TP 7 – Piles et files

Année 2019-2020

Ce TP permet de créer des structures de données de type pile, de les implémenter à partir de listes chaînées simples, et d'écrire des algorithmes dans le cadre d'applications qui les utilisent.

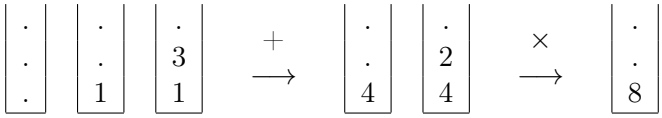
Exercice 1 – Implémentation du type abstrait Pile par une liste chaînée simple

Une pile est une séquence d'éléments dans laquelle les insertions et les suppressions se font à une seule extrémité. Les piles sont aussi appelées LIFO (Last-In-First-Out), c'est-à-dire dernier entré premier sorti.

1. Complétez la spécification de la classe `Pile_List` (fichier *tp7_pile_list.py*).
2. Implémentez l'ensemble des méthodes définies dans la spécification `Pile_List` à partir d'une liste chaînée.
3. Testez les opérations d'empilement et de dépilement en créant plusieurs exemples de piles.

Exercice 2 – Évaluation d'expressions arithmétiques à partir d'une pile

On se propose de simuler l'évaluation d'expressions arithmétiques. On s'appuie sur une notation (dite parfois "polonaise" inverse ou notation postfixée) utilisée par certaines calculatrices. On doit d'abord saisir les opérandes (les nombres), puis l'opérateur. Par exemple 2.3 (enter) 3.1 + provoque l'affichage de la valeur réelle 5.4 (2.3 + 3.1).
Dans cet exercice, les expressions seront composées d'entiers et des opérateurs `+`, `-`, `*` et `/`.
Donnez les résultats des expressions suivantes : `2 3 +, 1 3 + 2 *, 4 3 * 2 5 + +`
L'intérêt de cette notation est quelle permet d'éviter les parenthèses. Si on compare cette notation à la notation traditionnelle infixe, on remarque la nécessité dans la notation infixe (opérateurs entre les opérandes) de prendre en compte les priorités entre opérateurs :
1 + 2 * 3 : calculer d'abord la multiplication (seconde opération) puis l'addition (première opération)
1 * 2 + 3 : calculer d'abord la première opération puis la seconde
alors que pour la notation postfixe, on ne tient compte que de l'ordre des opérations, par exemple :
1 2 3 * + (=7) et 1 2 * 3 + (=5)
Vous utiliserez une structure de pile d'entiers pour empiler les opérandes qui seront dépilés lorsque l'on doit leur appliquer une opération.
Exemple : Pour l'expression `(1, 3, +, 2, *)`, les états de la pile aux différentes étapes sont :



1. Ecrivez la fonction `EvalueExpr` qui contient la méthode `evalue(self,exp)` permettant d'évaluer une expression arithmétique *exp*. Les expressions arithmétiques pourront être entrées sous la forme d'une séquence de chaînes de caractères (en notation postfixée). Vous considérerez que les expressions ne traitent que des chiffres, et qu'il n'y a pas de caractère séparateur dans la chaîne.

2. Testez l'évaluation d'expression arithmétiques données en notation postfixée.

3. Pour étendre l'évaluation de l'expression arithmétique à des nombres, vous entrerez l'expression *exp* sous la forme d'une liste de chaînes de caractères.

Exercice 3 – Implémentation du type abstrait File par une liste chaînée simple

Une file est une séquence d'éléments dans laquelle les insertions se font à une extrémité et les suppressions se font à l'autre extrémité. Les files sont aussi appelées FIFO (First-In-First-Out), c'est-à-dire premier entré premier sorti.

1. Complétez la spécification de la classe `File_List` (fichier *tp7_file_list.py*).

2. Implémentez l'ensemble des méthodes définies dans la spécification `File_List` à partir d'une liste chaînée.

3. Testez les opérations d'ajout en queue et de suppression en tête en créant plusieurs exemples de files.