1 Les piles et les files

1.1 Utilisation des piles

Exercice n° 1 Implémentation d'une pile par un tableau

Rappel : La pile est une structure de type LIFO (Last In, First Out), on accède uniquement au sommet : **la seule valeur directement accessible est la dernière ajoutée.** Pour manipuler une pile, on définit les fonctions suivantes :

- ***estVide*** : teste si la pile est vide (renvoie un booléen)
- ***afficherPile*** : affiche les éléments de la pile.
- ***sommetPile*** : affiche le sommet de la pile.
- ***empiler*** : ajoute un élément au sommet de la pile
- ***depiler*** : renvoie l'objet situé au sommet de la pile et le retire.

1. Ecrire et tester la fonction est ***estVide***

```
>>>tab1=[]
>>>tab2=["p","y","t","h","o"]
>>> estVide(tab1)
True
>>>estVide(tab2)
False
```

2. Ecrire et tester la procédure est ***sommetPile***

```
>>>sommetPile(tab2)
o
```

3. Ecrire et tester la fonction est ***empiler***

```
>>>empiler(tab2,"n")
["p","y","t","h","o","n"]
```

4. Ecrire et tester la fonction est ***depiler***

```
>>>depile(tab2)
["p","y","t","h","o"]
>>>depile(tab2)
["p","y","t","h"]
```

5. Ecrire et tester la procédure est ***afficherPile***

```
>>>afficherPile(tab2)
o
h
t
y
p
```

6. Votre procédure ***afficherPile*** a t-elle un effet de bord ? Si oui la réécrire afin de palier le problème.
7. Ecrire une fonction ***estPresent*** qui permet de tester si un élément est présent dans une pile.

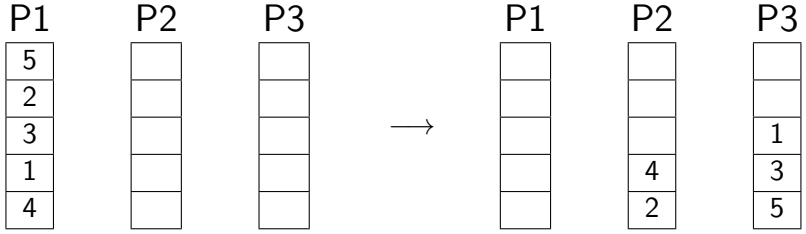
```
>>>estPresent(tab2,"y")
True
>>>estPresent(tab2,"a")
False
```

8. Votre procédure ***estPresent*** a t-elle un effet de bord ? Si oui la réécrire afin de palier le problème.
9. Ecrire une fonction ***nbEltPile*** qui permet de compter les éléments d'une pile. (sans effet de bord)

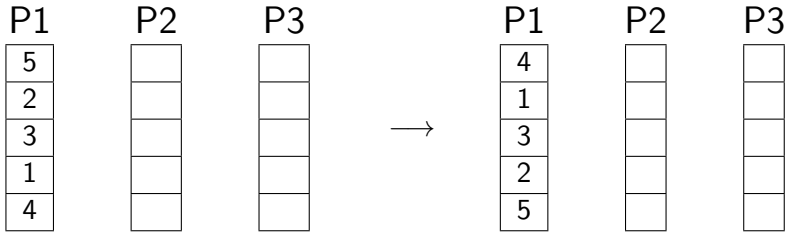
Exercice n° 2

En utilisant les fonctions vues précédemment, écrire algorithme qui permet de :

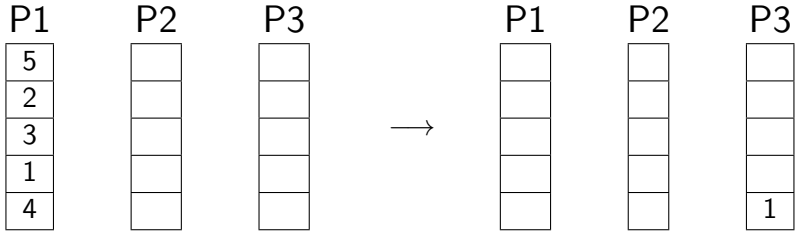
1. Séparer les nombres pairs des nombres impairs



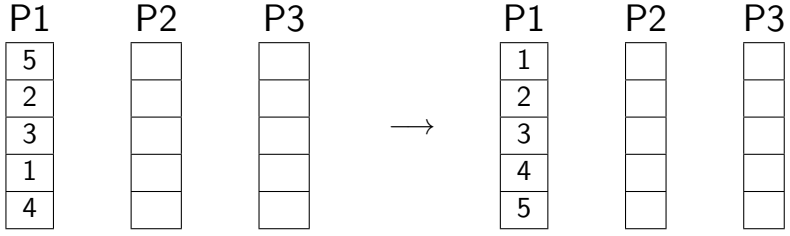
2. Renverser la pile P1



3. Obtenir le plus petit élément de P1.



4. Trier la pile P1.



Exercice n° 3

Nous allons étudier ici la vérification des délimiteurs dans une expression.
On distingue ici trois délimiteurs, les parenthèses"(" et ")", les accolades "{" et "}" et les crochets "[" et "]". L'objectif est de déterminer si une chaîne de caractères (qui pourrait être un programme) est correctement délimitée, c'est-à-dire si chacun des délimiteurs ouvrant est suivi par un délimiteur fermant.

