Université de Liège

INFO0947

Compléments de Programmation

Un exercice dont vous êtes le Héros · l'Héroïne.

Files – Utilisation des Files

Simon Liénardy Benoit Donnet

23 avril 2020

Préambule

Exercices

Dans ce « TP dont vous êtes le héros », nous vous proposons de suivre pas à pas la résolution d'un exercice sur la manipulation des files.

Il est dangereux d'y aller seul [1] !

Partagez vos commentaires, questions, solutions alternatives sur le forum eCampus. N'hésitez jamais !

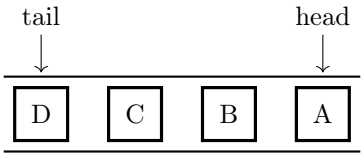
1. Référence vidéoludique bien connue des Héros.

8.1 Commençons par un Rappel

Si vous avez déjà lu ce rappel, vous pouvez directement atteindre le point de l'énoncé de l'exercice (Sec. 8.2).

8.1.1 File ?

Une *File* (« Queue ») est une structure de données linéaire basée sur le principe *FIFO* (FifoIn-FirstOut). Cela signifie qu'on ne peut ajouter de l'information qu'en fin de File mais qu'on retire un élément en début de File. Un exemple de File avec quatre valeurs est donnée ci-dessous :



tail indique la fin de la File, tandis que *head* indique son début. On ajoute donc des éléments dans la File au niveau de *tail* et on retir au niveau de *head*.

8.1.2 Opérations

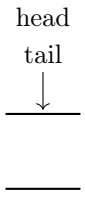
La spécification abstraite partielle [2] d'une File est donnée ci-dessous :

```
Type:
  Queue
Utilise:
  Boolean, Element
Opérations:
  empty_queue: → Queue
  is_empty: Queue → Boolean
  enqueue: Queue × Element → Queue
  dequeue: Queue → Queue
  head: Queue → Element
```

La File utilise définit cinq opérations. Nous détaillons, ci-dessous, l'effet de chacune de ces opérations.

8.1.2.1 Créer une File

L'opération *empty_queue* permet de créer une File vide. C'est donc un constructeur. Si on applique l'opératon *empty_queue*, on obtient :



La File est vide et les indicateurs *tail* et *head* pointe au même endroit.

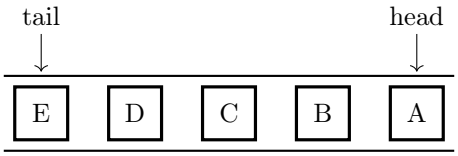
2. Pour compléter la spécification abstraite, il faut rajouter les Préconditions et les Axiomes (soit la partie sémantique de la spécicification abstraite). Voir Chapitre 8, Slide 9 → 10.

8.1.2.2 Vérifier si une File est Vide

L'opération *is_empty* permet de vérifier si une File est vide. Il s'agit donc d'un observateur.

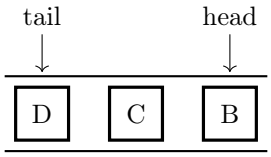
8.1.2.3 Ajouter un Elément dans une File

L'opération *enqueue* permet d'ajouter un élément en fin de File. Il s'agit donc d'un transformateur. Si on applique l'opération *enqueue*(E) à la File illustrée en début de rappel, on obtenons ceci :



8.1.2.4 Retirer un Elément d'une File

L'opération *dequeue* permet d'enlever un élément en début de File. Il s'agit donc d'un transformateur, puisque la File est modifiée après cette opération. Attention, c'est une opération partielle : impossible d'enlever une valeur à une File vide. Si on applique l'opération *dequeue* à la File illustrée en début de rappel, on obtient la File suivante :



8.1.2.5 Inspecter la Tête de File

L'opération *head* permet de connaître l'élément se situant en début de File sans le supprimer (contrairement à *dequeue*). Cette opération est donc bien un observateur. Attention, à l'instar de *dequeue*, *head* est une opération partielle : impossible de retourner le début d'une File vide. Si on applique l'opération *head* à la File illustrée en début de rappel, on obtient la valeur A, la File étant inchangée.

