Algorithmique et Programmation 2: Contrôle 1

On utilise la classe suivante poue les listes chaînées :

```
1  class Cellule (object):
2      def __init__(self, entier):
3          self.entier = entier
4          self.suiv = None
```

1 Affichage Incomplet

```
5  def afficheX(l,x):
6      tmp = l.suiv
7      while x != 0 and tmp != None:
8          print tmp.entier
9          tmp = tmp.suiv
10         x = x - 1
11
12 l1 = Cellule(0)
13 l1.suiv = Cellule(3)
14 l1.suiv.suiv = Cellule(2)
15 l1.suiv.suiv.suiv = Cellule(1)
16
17 afficheX(l1, 2)
18
19 afficheX(l1.suiv, 5)
20
21 l1.suiv.suiv.suiv = l1.suiv
22
23 afficheX(l1, 5)
```

1.1

Ligne 17 affiche quoi ?

1.2

Ligne 19 affiche quoi ?

1.3

Ligne 23 affiche quoi ?

2 Sous-Liste Monotone

2.1

```
24 def fonction1(l):
25     tmp = l.suiv
26     if tmp == None:
27         return
28     while tmp.suiv != None:
29         x = tmp.entier
30         if tmp.suiv.entier < x:
31             tmp.suiv = None
32         tmp = tmp.suiv
```

Comment cette fonction transforme la liste 1→2→4→2→1 ? Comment cette fonction transforme la liste 1→3→2→3→4 ?

2.2

```
33 def fonction2(l,x):
34     tmp = l.suiv
35     if tmp == None:
36         return
37     while tmp.suiv != None:
38         if tmp.suiv.entier < x:
39             tmp.suiv = tmp.suiv.suiv
40         else:
41             tmp = tmp.suiv
```

Si on prend x= 3, comment cette fonction transforme la liste 1→2→4→2→1 ?
Si on prend x= 2, comment cette fonction transforme la liste 1→3→2→3→4 ?

2.3

On appelle une liste *monotone* si aucun élément est plus petit que son prédecesseur. Ecrire une fonction `sousListeMon`, avec un argument, qui renvoie une sous-liste monotone tel qu'aucune autre sous-liste monotone contient cette sous-liste.

Pour 1→2→4→2→1 cette fonction devrait retourner 1→2→4. Pour 1→3→2→3→4 cette fonction devrait retourner 1→3→3→4 (sauf si vous trouvez une solution bizarre).

3 Arithmétique Postfix

On utilise les fonctions suivantes pour les piles et les files.

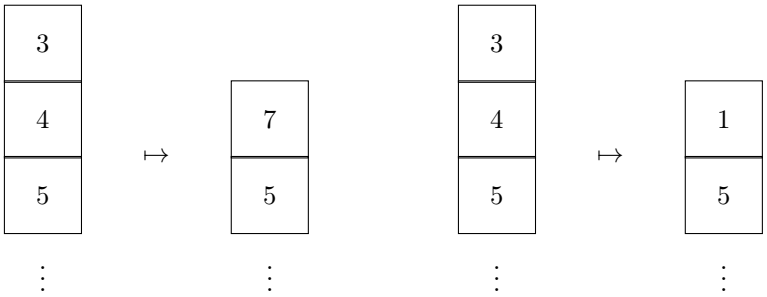
```
42 def empiler (e, p):
43     p.append(e)
44
45 def depiler (p):
46     p.pop()
47
48
49
50
51 def sommet (p):
52     return p [-1]
```

```
53 def enfiler (e,f):
54     f.append(e)
55
56 def defiler (f):
57     for i in range (len (f)
           -1):
58         f [i] = f [i + 1]
59     f.pop ()
60
61 def tete (f):
62     return f [0]
```

3.1

Ecrire une fonction `ajouter`, avec un argument, qui dépile deux éléments, les ajoute, et empile la somme.

Ecrire une fonction `soustraire` qui prend le sommet d'une pile et l'élément en dessous, soustrait le premier du deuxième, et met le résultat au sommet.



3.2

Supposons que `ajouter` et `soustraire` soient définies.

```
63 def resoudre(f):
64     p = []
65     while f != []:
66         x = tete(f)
67         defiler(f)
68         if x == 'a':
69             ajouter(p)
70         elif x == 's':
71             soustraire(p)
72         else:
73             empiler(x,p)
74     return p
```

Qu'est-ce qui est renvoyé par `resoudre([3,2,'s',4,'a'])`?

3.3

Trouver une file `f` avec exactement les éléments `'a'` et `'s'` et `2` et `3` et `4` (mais pas dans cet ordre) pour laquelle `resoudre(f)` retourne `[-1]`. (A noter: quelques files produissent erreurs.)

4 *Extra Postfix

Supposons que nous avons une liste chainée qui contient

`'('` `')'` `'+'` `'-'` `2` `3` `4`

(peut-être avec des répétitions). On appelle la liste «bien formée» si on peut calculer le résultat d'une manière unambigue. Par exemple, la liste

`'('` → `'('` → `2` → `'+'` → `3` → `')'` → `'-'` → `4` → `')'`

est bien formée.

4.1

Ecrire une fonction `postfix` tel que `resoudre(postfix(l))` retourne la bonne réponse si `l` est bien formée.

L'idée est la suivante: quand on a un numéro, mettez-le direct dans la file. Quand on a un opérateur, mettez `'a'` ou `'s'` dans une pile auxiliaire. Quand on a un paranthèse droite, prenez l'opérateur au sommet de la pile et mettez-le dans la file.