1. Les exemples suivants sont-ils bien délimitées ?

   a(b)c ; a[b(c)}d ; {a(b[c[d))]e]} ; {a(b(d[d]))e}

2. Est-il possible de vérifier une chaîne uniquement en comptant les délimiteurs ? Si non, proposez un contre-exemple.
3. Ecrire une fonction ***verifDelim*** qui prend en paramètre une chaîne de caractère est qui renvoie un boolean indiquant si la chaîne est correctement délimitée. Cette fonction ne considérera que les délimiteurs "(" et ")"
4. Ecrivez une fonction ***verifDelimStack*** qui réalise vérification de délimiteurs à partir d'une PILE.
5. **[BONUS]** Modifiez votre algorithme de manière à afficher les indices des caractères qui posent problème, par exemple lorsqu'il manque un délimiteur fermant ou qu'il ne s'agit pas du délimiteur fermant correspondant au délimiteur ouvrant.

Exercice n° 4 Notation polonaise inversée : NPI

Avec la notation polonaise inversée,les opérandes (nombres) précèdent l'opérateur (+, -, /,*).

Par exemples :

2 3 + signifie 2 + 3 1 2 + 3 4 − ∗ signifie $(1+2)*(3-4)$

Pour effectuer le calcul d'une NPI, on applique l'algorithme suivant : On lit (de gauche à droite) un élément à la fois.

- Si c'est une opérande (un entier), on l'empile
- Si c'est un opérateur binaire, on dépile deux éléments sur lesquelles on applique l'opérateur et on empile le résultat.

Appliquer cet algorithme pour calculer les expressions ci-dessous :

1. 4 5 ∗ 2 ∗ 1 +

2. 7 3 + 7 3 3 ∗ + −

3. 5 9 3 + 4 2 ∗ ∗ 7 + ∗

4. 3 7 + 6 5 − ∗ 4 2 − 2 3 + ∗ /

Remarques : L'avantage de cette notation est que les calculs peuvent être écrit sans parenthèses.

1.2 Utilisation des files

Exercice n° 5 Implémentation d'une file par tableau

La file est une structure de type FIFO (First In, First Out). La seule valeur directement accessible est la première ajoutée. Pour manipuler une file, on définit les fonctions suivantes

- ***estVide*** : teste si la file est vide (renvoie un booléen)
- ***afficherFile*** : affiche les éléments de la file.
- ***sommetFile*** : affiche le sommet de la file.
- ***enfiler*** : ajoute un élément au sommet de la pile
- ***defiler*** : renvoie l'objet situé au sommet de la file et le retire.

1. Ecrire et tester la fonction est ***estVide***

```
>>>tab1=[]
>>>tab2=[”p”,”y”,”t”,”h”,”o”]
>>> estVide(tab1)
True
>>>estVide(tab2)
False
```

2. Ecrire et tester la fonction est ***sommetFile***

```
>>>tab=[”H”, ”e” ,”l”, ”l”]
>>>sommetFile(tab)
”H”
```

3. Ecrire et tester la fonction est ***afficherFile***

```
>>>tab=[”H”, ”e” ,”l”, ”l”]
>>>afficherFile(tab)
”H <- e <- l <- l”
```

4. Ecrire et tester la fonction est ***enfiler***

```
>>>enfiler(tab,”o”)
[”H”, ”e” ,”l”, ”l”, ”o”]
```

5. Ecrire et tester la fonction est ***defiler***

```
>>>depile(tab)
[”H”, ”e” ,”l”, ”l”]
>>>depile(tab)
[”H”, ”e” ,”l”]
```

6. Ecrire une fonction qui permet de tester si un élément est présent dans une file.

```
>>>file=[”a”,”z”,”e”,”r”,”t”,”y”]
>>>estPresent(file,”y”)
True
>>>estPresent(file,”u”)
False
```

7. Votre procédure ***estPresent*** a t-elle un effet de bord ? Si oui la réécrire afin de palier le problème.
8. Ecrire une fonction qui permet de compter le nombre d'éléments dans une file.
9. Votre procédure ***estPresent*** a t-elle un effet de bord ? Si oui la réécrire afin de palier le problème.
10. Ecrire une fonction ***nbEltFile*** qui permet de compter les éléments d'une file. (sans effet de bord)