8.1.3 Interface (`queue.h`)

L'implémentation d'une File a peu d'importance dans le cadre de cet exercice. Nous pouvons utiliser directement la File grâce à l'interface (i.e., header) suivante :

```
#ifndef __QUEUE__
#define __QUEUE__
#include "boolean.h"

typedef struct queue_t Queue;

Queue *empty_queue(void);

Boolean is_empty(Queue *q);

Queue *enqueue(Queue *q, void *e);

Queue *dequeue(Queue *q);

void *head(Queue *q);

#endif
```

8.2 Énoncé

Soit le bout de code C suivant manipulant une File.

```
1  int i;
2  Queue *q = empty_queue();
3
4  for(i=1; i<=8; i++)
5     q = enqueue(q, i);
6
7  q = dequeue(q);
8  q = dequeue(q);
9
10 for(i=0; i<3; i++)
11    q = enqueue(q, i);
12
13 q = dequeue(q);
14
15 q = enqueue(q, 10);
16 q = enqueue(q, 12);
```

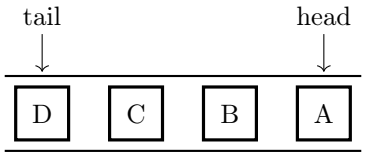
Supposons que la File utilisée dans le bout de code indiqué supra soit implémentée de manière dynamique à l'aide de pointeurs (i.e., liste chaînée avec pointeur de début et de fin – cfr. Chapitre 8, Slides 30 → 36).

Quel sera le contenu de la file `q` après l'exécution du bout de code ?

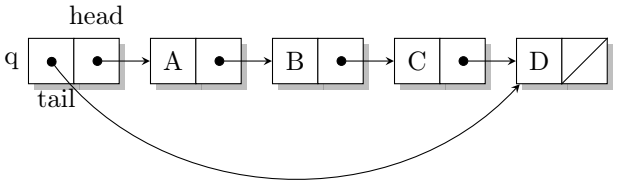
Indiquez les différentes étapes de l'évolution de `q`.

8.2.1 Méthode de résolution

L'exercice implique que la File soit implémentée sous la forme d'une liste chainée avec pointeur de début et de fin). Ainsi, la file suivante :



sera représentée par la liste suivante (avec `q` de type `Queue *` – voir le rappel, Sec. 8.1) :



Nous allons passer en revue chacune des lignes du code donné dans l'énoncé. En particulier, nous représenterons à chaque fois la File de manière conceptuelle et en liste chainée avec pointeur de début/fin.

Les étapes de résolutions sont les suivantes :

Initilisation	Ligne 2	Sec. 8.3.1
Boucle 1	Lignes 4 → 5	Sec. 8.3.2
Séquence de Retraits	Lignes 7 → 8	Sec. 8.3.3
Boucle 2	Lignes 10 → 11	Sec. 8.3.4
Retrait	Ligne 13	Sec. 8.3.5
Séquence d'Ajouts	Lignes 15 → 16	Sec. 8.3.6

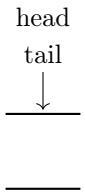
8.3 Résolution

Avant d'entamer la résolution de l'exercice, il est impératif que vous vous sentiez à l'aise avec la manipulation des Files. Si ce n'est pas le cas, jetez un œil au rappel.

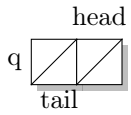
Une fois n'est pas coutume, la résolution de l'exercice est linéaire. Voici la table des matières :

8.3.1 Initilisation

Cette étape permet de créer une File. Par définition, une File nouvellement créée est vide. Soit :



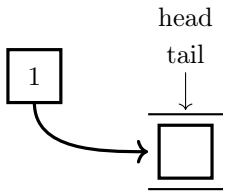
Ce qui donne, pour la structure de données concrète :



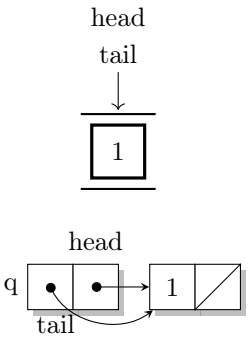
8.3.2 Boucle 1

La boucle rajoute des valeurs entières ($\in [1;8]$, dans l'ordre croissant) dans la File. Regardons, itération par itération, ce que cela donne.

