Devoir surveillé n°3 – NSI
28/11/2022

Exercice 1

La notation polonaise inverse (NPI) permet d'écrire des expressions de calculs numériques sans utiliser de parenthèse. Cette manière de présenter les calculs a été utilisée dans des calculatrices de bureau dès la fin des années 1960. La NPI est une forme d'écriture d'expressions algébriques qui se distingue par la position relative que prennent les nombres et leurs opérations.

Exemple

Notation classique	Notation NPI
`3+9`	`3 9 +`
`8×(3+5)`	`8 3 5 + ×`
`(17+5)×4`	`17 5 + 4 ×`

L'expression est lue et évaluée de la gauche vers la droite en mettant à jour une pile.

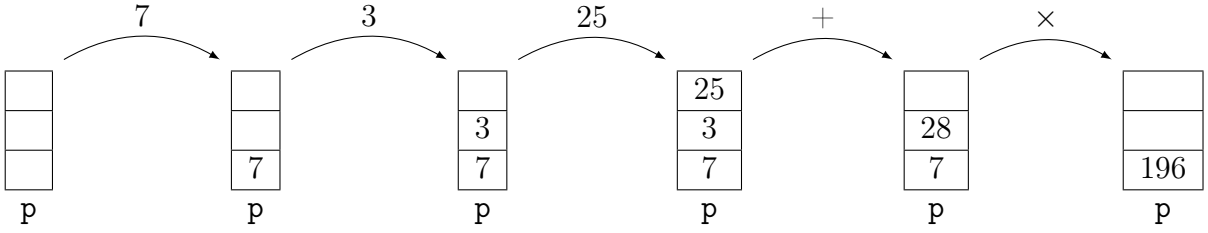
- Les nombres sont empilés dans l'ordre de la lecture.
- Dès la lecture d'un opérateur ($+$, $-$, $\times$, $/$), les deux nombres au sommet de la pile sont dépilés et remplacés par le résultat de l'opération effectuée avec ces deux nombres. Ce résultat est ensuite empilé au sommet de la pile.

A la fin de la lecture, la valeur au sommet est renvoyée.

Exemple l'expression `7 3 25 + ×` qui correspond au calcul `7×(3+25)` s'évalue à 196 comme le montrent les états successifs de la pile créée, nommée `p` :

- On empile la valeur 7.
- On empile la valeur 3.
- On empile la valeur 25.
- On remplace les deux nombres du sommet de la pile (25 et 3) par leur somme 28.
- On remplace les deux nombres du sommet de la pile (28 et 7) par leur produit 196.

Schéma descriptif des différentes étapes d'exécution



1. En vous inspirant de l'exemple ci-dessus, dessiner le schéma descriptif de ce que donne l'évaluation par la NPI de l'expression `12 4 5 × +`.
2. On dispose de la pile suivante nommée `p1` :

On rappelle ci-dessous les primitives de la structure de pile (LIFO : Last In First out) :

Fonction	**Description**
`pile_vide()`	Créé et renvoie une nouvelle pile vide
`empiler(p, e)`	Place l'élément `e` au sommet de la pile `p`.
`depiler(p)`	Supprime et renvoie l'élément se trouvant au sommet de `p`.
`est_vide(p)`	Renvoie un booléen indiquant si `p` est vide ou non.

On dispose aussi de la fonction suivante, qui prend en paramètre une pile `p` :

```
def top(p) :
    x = depiler(p)
    empiler(p, x)
    return x
```

On exécute la ligne suivante `temp = top(p1)` :

(a) Quelle valeur contient la variable `temp` après cette exécution ?

(b) Représenter la pile `p1` après cette exécution.

3. En utilisant uniquement les quatre primitives d'une pile, écrire en langage Python la fonction `addition(p)` qui prend en paramètre une pile `p` d'au moins deux éléments et qui remplace les deux nombres du sommet d'une pile `p` par leur somme.

   Remarque : cette fonction ne renvoie rien, mais la pile `p` est modifiée.

