1 Structures de données

DEFINITION 1 :
• *Un **objet** représente un concept, une idée ou toute entité du monde physique ou virtuel.*
• *Un **conteneur** est un objet permettant de stocker d'autres objets (non forcément distincts).*
• *Une **structure de données** est un conteneur dynamique (i.e. de longueur modifiable). Elle est destinée à organiser des données informatiques dans le but de maximiser l'**efficacité** de certaines opérations (moindre coût temporel, moindre espace en mémoire).*
Un conteneur S peut être étudié selon trois points de vue :
• l'accès, c'est-à-dire la manière d'accéder aux éléments du conteneur.
• le stockage, c'est-à-dire la manière de stocker les éléments du conteneur ;
• le parcours, c'est-à-dire la manière de parcourir les éléments du conteneur.

2 Pile (*stack*)

2.1 Structure LIFO

DEFINITION 2 : *Soit E un ensemble non vide. Une pile p d'éléments de E est une structure de données qui met en oeuvre le principe :*

dernier entré, premier sorti** ou **LIFO (Last - In - First - Out)

avec les conventions suivantes : une pile P est :
• *soit $\emptyset$ (pile vide, de profondeur 0) ;*
• *soit une pile de profondeur $n \geqslant 1$, de la forme $p = (q, s)$ où s est un élément de E, appelé **sommet** de p. et q une pile de profondeur $n-1$, appelé **queue** de p.*

Applications en "ingénierie inverse" : "retour à la page précédente", " Undo"... Quid d'une file d'attente ?

2.2 Opérations primitives sur les piles

Pour pouvoir utiliser les piles, il suffit de disposer des fonctions suivantes :
• est_vide(p) : renvoie Vrai si la pile p est vide, Faux sinon ;
• pile_vide() : renvoie la pile vide ;
• empiler(p, s) : ajoute s au sommet de la pile p (en anglais **push**), mais **ne renvoie rien** ;
• depiler(p) : renvoie et supprime le sommet de la pile non vide p (en anglais **pop**) ;
• sommet(p) : renvoie (sans le supprimer) le sommet de la pile non videp (en anglais *peek* ou *top*).

Une pile est une structure LIFO sans parcours possible : on ne peut accéder qu'à l'élément situé au sommet.

2.3 Mise en oeuvre d'une pile avec un tableau

DEFINITION 3 : *Un **tableau** T est une structure ordonnée qui contient des éléments de même nature.*

Tous les langages de programmation permettent de réaliser des tableaux. Pour le langage Python, le plus simple est d'utiliser des listes : $\boxed{T[0] \mid T[1] \mid \ldots \mid T[n-1]}$ où n est la longueur de T

et de réaliser une pile $p = (q, s)$ de profondeur n comme un tableau T tel que $q = T[: n-1]$ et $s = T[n-1]$.

Exercice n° 1 *battre les cartes*
Écrire une fonction *melanger(p1, p2)* qui mélange les éléments de $p1$ et $p2$ dans une troisième pile de la façon suivante : tant qu'une pile (au moins) n'est pas vide, on retire aléatoirement un élément au sommet d'une des deux piles et on l'empile sur la pile résultat.

Exercice n° 2 *conversion base 10 / base 2*
Écrire une fonction *conversion(nb10)* qui retourne, à l'aide de piles, la représentation en base 2 du nombre nb10 représenté en base 10.

Exercice n° 3 *évaluer une expression en notation polonaise inversée*
Écrire une fonction *eval_polonaise(exp)* qui permet d'évaluer une expression en notation polonaise inversée, ceci à l'aide d'une pile. (cf. les explications au tableau)

Exercice n° 4 *Fonctions évoluées*

Quand on travaille sur les piles, on s'impose 2 contraintes :
• utiliser **uniquement les 5 fonctions primitives**,
• si une fonction admet une pile p pour paramètre et la modifie en cours de programme, il faut rétablir p dans son état initial à la sortie du programme
(on rappelle que, si `L` est une liste Python, `L1=L` crée la liste `L1` à la même adresse mémoire que `L`).

Programmer alors les fonctions suivantes :

1. `afficherPile` doit afficher la pile en colonne, sommet en haut, base en bas.
2. `renverserPile(`p`)` doit afficher en colonne la pile dont les éléments sont dans l'ordre inverse de ceux de p.
3. `permuter2(`p`)` doit afficher en colonne la pile dont les 2 éléments au sommet sont permutés.

Exercicen° 4 *Fonctions évoluées*

Quand on travaille sur les piles, on s'impose 2 contraintes :
• utiliser **uniquement les 5 fonctions primitives**,
• si une fonction admet une pile p pour paramètre et la modifie en cours de programme, il faut rétablir p dans son état initial à la sortie du programme
(on rappelle que, si `L` est une liste Python, `L1=L` crée la liste `L1` à la même adresse mémoire que `L`).

Programmer alors les fonctions suivantes :

1. `afficherPile` doit afficher la pile en colonne, sommet en haut, base en bas.
2. `renverserPile(`p`)` doit afficher en colonne la pile dont les éléments sont dans l'ordre inverse de ceux de p.
3. `permuter2(`p`)` doit afficher en colonne la pile dont les 2 éléments au sommet sont permutés.

Exercicen° 4 *Fonctions évoluées*

Quand on travaille sur les piles, on s'impose 2 contraintes :
• utiliser **uniquement les 5 fonctions primitives**,
• si une fonction admet une pile p pour paramètre et la modifie en cours de programme, il faut rétablir p dans son état initial à la sortie du programme
(on rappelle que, si `L` est une liste Python, `L1=L` crée la liste `L1` à la même adresse mémoire que `L`).

Programmer alors les fonctions suivantes :

1. `afficherPile` doit afficher la pile en colonne, sommet en haut, base en bas.
2. `renverserPile(`p`)` doit afficher en colonne la pile dont les éléments sont dans l'ordre inverse de ceux de p.
3. `permuter2(`p`)` doit afficher en colonne la pile dont les 2 éléments au sommet sont permutés.

Exercicen° 4 *Fonctions évoluées*

Quand on travaille sur les piles, on s'impose 2 contraintes :
• utiliser **uniquement les 5 fonctions primitives**,
• si une fonction admet une pile p pour paramètre et la modifie en cours de programme, il faut rétablir p dans son état initial à la sortie du programme
(on rappelle que, si `L` est une liste Python, `L1=L` crée la liste `L1` à la même adresse mémoire que `L`).

Programmer alors les fonctions suivantes :

1. `afficherPile` doit afficher la pile en colonne, sommet en haut, base en bas.
2. `renverserPile(`p`)` doit afficher en colonne la pile dont les éléments sont dans l'ordre inverse de ceux de p.
3. `permuter2(`p`)` doit afficher en colonne la pile dont les 2 éléments au sommet sont permutés.