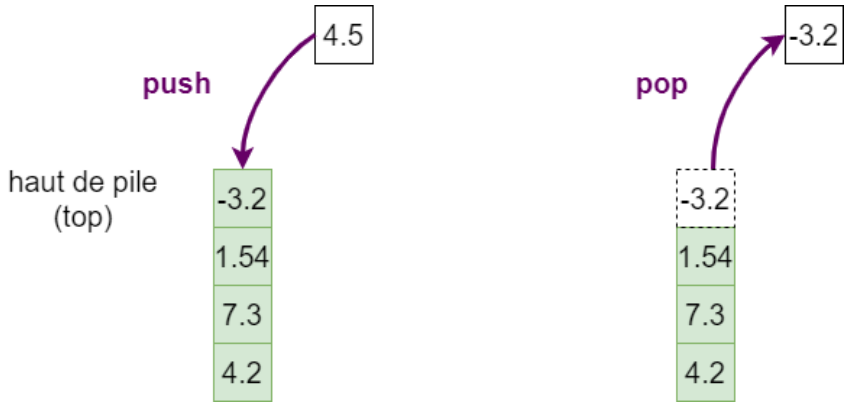
TP n°13 - Piles et files.

Définition 1 (Pile)

Une pile (*stack* en anglais) est une structure de données séquentielle fondée sur le principe « dernier arrivé, premier sorti » (en anglais LIFO pour *last in, first out*), ce qui veut dire qu'en général, le dernier élément ajouté à la pile est le premier à en sortir.



Les fonctionnalités nécessaires à la manipulation de piles sont les suivantes :

`stack_create` : créer un objet de type pile (constructeur)

`stack_push` : empiler un élément sur la pile (transformateur)

`stack_top` : renvoyer la valeur de l'élément situé en haut de la pile (accesseur)

`stack_pop` : renvoyer la valeur de l'élément situé en haut de la pile et dépiler cet élément (transformateur)

`stack_length` : renvoyer la taille de la pile (accesseur)

`stack_is_empty` : renvoyer vrai si la pile est vide, faux sinon (accesseur)

`stack_print` : afficher tous les éléments de la pile (du haut de la pile vers le bas) (accesseur)

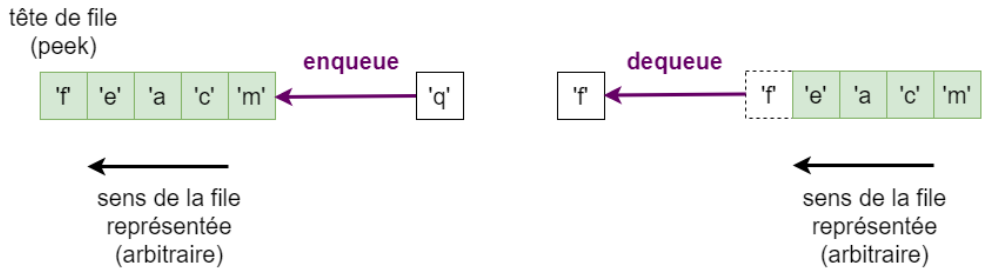
`stack_free` : détruire un objet de type pile (destructeur)

Exercice 1 (Pile implémentée avec un tableau).

1. Vous réfléchirez sur papier avec des dessins à la meilleur manière d'implémenter les opérations de manipulation de pile en utilisant un grand tableau, de sorte que votre implémentation soit la plus optimale possible en terme de complexité temporelle des transformateurs. Appelez moi pour valider vos choix d'implémentation.
2. Dans un fichier `NOM_stack_array.c`, définissez un type pile utilisant un grand tableau et implémentez toutes les opérations de manipulation décrites ci-dessus, en lien avec les observations faites à la question précédente et en respectant les noms imposés.

Définition 2 (File)

Une file (*queue* en anglais) est une structure de données séquentielle fondée sur le principe « premier arrivé, premier sorti » (en anglais FIFO pour *first in, first out*), ce qui veut dire qu'en général, l'élément le plus ancien est le premier à en sortir.



