Cours: ASD 2

Chapitre 3: Les Piles

Réalisé par:
Sakka Rouis Taoufik

1

Ch1: Les Piles

I. Introduction

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

- **creer_pile** : elle permet de créer une pile vide
- **pile_vide** : elle permet de voir est ce qu'une pile est vide ou non ?
- **empiler** : elle permet d'ajouter un élément de type T à la pile
- **depiler** : elle permet de supprimer un élément de la pile
- **dernier** : elle retourne (ou rend) le dernier élément empilé et non encore dépilé.

2

I. Introduction

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 pour l'opération dernier.

II. 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.

Ch1: Les Piles

II. Propriétés

REMARQUE : ces cinq 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).

5

Ch1: Les Piles

III. Représentation physique

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.

6

Ch1: Les Piles

III. Représentation physique

3.1 Représentation chaînée

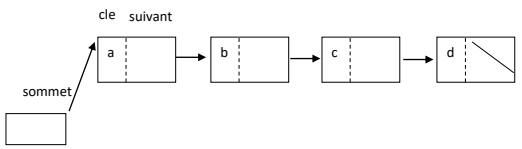
Utilisation des pointeurs pour relier explicitement les éléments formant la SD.

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.

a
b
c
d

Représentation abstraite



7

Ch1: Les Piles

III. Représentation physique

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 utilisé**e : elle est comprise entre sommet+1 et n.
- Avec sommet est indice compris entre 0 et n.

a
b
c
d

Sommet = 4

d	1
c	2
b	3
a	4
	n

utilisé

non utilisée

8

Ch1: Les Piles

IV. Matérialisation de la SD Pile

Matérialisation de la SD Pile :
- **Objet abstrait (OA)**
- **Type de données Abstrait (TDA)**

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

9

Ch1: Les Piles

IV. Matérialisation de la SD Pile
4.1 Structure de données Pile comme OA

- On va proposer une interface pile regroupe les services exportées par cette SD.
- 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

10

Ch1: Les Piles

IV. Matérialisation de la SD Pile
4.1 Structure de données Pile comme OA

- En supposant que les éléments de la Pile sont des entiers, celle-ci se déclare de la façon suivante :

```
/*représentation physique*/
Types
Cellule = Struct
        cle : entier
        Suiv : ^Cellule
FinStruct
Var
        sommet :^Cellule statique
```

```
Procédure creer_pile ( )
Début
    sommet←NIL

Fin proc
```

11

Ch1: Les Piles

IV. Matérialisation de la SD Pile
4.1 Structure de données Pile comme OA

```
Fonction pile_vide ( ) : boolean
Début
    Si (sommet=Nil)  alors
        pile_vide← vrai
    Sinon
        pile_vide ← faux
    finsi
Fin Fn
```

```
Procedure empiler (info : entier)
Var
    p :^Cellule
Debut
    Allouer (p)
    p^.cle←info
    p^.suiv←sommet
    /*mettre à jour sommet*/
    sommet←p
Fin Proc
```

12

Ch1: Les Piles

IV. Matérialisation de la SD Pile
4.1 Structure de données Pile comme OA

```
Fonction dernier ( ) : entier
Debut
    assure( ! pile_vide() )
    dernier←sommet^.cle
Fin Fn
```

```
Procedure depiler ( )
Var
    q:^Cellule
Debut
    assure( ! pile_vide())
    q←sommet
    sommet←sommet^.suiv
    liberer (q)
Fin Proc
```

13

Ch1: Les Piles

IV. Matérialisation de la SD Pile
4.1 Structure de données Pile comme OA

```
/* Exemple d'utilisation  */
Algorithme Exemple
Var
    i : entier
Début
    creer_pile()
    assure ( pile_vide())
/* l'instruction assert en  C et en Java*/
    pour i de 1 à 10 faire
        empiler(i)
    fin pour
    assure (! pile_vide())
    pour i de 1 à 10 faire
        ecrire  (dernier ( ) )
        depiler()
    fin pour
    Assure ( pile_vide ( ) )
Fin Algo
```

14

IV. Matérialisation de la SD Pile
4.2 Structure de données Pile comme TDA

Dans cette partie, on va concrétiser la SD pile sous forme d'un **Type de données Abstrait** (TDA) capable de gérer plusieurs exemplaires de la SD pile et non pas un seul exemplaire (Objet Abstrait voir 4.1).

