Université de Lille 1 SCIENCES ET TECHNOLOGIES

ufr d'IEEA
Formations en
Informatique de
Lille 1

L2 Informatique - 2016–2017

Algorithmes et Programmation 2

DS2 - 04/01/2017

durée 2h - documents non autorisés (sauf memento)

Ce devoir contient deux exercices indépendants. Sauf mention du contraire, on ne vous demande pas la documentation des fonctions que vous réaliserez.

Exercice 1 *Évaluation d'expressions arithmétiques infixées*

Le but de cet exercice est d'écrire un évaluateur d'expressions arithmétiques infixées.

Question 1 Rappelez l'interface du module `stack` vu en cours.

Analyse lexicale : Les expressions arithmétiques considérées dans cet exercice peuvent contenir des nombres, les quatres opérateurs arithmétiques de base (`+`, `-`, `*` et `/`), et des parenthèses ouvrantes (`(`) et fermantes (`)`). Par la suite nous appellerons ces différents composants légitimes d'une expression arithmétique infixée des *lexèmes* ou *token* en anglais.

Nous supposerons défini un module nommé `my_token` qui fournit un constructeur `make` de lexème, ainsi que deux sélecteurs `type` et `value`

```
>>> tok1 = my_token.make('(')
>>> my_token.type(tok1)
'PAR'
>>> my_token.value(tok1)
'('
>>> tok2 = my_token.make('+')
>>> my_token.type(tok2)
'OP'
>>> my_token.value(tok2)
'+'
>>> tok3 = my_token.make('123')
>>> my_token.type(tok3)
'NUM'
>>> my_token.value(tok3)
123.0
>>> my_token.make('12a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/eric/Enseignement/Licence/AP2/DS/16-17/my_token.py", line 65, in make
    raise BadString('{:s} is not a valid token'.format(string))
my_token.BadString: 12a is not a valid token
```

Ainsi l'expression arithmétique `'10 + 12 * ( 5 + 6 )'` est constituée de 9 lexèmes : 2 de type `PAR`,

4 de type `NUM` et 3 de type `OP`.

Question 2 Réalisez une fonction nommée `string2token_list` qui renvoie la liste des lexèmes contenus dans une expression arithmétique (sous forme d'une chaîne de caractères) supposée correctement espacée de sorte que la méthode `split` assure un bon découpage en lexème.

```
>>> expr = '10 + 12 * ( 5 + 6 )'
>>> expr.split()
['10', '+', '12', '*', '(', '5', '+', '6', ')']
>>> tok_list = string2token_list(expr)
>>> len(tok_list)
9
>>> my_token.type(tok_list[0])
'NUM'
>>> my_token.value(tok_list[0])
10.0
>>> my_token.type(tok_list[1])
'OP'
>>> my_token.value(tok_list[1])
'+'
```

Transformation en liste postfixée : L'analyse lexicale effectuée précedemment ne permet pas d'évaluer simplement une expression arithmétique infixée. Il faut encore en effectuer une analyse syntaxique pour connaître la structure de l'expression. En revanche, comme il a été vu en cours, les expressions postfixées sont très simples à évaluer, en particulier il n'y a plus besoin de parenthèses. Cette partie est donc consacrée au calcul de la liste des lexèmes dans l'ordre postfixé à partir de celle dans l'ordre infixé d'une expression arithmétique.

Question 3 Écrivez l'expression `'10 + 12 * ( 5 + 6 )'` sous forme postfixée.

La conversion d'un ordre infixé en un ordre postfixé d'une liste de lexèmes peut se faire aisément en utilisant une pile pour empiler les opérateurs et parenthèses ouvrantes. Cependant il est nécessaire d'introduire une notion de *précédence* qui indique quels sont les opérateurs prioritaires sur d'autres. La précédence d'un opérateur sera désignée par un nombre entier. Si la précédence d'un opérateur est strictement supérieure à celle d'un autre, cela signifie que le premier opérateur est prioritaire au second. La précédence est aussi définie pour les parenthèses qui ont la précédence la plus petite. Vous pouvez supposer définie une fonction `precedence` qui donne la précédence d'un lexème représentant un opérateur ou une parenthèse.

```
>>> [precedence(my_token.make(char)) for char in '(+-*/)']
[0, 1, 1, 2, 2, 0]
```

L'algorithme de conversion d'une liste de lexèmes dans l'ordre infixé en une liste de lexèmes en ordre postfixé peut être décrit comme suit :

1. Créer une pile vide `op_stack`, et créer une liste vide `res` pour le résultat à produire.
2. Pour chacun des lexèmes de la liste des lexèmes dans l'ordre infixé :
   (a) si le lexème est un nombre, l'ajouter à la fin de la liste `res`.
   (b) si le lexème est une parenthèse ouvrante, l'empiler dans `op_stack`.
   (c) si le lexème est une parenthèse fermante, dépiler `op_stack` jusqu'à ce que la parenthèse ouvrante correspondante soit dépilée, et ajouter les opérateurs dépilés en fin de la liste `res`.