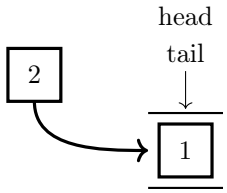
Premier tour de boucle : il s'agit d'ajouter la valeur 1 à la File. Soit :



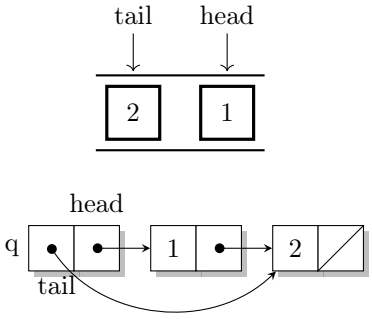
Ce qui donne :



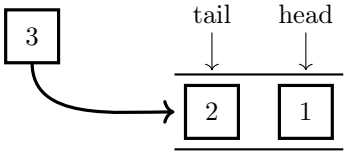
Deuxième tour de boucle : il s'agit d'ajouter la valeur 2 à la File. Soit :



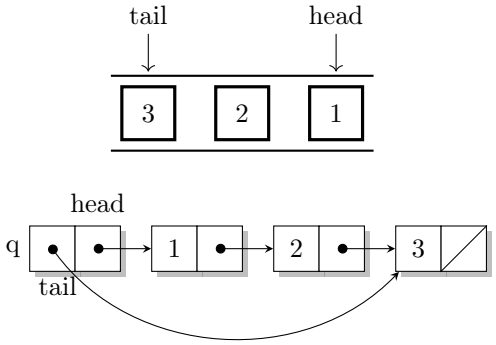
Ce qui donne :



Troisième tour de boucle : il s'agit d'ajouter la valeur 3 à la File. Soit :

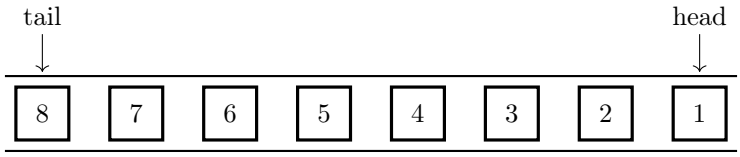


Ce qui donne :

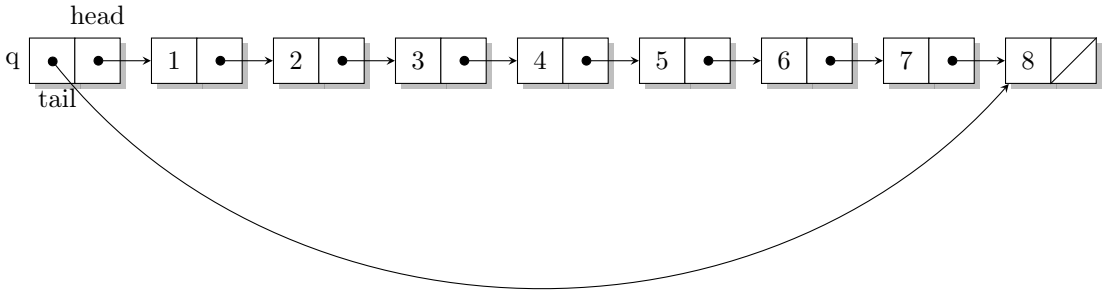


etc ...

