Informatique en CPGE (2017-2018)
Corrigé exercices : notion de piles

On commence par écrire les fonctions qui gèrent les piles soient les fonctions **creer_pile**, **empiler**, **depiler**, **pile_vide** et **taille**.

```
def creer_pile(c):  # crée une pile de capacité c
    p=(c+1)*[None]
    p[0]=0
    return p

def empiler(p,x):
    assert p[0]<len(p)-1
    p[0]=p[0]+1
    p[p[0]]=x

def depiler(p):
    assert p[0]>0
    x=p[p[0]]
    p[p[0]]=None
    p[0]=p[0]-1
    return x

def pile_vide(p):
    return p[0]==0

def taille(p):
    return p[0]
```

On importe ensuite ce fichier pour les différents exercices
On peut aussi créer une pile pour tester les différentes fonctions, par exemple :

```
from piles import *

# une pile contenant les caractères 'a', 'b', ..., 'z'
p=creer_pile(26)
for i in range(97,123):
    empiler(p,chr(i))
```

Exercice 1

```
def intervertit(p):
    x=depiler(p)
    y=depiler(p)
    empiler(p,x)
    empiler(p,y)
# pas besoin d'instruction return
```

Exercice 2

```
def troisieme(p):
    x=depiler(p)
    y=depiler(p)
    z=depiler(p)
    empiler(p,y)
    empiler(p,x)
    return z
```

Exercice 3

```
def nieme(p,n):
    assert n<=p[0]
    pile_temp=creer_pile(n-1)
    for i in range (n-1):
        empiler(pile_temp,depiler(p))
    z=depiler(p)
    empiler(p,z)
    for i in range(n-1):
        empiler(p,depiler(pile_temp))
    return z
```

Exercice 4

La fonction **pile_vide** utilise la structure de piles qui a été choisie.

```
def pile_vide(p):
    return p[0]==0:
```

Les fonctions **sommet** et **taille** sont indépendantes de la structure de pile choisie.

```
def sommet(p):
    assert pile_vide(p)==False
    x=depiler(p)
    empiler(p,x)
    return x

def taille(p):
    cpt=0
    while not pile_vide(p):
        depiler(p)
        cpt+=1
    return cpt
```

La fonction **taille** a une complexité en $\mathcal{O}(n)$ où n est la taille de la pile.

Exercice 5

```
def aufond(p):
    n=taille(p)
    pile_temp=creer_pile(n-1)
    sommet=depiler(p)
    for i in range (n-1):
        empiler(pile_temp,depiler(p))
    empiler(p,sommet)
    for i in range(n-1):
        empiler(p,depiler(pile_temp))
```

La complexité est en $\mathcal{O}(n)$ où n est la taille de la pile. En temps, le nombre d'opérations est de la forme $an+b$, et en espace on crée une nouvelle pile de taille $n-1$.

Exercice 6

```
def renverse(p):
    n=taille(p)
    pile_temp=creer_pile(n)
    for i in range (n):
        empiler(pile_temp,depiler(p))
    return pile_temp
```

Complexité comme à l'exercice précédent.

Exercice 7

```
from random import randint

def couper(p):
    n=taille(p)
    k=randint(1,n-1)
    pile_temp=creer_pile(k)
    for i in range (k):
        empiler(pile_temp,depiler(p))
    p2=creer_pile(k)
    for i in range (k):
        empiler(p2,depiler(pile_temp))
    return p2
```

Exercice 8

```
def melange(p1,p2):
    n1=taille(p1)
    n2=taille(p2)
    pile_temp=creer_pile(n1+n2)
    while (not pile_vide(p1)) and (not pile_vide(p2)):
        choix=randint(1,2)
        if choix==1:
```

```
            empiler(pile_temp,depiler(p1))
        else:
            empiler(pile_temp,depiler(p2))
    while not pile_vide(p1):
        empiler(pile_temp,depiler(p1))
    while not pile_vide(p2):
        empiler(pile_temp,depiler(p2))
    return pile_temp
```