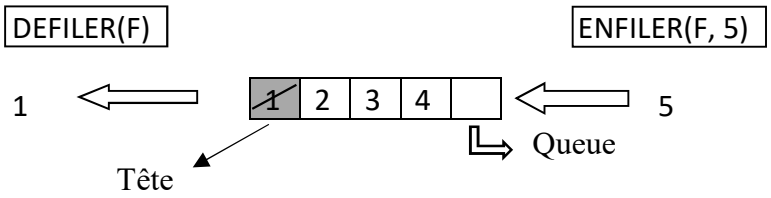
4. Les files

Définition : Une file est une structure de données dans laquelle on accède aux éléments suivant la règle du "premier arrivé, premier sorti". Autrement dit, on ne peut accéder qu'à **l'objet situé au début de la file**. On décrit souvent ce comportement par l'expression **"premier arrivé, premier sorti"**, c'est-à-dire, en anglais, **"First In, First Out" (FIFO)**. La file comporte une **tête** et une **queue**.

Analogie : la **file d'attente** des clients à un guichet ou à une caisse convient à cette description. En effet, le client qui passe en premier est celui qui est arrivé le premier.

Exemple d'application : Lorsqu'on veut imprimer plusieurs documents sur une imprimante, ceux-ci sont stockés dans une file : celui qui aura été lancé en premier sera le premier imprimé.

Les deux opérations élémentaires dont on a besoin pour cette structure sont :
ENFILER(F, x) qui correspond à l'insertion de la donnée x à la queue de la file F si la file n'est pas pleine.
DEFILER(F) qui retire la donnée de tête de la file F et la retourne si la file n'est pas vide.



Type abstrait File

Type abstrait : File
Données : éléments de type T

Opérations
CREER_FILE_VIDE() qui retourne un objet de type File.
La file existe et elle est vide.

ENFILER(F, e)
L'élément est inséré en queue de la file F.
DEFILER(F) qui retourne un objet de type T
L'élément situé en tête de la file F est enlevé de la file et est retourné.
EST_VIDE(F) qui retourne un objet de type Booléen.
Retourne Vrai si la file F est vide et retourne Faux sinon.
EST_PLEINE(F) qui retourne un objet de type Booléen.
Retourne Vrai si la file F est pleine et retourne Faux sinon.

Conditions :
ENFILER(F, e) est définie si, et seulement si, EST_PLEINE(F) = Faux.
DEFILER(F) est définie si, et seulement si, EST_VIDE(F) = Faux.

Exemple d'application de ce type abstrait :

On considère la suite d'instructions suivante :
F = CREER_FILE_VIDE()
ENFILER(F, 21)
ENFILER(F, 22)

ENFILER(F, 23)
N = DEFILER(F)
ENFILER(F, 24)
ENFILER(F, 25)
N = DEFILER(F)

Donner une représentation de la file F à chaque étape et le contenu de la variable N.

Représentation d'une file avec un tableau

On peut par exemple réaliser une file capable de contenir *n* éléments avec un tableau F[0 .. (*n* + 2)] pouvant contenir (*n* + 3) éléments :
●La première case du tableau (d'indice 0) contient l'indice de la tête de la file.
●La deuxième case du tableau (d'indice 1) contient l'indice de la queue de la file, c'est-à-dire la prochaine case disponible pour la queue.
●La troisième case du tableau (d'indice 2) contient le nombre d'éléments présents dans la file, c'est-à-dire la taille de la file.
●Les cases suivantes du tableau (d'indices 3 à *n* + 2) contiennent les éléments de la file ou sont vides.

Remarques :
Si (taille == 0) la file est vide et si (taille == *n*) la file est pleine.
A chaque fois qu'on enfile un élément, on augmente la taille d'une unité ainsi que la queue.
A chaque fois qu'on défile un élément, on diminue la taille d'une unité et on augmente la tête d'une unité.
Dès que les indices de tête ou de queue dépassent la longueur du tableau, ils repartent au début du tableau (on appelle cela un tableau avec une gestion circulaire).

Exemple de représentation pour *n* = 5 (5 sera la taille maximale de la file) :

Après enfilage de 8, 3 et 5 (la file contient (8, 3 , 5)) :

Tête	Queue	Taille					
3	**6**	**3**	8	3	5		
0	1	2	3	4	5	6	7

Après un défilage (la file contient (3 , 5))

Tête	Queue	Taille					
4	**6**	**2**	8	3	5		
0	1	2	3	4	5	6	7

Après un défilage supplémentaire (la file contient (5)) :

Tête	Queue	Taille					
5	**6**	**1**	8	3	5		
0	1	2	3	4	5	6	7

Après enfilage de 10, puis 12 (la file contient (5, 10, 12)) :

Tête	Queue	Taille					
5	**3**	**3**	8	3	5	10	12
0	1	2	3	4	5	6	7

L'indice de Queue devrait être 8, mais comme on dépasserait le dernier indice possible du tableau, on revient au début du tableau à la première case "vide". On sait que c'est possible car la taille de la liste (3) est inférieure à 5 (taille maximale de la file).