IV. Matérialisation de la SD Pile
4.2 Structure de données Pile comme TDA

```
    /*représentation physique*/
Types
    Cellule = Struct
        cle : entier
        Suiv : ^Cellule
    FinStruct

Fonction creer_pile ( ) : ^Cellule
Début
    creer_pile←NIL
Fin Fn
```

```
Fonction pile_vide (P :^Cellule): boolean
Début
        pile_vide ← (P=Nil)
fin Fn

Fonction dernier (P :^Cellule): entier
Début
assure( ! pile_vide (P))
dernier←P^.cle
fin Fn
```

Ch1: Les Piles

IV. Matérialisation de la SD Pile
4.2 Structure de données Pile comme TDA

```
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( ! pile_vide (A))
q←A
A←A^.suiv
liberer (q)
Fin Proc
```


Ch1: Les Piles

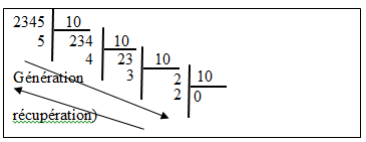
IV. Matérialisation de la SD Pile
4.2 Structure de données Pile comme TDA

```
/* Exemple d'utilisation  */
Algorithme principale
Var
    P1 :^Cellule
    P2 :^Cellule
    i : entier
Début
    p1←creer_pile()
    pour i de 1 à 10 faire
        empiler (i, P1)
    fin pour

    p2←creer_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 (pile_vide (p1) et pile_vide (p2)) alors
        écrire("quelle joie!!")
    sinon
        écrire("problème!?!?")
    fin Si
Fin Algo
```

V. Exercices d'application

Exercice 1 :
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



V. Exercices d'application

Exercice 2 :

Enrichir l'objet abstrait PILE, concrétisé par une représentation chainée, en intégrant les opérations suivantes :
nb_element : renvoie le nombre d'éléments de la pile.
remplace_sommet : change le sommet de la pile. Elle exige que la pile soit non vide.
effacer : efface tous les éléments de la pile.

V. Exercices d'application

Exercice 3 :

Ecrire un algorithme permettant de lire une expression avec parenthèses supposée valide (nombre des parenthèses ouvrantes=nombre de parenthèses fermantes) et d'afficher toutes les sous-expressions entre parenthèses en commençant par la plus interne. Par exemple, si l'expression soumise au programme est :

a+ (b-(c*d)+8.14)-(d*k) alors le programme demandé doit afficher :

(c*d)

(b-(c*d) +8.14)

(d*k)

V. Exercices d'application

Exercice 4 : Une maîtresse d'école décide d'instaurer un système de points correspondant à la participation des élèves. À chaque élève, est associée une pile de type LIFO qui mémorise les différentes réponses qu'il a faites. Si un élève répond bien à une question, alors un feu vert est empilé sur sa pile. Si un élève répond de façon approximative à une question, alors un feu jaune est empilé sur sa pile. Enfin, si un élève répond mal à une question, alors un feu rouge est empilé sur sa pile. Un élève a 0 point initialement, puis les points sont comptés de la façon suivante : si la pile contient 3 fois de suite la même couleur de feu, alors le nombre de points de l'élève est incrémenté de 10 points si les feux sont verts, 5 points s'ils sont jaunes et décrémenté de 5 points s'ils sont rouges. En outre, si 3 feux identiques apparaissent au sommet de la pile, alors les trois feux sont dépilés de la pile.

1. Détaillez le principe mis en place par la maîtresse, le nombre de points obtenus et la pile finale pour l'élève qui a reçu la suite de feux suivants (V désigne un feu vert, J un jaune et R un rouge) **VVVJJJRRRRVVJJRRRJVVV**

2. Écrivez une fonction qui prend en entrée la pile d'un élève munie des primitives vues en cours, et un tableau de feux et qui modifie la pile afin de calculer la valeur associée à l'élève et qui renvoie cette valeur. On pourra considérer que la pile est vide lors de l'appel de la fonction.