(d) si le lexème est un opérateur, l'empiler dans `op_stack` en ayant pris soin auparavant de dépiler tous les opérateurs qui ont une précédence supérieure ou égale et les placer à la fin de `res`.

3. Vider `op_stack` en plaçant les opérateurs qui s'y trouvent à la fin de `res`.

Question 4 Réalisez une fonction nommée `infix2postfix` qui transforme une liste de lexèmes d'une expression infixée en une liste de lexèmes de l'expression postfixée équivalente.

```
>>> tok_list_post = infix2postfix(tok_list)
>>> len(tok_list_post)
7
>>> my_token.type(tok_list_post[0])
'NUM'
>>> my_token.value(tok_list_post[0])
10.0
>>> my_token.type(tok_list_post[1])
'NUM'
>>> my_token.value(tok_list_post[1])
12.0
```

Évaluation d'expressions postfixées : L'évaluation des expressions postfixées a été vue en cours.

Question 5 Réalisez une fonction nommée `eval_postfix`, paramétrée par une liste de lexèmes dans l'ordre postfixé qui renvoie la valeur de l'expression représentée par la liste.

```
>>> eval_postfix(tok_list_post)
142.0
```

Évaluation d'expressions infixées : Voici enfin atteint le but de cet exercice.

Question 6 Réalisez une fonction nommée `eval_expr` qui prend en paramètres une expression arithmétique infixée sous forme d'une chaîne de caractères, qui renvoie la valeur de cette expression.

```
>>> eval_expr(expr)
142.0
```

Exercice 2 *Économiser sur un trajet Lille-Paris*

Au 1er février 2016, pour effectuer un trajet Lille-Paris via l'autoroute A1, depuis la gare de péage de Fresnes jusqu'à celle de Roissy, il en coûte 16.30 euros pour un véhicule de classe 1, à condition de ne pas sortir entre ces deux gares (source SANEF). Mais qu'en est-il si on s'autorise de quitter l'autoroute à certaines gares pour y revenir immédiatement ? Par exemple, si un automobiliste venant de Fresnes quitte l'autoroute à Senlis, il lui en coûtera 12,40 euros, et s'il reprend immédiatement l'autoroute jusqu'à la gare de Roissy, il lui en coûtera 1,80 euros supplémentaires, ce qui fait un coût total de 14,20 euros, soit 2,10 euros de moins que le trajet direct. En fait, comme on le verra, le trajet le moins cher a un coût de 11,70 euros et nécessite de sortir à quatre gares de péage.

Le but de cet exercice est de programmer le trajet optimal du point de vue du coût.

Pour cela on suppose définies

— une liste `GARES_PEAGE_A1` contenant toutes les gares de péages rencontrées sur l'autoroute A1 dans le sens Lille-Paris :

```
>>> GARES_PEAGE_A1[0]
'FRESNES'
>>> GARES_PEAGE_A1[-1]
'ROISSY'
```

— et un dictionnaire `TARIFS_A1` dont les clés sont tous les couples d'indices (i, j) avec $0 \leq i \leq j <$ `len(GARES_PEAGE_A1)` et les valeurs associées sont les tarifs des tronçons d'autoroute compris entre la gare d'indice i et celle d'indice j. :

```
>>> TARIFS_A1[0,2]
3.3
>>> TARIFS_A1[0, len(GARES_PEAGE_A1) - 1]
16.3
>>> TARIFS_A1[GARES_PEAGE_A1.index('FRESNES'), GARES_PEAGE_A1.index('SENLIS')]
12.4
>>> TARIFS_A1[GARES_PEAGE_A1.index('SENLIS'), GARES_PEAGE_A1.index('ROISSY')]
1.8
```

Le but de la question qui suit est de réaliser une fonction `tarif_optimal` calculant le tarif correspondant à un trajet optimal. Cette fonction est paramétrée par les indices de deux gares, celles de départ et celle d'arrivée, et elle renvoie le tarif du parcours le moins cher du tronçon d'autoroute compris entre ces deux gares.

```
>>> tarif_optimal(0,2)
2.7
>>> tarif_optimal(0, len(GARES_PEAGE_A1) -1)
11.700000000000001
```

Question 1

Q 1–1 Quel est le tarif optimal entre une gare (d'indice i) et la gare suivante ?

Q 1–2 Soient deux indices d (comme départ) et a (comme arrivée) vérifiant $d < a - 1$. Donnez une formule exprimant `tarif_optimal(d, a)` en fonctions des grandeurs `TARIFS_A1[(d,i)]` et `tarif_optimal(i,a)`, où i varie dans un intervalle de valeurs que vous préciserez.

Q 1–3 Réalisez de manière récursive la fonction `tarif_optimal`.

Question 2 Réalisez une fonction récursive nommée `trajet_optimal` paramétrée par les indices de deux gares qui renvoie un couple dont la première composante est le tarif d'un trajet optimal, et la seconde est la liste de toutes les gares auxquelles il faut sortir pour effectuer le parcours le moins cher.

```
>>> trajet_optimal(0,2)
(2.7, [1, 2])
>>> trajet_optimal(0, len(GARES_PEAGE_A1) -1)
(11.700000000000001, [1, 3, 9, 10])
```