Les fonctionnalités nécessaires à la manipulation des files sont les suivantes :

`queue_create` : créer un objet de type file (constructeur)

`queue_enqueue` : ajouter un élément en bout de file (transformateur)

`queue_peek` : renvoyer la valeur de l'élément de tête de file (accesseur)

`queue_dequeue` : renvoyer la valeur de l'élément de tête de file et enlever cet élément de la file (transformateur)

`queue_length` : renvoyer la taille de la file (accesseur)

`queue_is_empty` : renvoyer vrai si la file est vide, faux sinon (accesseur)

`queue_print` : afficher tous les éléments de la file (de la tête jusqu'au bout) (accesseur)

`queue_free` : détruire un objet de type file (destructeur)

Exercice 2 (File implémentée avec des maillons chaînés).

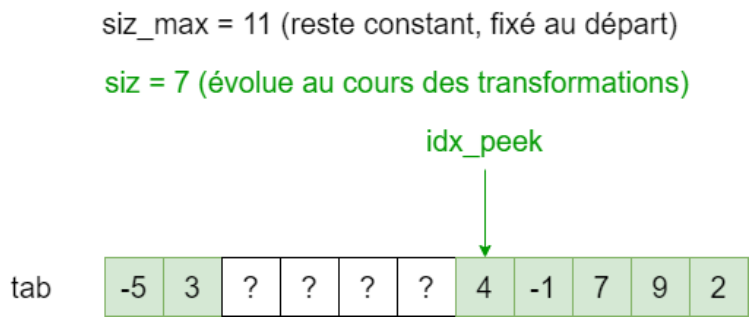
1. Réfléchir sur papier à la définition d'un type file utilisant des maillons chaînés. Quelles informations faut-il stocker pour que l' implémentation soit la plus optimale possible en terme de complexité temporelle des accesseurs et transformateurs ?
2. Dans un fichier `NOM_queue_cells.c`, définissez le type `queue` à la lumière de l'étude effectuée à la question précédente et implémentez une bibliothèque de fonctions permettant de manipuler des files.
   Vous implémenterez toutes les opérations de manipulation décrites ci-dessus.

Exercice 3 (File implémentée avec un tableau circulaire (*ring buffer*)).

1. Réfléchissez sur papier à une implémentation de file avec un grand tableau. Peut-on utiliser la même astuce que pour l'implémentation de pile ? Quelle est la conséquence sur l'efficacité de votre implémentation ?
2. Il existe une manière très élégante de régler ce problème. On range toujours les éléments de la file dans un grand tableau de taille `siz_max` fixée au départ, mais en utilisant ce tableau de manière circulaire (on parle de *ring buffer*) :

   La suppression de l'élément en tête de file va consister à incrémenter un indice `idx_peek` localisant la case du tableau contenant l'élément de tête.

   L'ajout d'un élément en queue de file se fait dans la case située `siz` cases plus loin que la case d'indice `idx_peek`, en considérant que le grand tableau de taille `siz_max` est circulaire : quand on arrive à la dernière case du grand tableau, on repart à la case d'indice 0.



   a. Dessinez sur papier les opérations `enqueue` et `dequeue` sur de petits exemples (avec des valeurs petites de `siz_max`). Prenez le temps de bien comprendre comment évoluent les valeurs `idx_peek` et `siz` lors des transformations de la liste.

   b. Écrire précisément l'opération arithmétique permettant de calculer l'indice de la case où sera ajouté le prochain élément

   c. Dans un fichier `NOM_queue_ringbuffer.c`, définissez un type et implémentez une bibliothèque de fonctions permettant de manipuler des files implémentées avec un *ring buffer*.
   Vous implémenterez toutes les opérations de manipulation décrites ci-dessus et respecterez rigoureusement tous les noms imposés par cet énoncé.
   Pour gagner un peu de temps, vous pourrez repartir du fichier `listes_tableau.c` corrigé du TP précédent, disponible sur Moodle, et le modifier.