Un exemple d'utilisation de piles : expressions postfixes et infixes

Dans ce tp, on se sert de *piles*. Pour simuler une pile en python, on utilisera le type `list` limité aux méthodes `append` et `pop`.

1 Présentation

La *notation polonaise inversée* dite aussi *postfixe* permet d'écrire sans parenthèses ni règles de priorités une suite d'opérations arithmétiques qui impliquent des opérateurs et des opérandes. Pour simplifier, on suppose que l'on n'utilise que les 4 opérateurs binaires `+,*,/,-`. En postfixe, les deux opérandes d'un opérateur apparaissent *avant* l'opérateur.

Exemple : l'expression infixe `( ( 3 + 14 ) / ( 5 - 2 ) )` s'écrit en postfixe :

```
3 14 + 5 2 - /
```

1. Convertir l'expression infixe `( ( ( 2 * ( 7 - 9 ) ) + 3 ) / 2 )` en postfixe, en préservant l'ordre des opérandes.
2. Évaluer l'expression postfixe suivante.

   ```
   3 4 + 5 * 7 /
   ```

2 Évaluation postfixe

Le principe : on part d'une pile vide et on lit l'expression postfixe de la gauche vers la droite.
- Si la chaîne lue est un opérande, on l'empile ;
- si c'est un opérateur, on dépile les deux dernières valeurs de la pile, on leur applique l'opérateur, et on empile le résultat.

On continue ainsi jusqu'à épuisement de l'expression. À l'issue du processus – si l'expression postfixe initiale est valide – la pile ne contient qu'une valeur qui est le résultat des opérations arithmétiques.

3. Écrire une fonction `eval_postfixe` qui prend en entrée une chaîne [1] représentant une expression postfixe, et qui renvoie le résultat en suivant le principe détaillé plus haut. On utilisera la méthode `split` sur les chaînes pour transformer la chaîne initiale à traiter en une liste de chaînes.

3 Conversion infixe vers postfixe

On suppose que l'on dispose d'une expression infixe où chaque opération est parenthésée. Par exemple, `( ( 5 * 3 ) + 4 )`.

Dans une première étape, on vérifie que l'expression infixe est correctement parenthésée.

Le principe : on part d'une pile vide et on lit l'expression infixe de la gauche vers la droite.
- Si la chaîne lue est une parenthèse ouvrante, on la place dans la pile ;
- si la chaîne lue est une parenthèse fermante, on dépile la parenthèse ouvrante correspondante.

L'expression est correctement parenthésée si la pile est vide à la fin.

4. Écrire une fonction `verif` qui prend une expression infixe et qui vérifie qu'il y a autant de parenthèses ouvrantes que fermantes.
5. On veut convertir une expression infixe correctement et complètement parenthésée en une expression postfixe *en gardant l'ordre des opérandes*. Par exemple, `( 5 * ( 3 + 4 ) )` sera converti en `5 3 4 + *`

1. Les différents éléments sont séparés par un espace ; dans le cas de nombres négatifs, on veillera à coller (donc sans espace) le signe - devant le nombre pour ne pas confondre avec l'opérateur soustraction.

Le principe : on part de deux piles vides `pile1` et `pile2`. La pile `pile1` va contenir les opérateurs et les parenthèses et la pile `pile2` l'expression postfixe recherchée. De plus, on associe à chaque opérateur un ordre de priorité ; les opérateurs `*` et `/` sont de priorité 2 et les opérateurs `+` et `-` sont de priorité 1. On lit l'expression de la gauche vers la droite :

- Si la chaîne lue est un opérande on l'empile dans `pile2` ;
- si la chaîne lue est une parenthèse ouvrante, on l'empile dans `pile1` ;
- si la chaîne lue est une parenthèse fermante, on dépile `pile1` jusqu'à enlever la parenthèse ouvrante correspondante. Chaque opérateur rencontré pendant ce dépilage est empilé dans `pile2` ;
- si la chaîne lue est un opérateur, on l'empile dans `pile1` après avoir dépilé les éventuels opérateurs déjà présents dans `pile1` et qui ont une priorité supérieure ou égale à cet opérateur.

Si à l'issue de la lecture il reste des opérateurs dans `pile1`, on les dépile et on les empile dans `pile2`.