A la sortie de la boucle, nous avons la situation suivante :

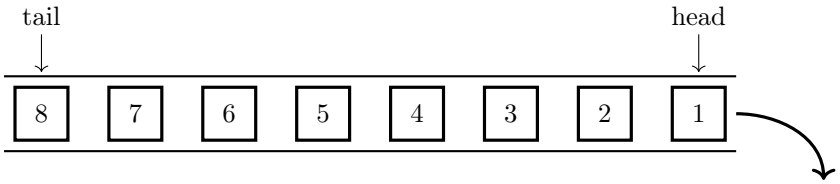


Ce qui donne, dans la structure de données concrètes :

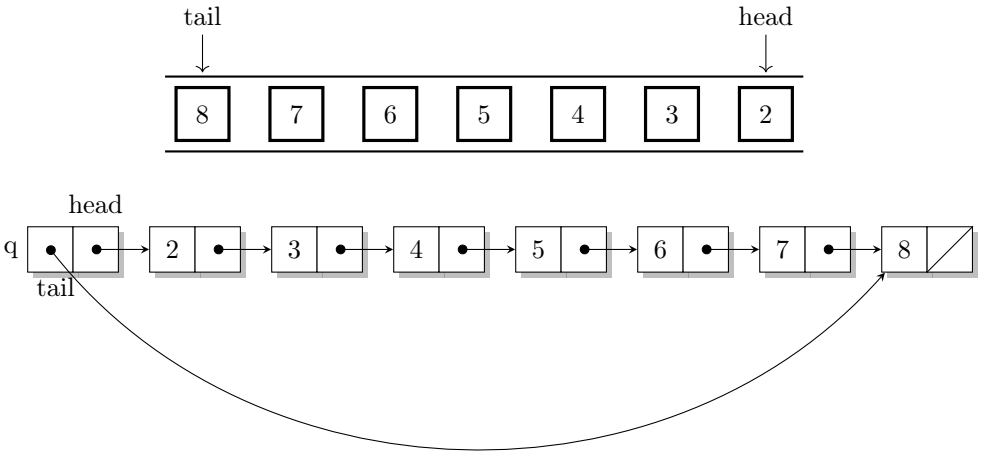


8.3.3 Séquence de Retraits

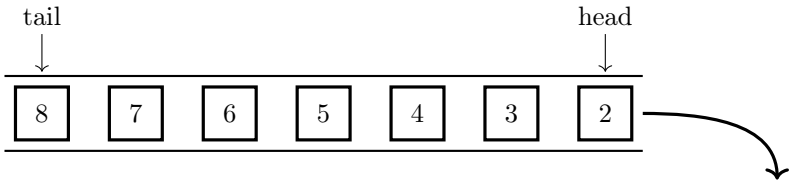
Ligne 7 : il s'agit de retirer la valeur en tête de File. Soit :



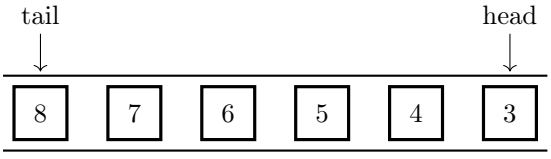
Ce qui donne :

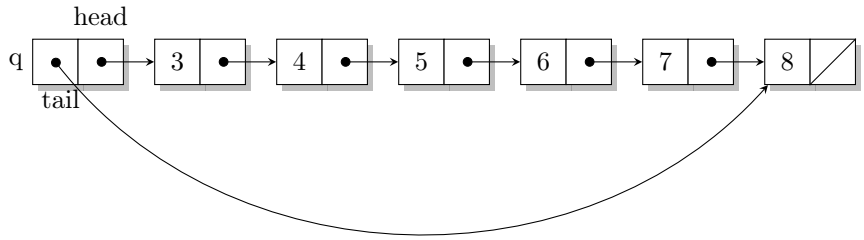


Ligne 8 : il s'agit de retirer la valeur en tête de File. Soit :



Ce qui donne :