Après trois défilages successifs , la file est vide :

Tête	Queue	Taille					
3	**3**	**0**	8	3	5	10	12
0	1	2	3	4	5	6	7

Là aussi, l'indice de Tête devrait être 8, mais comme on dépasserait le dernier indice possible du tableau, on revient au début du tableau : 8 – 5 = 3.
Maintenant, tout se passe comme si la file venait d'être créée vide.

Remarque : on a toujours (**Queue** – **Tête**) = **Taille** **(modulo *n*)**.

Exercices

Exercice 1

On donne la séquence d'instructions suivantes :

```
F = CREER_FILE_VIDE()
ENFILER(F, 4)
ENFILER(F, 1)
ENFILER(F, 3)
N = DEFILER(F)
ENFILER(F, 8)
ENFILER(F, 9)
N = DEFILER(F)
```

1) Donner le contenu de la file F à chaque étape, en la représentant sous forme d'un tuple, et le contenu de la variable N.
2) On décide de représenter la file F par un tableau de 7 éléments, comme vu plus haut dans le cours.
a) Quelle est la taille maximale de la file ?
b) Représenter le contenu du tableau à chaque étape.

Exercice 2

On suppose que l'on a déjà une file F1 qui contient les éléments suivants saisis dans l'ordre alphabétique : F1 = ('A', 'B', 'C', 'D', 'E').
1) Quel est l'élément issu d'un défilage de F1 ?
2) Proposer une séquence d'instructions utilisant deux piles P1 et P2, permettant la saisie d'affilée des 5 éléments 'A', 'B', 'C', 'D' et 'E', puis de sortir ces éléments comme s'ils sortaient d'une file.

Exercice 3

Implémenter en Python les opérations classiques sur les files à l'aide d'un tableau que l'on "simulera" à l'aide d'une liste. On supposera que le tableau a une taille fixe, par exemple 10. On gèrera ce tableau de façon circulaire : dès que les indices de tête ou de queue dépassent la longueur du tableau, ils repartent au début du tableau.

Exercice 4

Proposer une implémentation des opérations classiques de la file à l'aide des méthodes .pop() et .append() du type liste de Python. Une file vide sera représentée par la liste vide. On pourra se donner arbitrairement une taille maximale pour la file, initialisée dans une variable globale, par exemple MAX = 10.
On rappelle que, si F est une liste Python, L.pop(0) retourne l'élément d'indice 0 de F et le supprime de la liste.

Exercice 5 : Conversion d'une expression infixée en postfixée

On ne considère ici que les expressions numériques contenant les quatre opérations élémentaires (+, –, *, /).
On souhaite ici convertir une expression arithmétique infixée (avec des parenthèses) en une expression postfixée (notation polonaise inversée (NPI), sans parenthèses).
Par exemple, l'expression infixée :

7 * (((13 + 22) – 15) / 5)

sera traduite en :

7 13 22 + 15 – 5 / *

Dans l'expression infixée, chaque opération entre deux opérandes sera nécessairement mise entre parenthèses, car on ne tiendra pas compte ici de la priorité des opérations les unes sur les autres.
Ainsi, une expression telle que :

3 * 2 + 4

sera écrite : (3 * 2) + 4.

Le programme lit l'expression infixée comme une suite de caractères, et range le résultat de la conversion dans une file qui s'affichera.

Ce programme utilise une pile (**pile_operateurs**) pour gérer les opérateurs, et une file (**file_expression**) pour conserver l'expression postfixée.

Les **valeurs numériques** lues sont directement rangées dans la file, alors que les **opérateurs** sont traités afin d'apparaitre dans la file *après* leurs opérandes. Ils sont d'abord empilés (mis de côtés) dans **pile_operateur**, puis dépilés pour être rangés dans la file, après que les deux opérandes sont enfilées. Cela se produit quand la parenthèse fermante est rencontrée. La file contient les différentes valeurs et les opérateurs.

Quand la fin de ligne de l'expression infixée est atteinte, tous les opérateurs qui restent encore dans la pile sont dépilés et rangés dans la file.

Pour afficher l'expression postfixée, on défile un par un tous les éléments de la file et on les concatène en les séparant par un espace.