


Piles






1. Définition

- Structure de données à accès restreint qui regroupe des éléments de même type
 Seul le dernier élément entré est accessible : le **sommet**
- LIFO : Last In First Out

Exemple : Une pile d'assiettes

Piles

2. Primitives

- Initialisation de la pile
  `init_pile(p)`
- Test de l'état de la pile
  `est_vide(p)`
- Ajout d'un élément en sommet de pile
  `empiler(p,v)`
- Retrait de l'élément de sommet de pile
  `depiler(p,v)`
- Renvoi de la valeur en sommet de pile
  `Consultation_Sommet(p)`

Piles

3. Exemples

- Pile d'exécution sollicitée lors de l'appel des sous-programmes

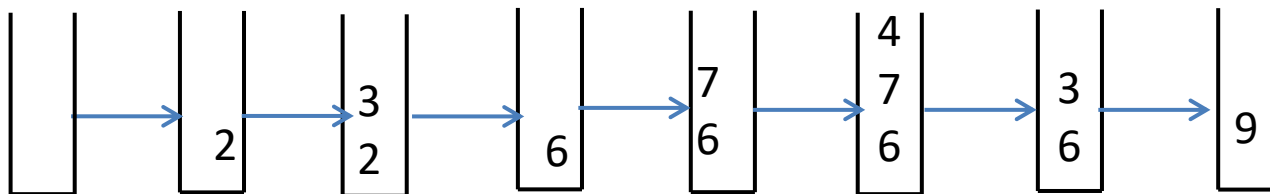
➡ Sauvegarde de contexte

- ☐ Informations nécessaires à l'appel du sous-programme :
paramètres, adresse de retour (prochaine instruction à exécuter)
- ☐ Variables locales du sous-programme

Piles

3. Exemples

- Evaluation des expressions arithmétiques postfixées
 - ❑ Une expression arithmétique = { opérandes, opérateurs }
 - ❑ Dans une expression arithmétique postfixée, les opérateurs sont placés après les opérandes :
 - 2 3 *
 - 2 3 * 7 4 - +



Piles

4. Implémentation sous forme de tableaux

- Tableau + un entier = sommet
- Sommet = indice de la dernière case remplie
- Sommet initialisé à 0
- Déclaration

```
const nbelt =100
type Pile = enregistrement
    tab : tableau[1..nbelt] de type entier
    sommet : entier          /* sommet allant de 0 à nbelt */
fin_enregistrement
```

Piles

4. Implémentation sous forme de tableaux

- Initialisation de la pile

```
procedure init_pile(var p :pile)
```

```
/* la pile est vide : sommet ne désigne aucune case remplie*/
```

```
Debut
```

```
    p.sommet := 0
```

```
fin
```

Piles

4. Implémentation sous forme de tableaux

- Test de la pile vide

```
fonction pile_vide(p :pile):booleen  
/* retourne vrai si la pile est vide, faux sinon*/  
debut  
    retourner(p.sommet = 0)  
fin
```

Piles

4. Implémentation sous forme de tableaux

- Test de la pile pleine : nécessaire à cause de la structure même des tableaux

```
fonction pile_pleine(p :pile):booleen  
/* retourne vrai si la pile est pleine, faux sinon */  
debut  
    retourner(p.sommet = nbelt)  
fin
```


Piles

4. Implémentation sous forme de tableaux

- Retrait de l'élément en sommet de pile

```
procedure depiler(var p : pile, var v : entier, var ok : boleen)
/* retrait d'un élément en sommet de pile si celle-ci n'est pas vide* /
debut
    ok := non(pile_vide(p))
    si ok alors
        v := p.tab[p.sommet]
        p.sommet := p.sommet - 1
    fsi
fin
```

Piles

4. Implémentation sous forme de tableaux

- Ajout d'un élément sur la pile

procedure empiler(var p :pile, v : entier)

