet ← NIL  <b>Fin proc</b>  <b>Fonction</b> vide ( ) : boolean <b>Debut</b>     <b>Si</b> (sommet=Nil) <b>alors</b>         vide ← vrai     Sinon vide ← faux     <b>finsi</b> <b>fin Fn</b>  <b>Fonction</b> dernier() : entier <b>Debut</b>     assure( ! vide())     dernier ← sommet^.cle <b>Fin Fn</b>  <b>Procédure</b> empiler (info :entier) <b>Var</b>     p : ^Cellule <b>Debut</b>      Allouer (p)     p^.cle ← info     p^.suiv ← sommet     /*mettre à jour sommet*/     sommet ← p <b>Fin Proc</b> </pre>	<pre> <b>Procédure</b> depiler( ) <b>Var</b>     q: ^Cellule <b>Debut</b>     assure( !vide())     q ← sommet     sommet ← sommet^.suiv     liberer (q) <b>Fin Proc</b>  /* Exemple d'utilisation */ <b>Algorithme</b> Exemple <b>Var</b>     i : entier <b>Début</b>     créer_Pile()     assure(vide())     /* l'instruction assert en C et en Java*/      <b>pour</b> i de 1 à 10 <b>faire</b>         empiler(i)     <b>fin pour</b>      assure(!vide())     <b>pour</b> i de 1 à 10 <b>faire</b>         ecrire (dernier())         depiler()     <b>fin pour</b>     assure(vide()) <b>Fin Algo</b> </pre>
--	---

## 4.2 Structure de données Pile comme type de donnée abstrait

Dans ce paragraphe, on va concrétiser la SD pile sous forme d'un TDA capable de gérer plusieurs exemplaires de la SD pile et non pas un seul exemplaire (objet abstrait voir 4.1).

```
/*représentation physique*/  
Types  
  Cellule = Struct  
  cle : entier  
  Suiv : ^Cellule  
  FinStruct  
  
Fonction creer_pile ( ) :^Cellule  
Début  
  creer_pile ← NIL  
Fin Fn  
  
Fonction vide (P :^Cellule): boolean  
Debut  
  Vide ← (P=NIL)  
fin Fn  
  
Fonction dernier(P :^Cellule): entier  
Debut  
  assure( ! vide(P))  
  dernier ← P^.cle  
fin Fn  
  
Procedure empiler (info :entier, var A :^Cellule)  
Var  
  p :^Cellule  
Debut  
  Allouer (p)  
  p^.cle ← info  
  p^.suiv ← A  
  /*mettre à jour sommet*/  
  A ← p  
Fin Proc  
  
Procedure depiler(var A :^Cellule)  
Var  
  q :^Cellule  
Début  
  assure( !vide(A))  
  q ← A  
  A ← A^.suiv  
  liberer (q)  
Fin Proc
```

**Exemple d'utilisation :****Algorithme principale****Var**

P1 : ^Cellule

P2 : ^Cellule

i : entier

**Début**

p1 ← créer\_pile()

**pour** i **de** 1 **à** 10 **faire**

empiler(i,P1)

**fin pour**

p2 ← créer\_pile()

**pour** i **de** 1 **à** 10 **faire**

empiler(i,P2)

**fin pour**

/\*affichage de p1 et p2\*/

**pour** i **de** 1 **à** 10 **faire**

ecrire(dernier(P1))

ecrire(dernier(P2))

depiler(P1)

depiler(P2)

**fin pour**

/\*en principe p1 et p2 sont vides\*/

**Si** (vide(p1) **et** vide(p2)) **alors**

écrire("quelle joie!!")

**sinon**

écrire("problème!?!")

**fin Si****Fin Algo**

## 5. Exercice d'application

Ecrire un algorithme qui permet de lire un entier et d'afficher tous les chiffres qui le composent.

**Exemple :**

Si le nombre proposé est 2345 alors le programme souhaité doit afficher dans l'ordre : 2, 3, 4, 5

**Solution****Idée :** divisons entiers successives

=&gt;utiliser la SD pile