4. On considère que l'on dispose également d'une fonction `multiplication(p)` qui remplace les deux nombres du sommet d'une pile `p` par leur produit (on ne demande pas d'écrire cette fonction). Recopier et compléter, en n'utilisant que les primitives d'une pile et les deux fonctions `addition` et `multiplication`, la suite d'instructions (ci-dessous) qui réalise le calcul `(3+5)`$\times$`7` dont l'écriture en NPI est : `3 5 + 7` $\times$.

```
p=pile_vide()
empiler(p,3)
```

Exercice 2

On cherche ici à mettre en place des algorithmes qui permettent de modifier l'ordre des informations contenues dans une file. On considère pour cela les structures de données abstraites de Pile et File définies par leurs fonctions primitives suivantes :

Pile :

- `creer_pile_vide()` renvoie une pile vide ;
- `est_pile_vide(p)` renvoie `True` si la pile `p` est vide, `False` sinon ;
- `empiler(p, element)` ajoute `element` au sommet de la pile `p` ;
- `depiler(p)` renvoie l'élément se situant au sommet de la pile `p` en le retirant de la pile `p` ;
- `sommet(p)` renvoie l'élément se situant au sommet de la pile `p` sans le retirer de la pile `p`.

File :

- `creer_file_vide()` renvoie une file vide ;
- `est_file_vide(f)` renvoie `True` si la file `f` est vide, `False` sinon ;
- `enfiler(f, element)` ajoute `element` dans la file `f` ;
- `defiler(f)` renvoie l'élément à la tête de la file `f` en le retirant de la file `f`.

On considère de plus que l'on dispose d'une fonction permettant de connaître le nombre d'éléments d'une file :

- `taille_file(f)` renvoie le nombre d'éléments de la file `f`.

On représentera les files par des éléments en ligne, l'élément de droite étant la tête de la file et l'élément de gauche étant la queue de la file. On représentera les piles en colonnes, le sommet de la pile étant le haut de la colonne.

La file suivante est appelée f :

4	3	8	2	1

La pile suivante est appelée p :

5
8
6
2

1. Les quatre questions suivantes sont indépendantes. Pour chaque question, on repartira de la pile `p` et de la file `f` initiales (présentées ci-dessus).
   (a) Représenter la file `f` après l'exécution du code suivant :
   ```
   enfiler(f, defiler(f))
   ```
   (b) Représenter la pile `p` après l'exécution du code suivant :
   ```
   empiler(p, depiler(p))
   ```
   (c) Représenter la pile `p` et la file `f` après l'exécution du code suivant :
   ```
   for i in range(2):
       enfiler(f, depiler(p))
   ```
   (d) Représenter la pile `p` et la file `f` après l'exécution du code suivant :
   ```
   for i in range(2):
       empiler(p, defiler(f))
   ```
2. On donne ici une fonction `mystere` qui prend une file en argument, qui modifie cette file, mais qui ne renvoie rien.
   ```
   def mystere(f):
       p = creer_pile_vide()
       while not est_file_vide(f):
           empiler(p, defiler(f))
       while not est_pile_vide(p):
           enfiler(f, depiler(p))
       return p
   ```
   Préciser l'état de la variable `f` après chaque boucle de la fonction `mystere` appliquée à la file $\boxed{1}\boxed{2}\boxed{3}\boxed{4}$. Indiquer le contenu de la pile renvoyée par la fonction.
3. On considère la fonction `knuth(f)` suivante dont le paramètre est une file :
   ```
   def knuth(f):
       p=creer_pile_vide()
       N=taille_file(f)
       for i in range(N):
           if est_pile_vide(p):
               empiler(p, defiler(f))
           else:
               e = defiler(f)
               if e >= sommet(p):
                   empiler(p, e)
               else:
                   while not est_pile_vide(p) and e < sommet(p):
                       enfiler(f, depiler(p))
                   empiler(p, e)
       while not est_pile_vide(p):
           enfiler(f, depiler(p))
   ```
   (a) Recopier et compléter le tableau ci-dessous qui détaille le fonctionnement de cet algorithme étape par étape pour la file `2,1,3`. Une étape correspond à une modification de la pile ou de la file. Le nombre de colonnes peut bien sûr être modifié.

f	2,1,3	2,1			...
p	$\sqcup$	$\boxed{3}$			...

   (b) Que fait cet algorithme ?