/ ajout de la valeur v en sommet de pile si celle-ci n'est pas pleine */*

debut

si non(pleine_pile(p)) **alors**

 p.sommet := p.sommet + 1

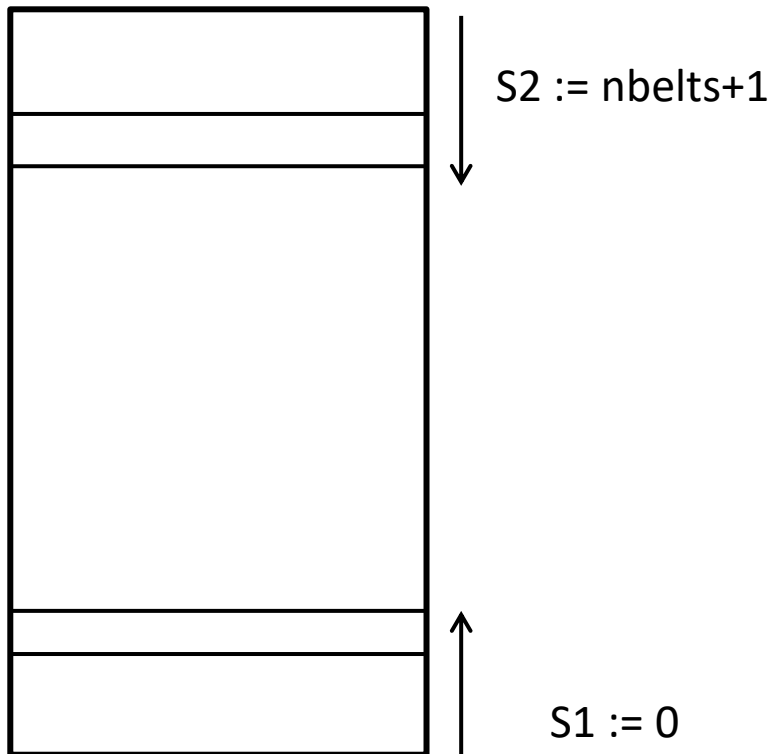
 p.tab[p.sommet]:= v

fsi

fin

Piles

5. Implémentation de deux piles dans un seul tableau

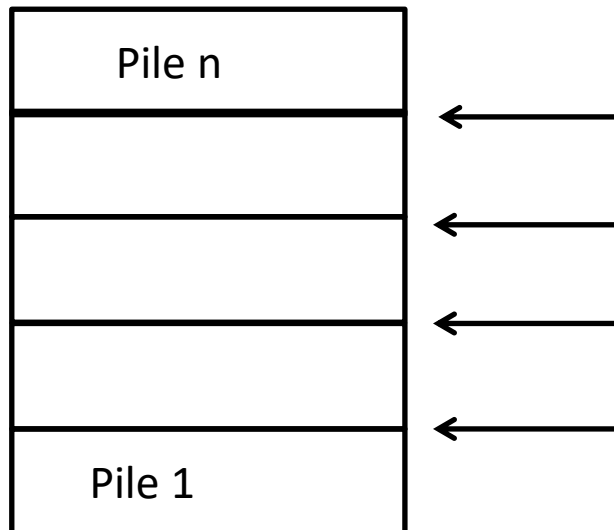


```
const nbelt = 100
type piles = enregistrement
    tab : tableau[1..nbelt] de type entier
    S1 : entier
    S2 : entier
fin_enregistrement
```

- S1 est incrémenté lors d'un ajout, décrémenté lors d'un retrait
- S2 est décrémenté lors d'un ajout, décrémenté lors d'un retrait
- S1 doit toujours être strictement inférieur à S2

Piles

6. Implémentation de plus de deux piles dans un seul tableau



- Toutes les piles ont le même nombre d'éléments : nombre maximum d'éléments du tableau divisé par le nombre de piles
- Lors de l'ajout d'un élément dans une pile , son sommet est incrémenté
- Lors du retrait d'un élément dans une pile, son sommet est décrémenté

Piles

7. Exercices

- ❑ Que contient la pile p après l'exécution de la procédure test_pile?

procedure test_pile(var p : pile)

var v : entier

début

init_pile(p)

empiler(p,1)

depiler(p,v)

empiler(p,2)

v=valeur_sommet(p) /* fonction qui retourne la valeur du sommet de pile*/

empiler(p,1)

empiler(p,v)

empiler(p,3)

v=sommet(p)

empiler(p,2)

empiler(p,v)

fin

Piles

7. Exercices

On considère deux piles d'entiers P1 et P2, de type pile (dimension max),

- ☐ Ecrire une procédure *inverse_ordre* qui déplace les entiers de P1 vers P2 en inversant leur ordre
- ☐ Ecrire une procédure *meme_ordre* qui déplace les entiers de P1 vers P2 en conservant leur ordre
- ☐ Ecrire une procédure *pair_impair* qui déplace les entiers de P1 vers P2 en faisant en sorte que les entiers pairs soient sous les entiers impairs