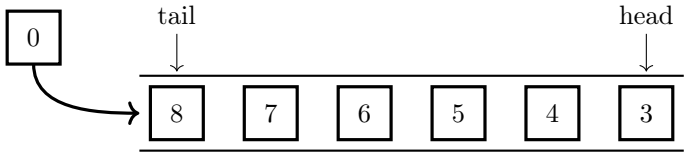



head
q
3
4
5
6
7
8
tail

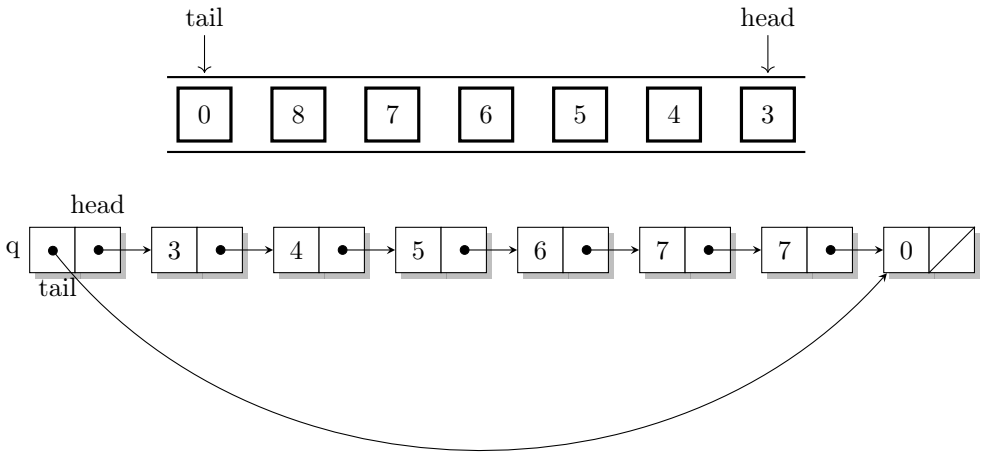
8.3.4 Boucle 2

La boucle rajoute des valeurs entières ($\in [0;2]$, dans l'ordre croissant) dans la File. Regardons, itération par itération, ce que cela donne.

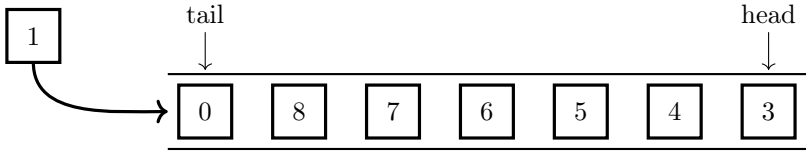
Premier tour de boucle : Il s'agit d'ajouter la valeur 0 à la File. Soit :



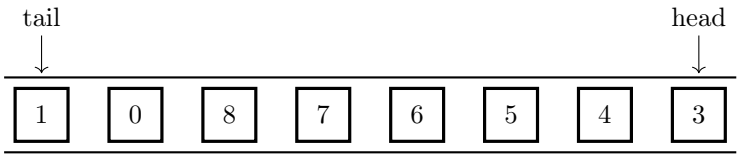
Ce qui donne :

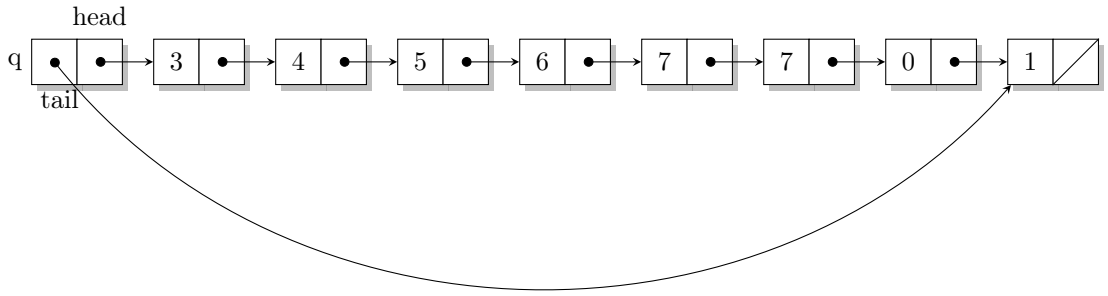


Deuixème tour de boucle : Il s'agit d'ajouter la valeur 1 à la File. Soit :

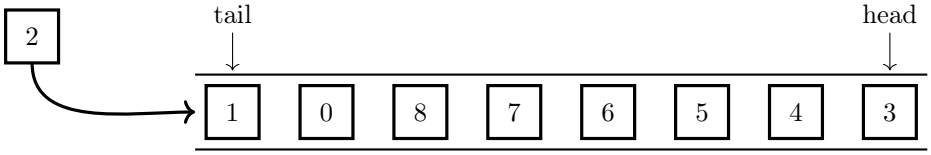


Ce qui donne :

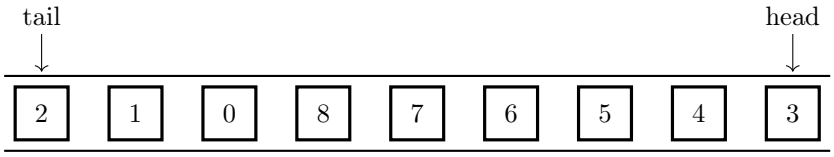


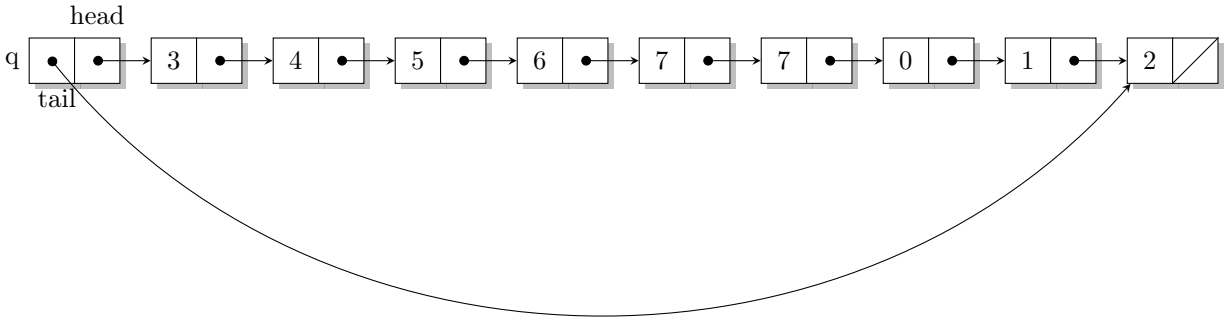


Troisième tour de boucle : Il s'agit d'ajouter la valeur 2 à la File. Soit :



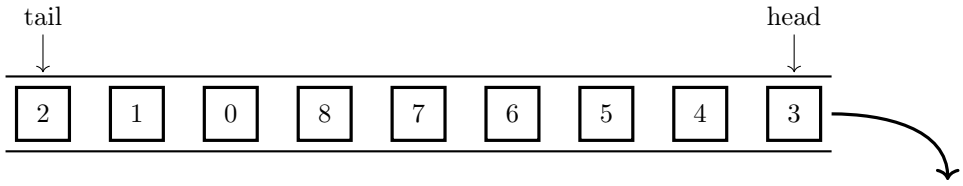
Ce qui donne :



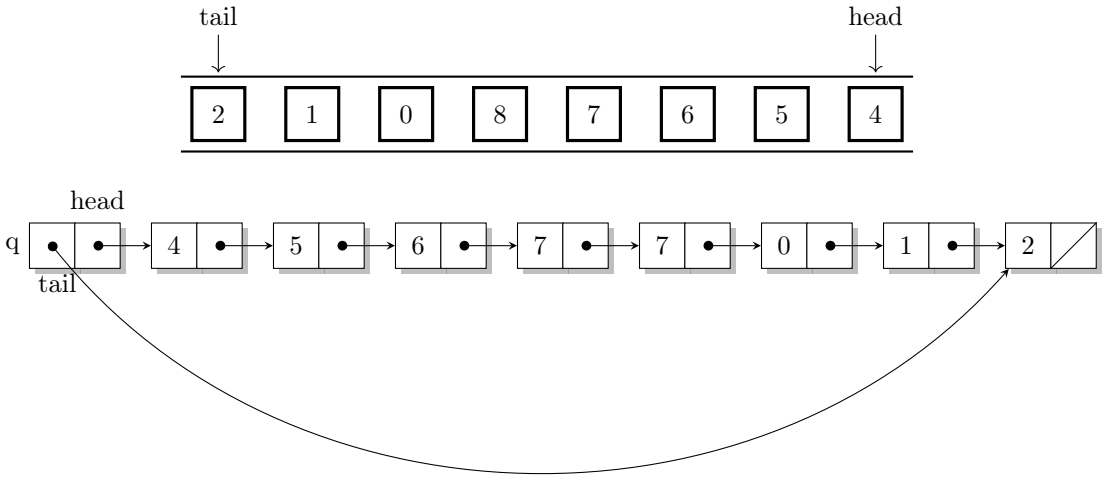


8.3.5 Retrait

Il s'agit de retirer la valeur en tête de File. Soit :

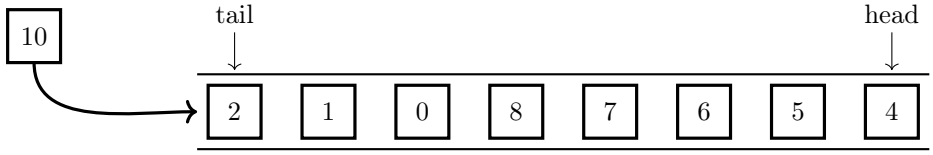


Ce qui donne :

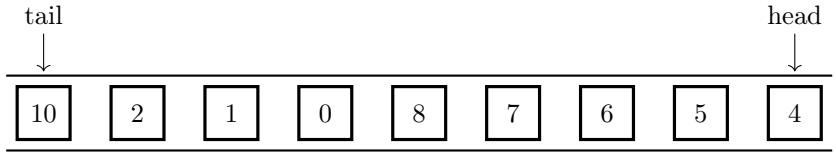


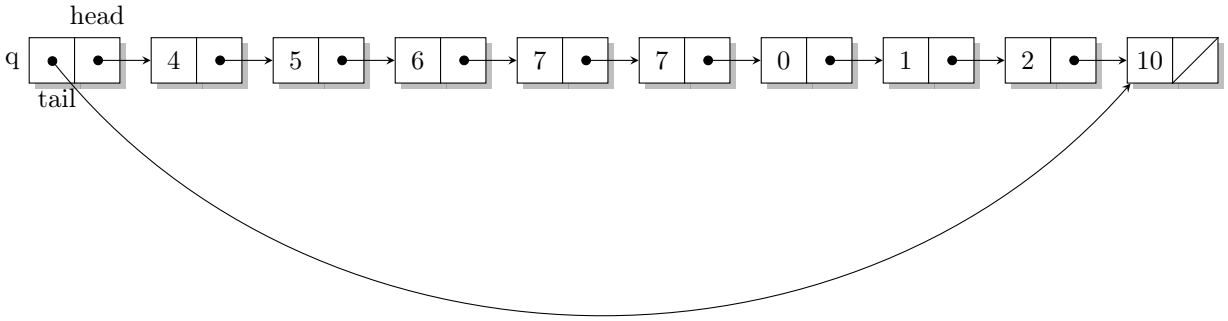
8.3.6 Séquence d'Ajouts

Ligne 15 : il s'agit d'ajouter la valeur 10 à la File. Soit :

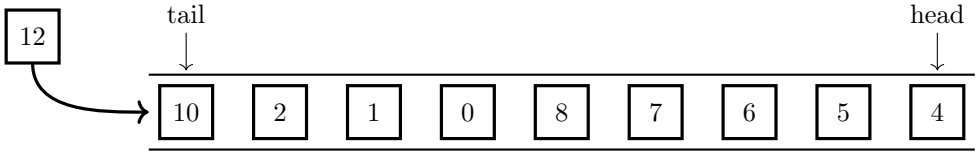


Ce qui donne :

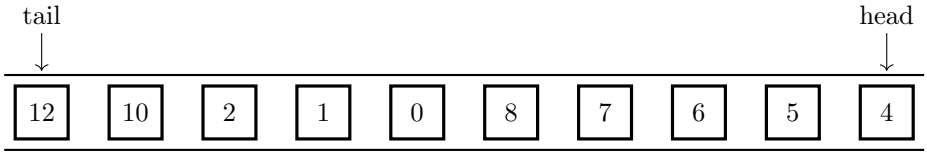


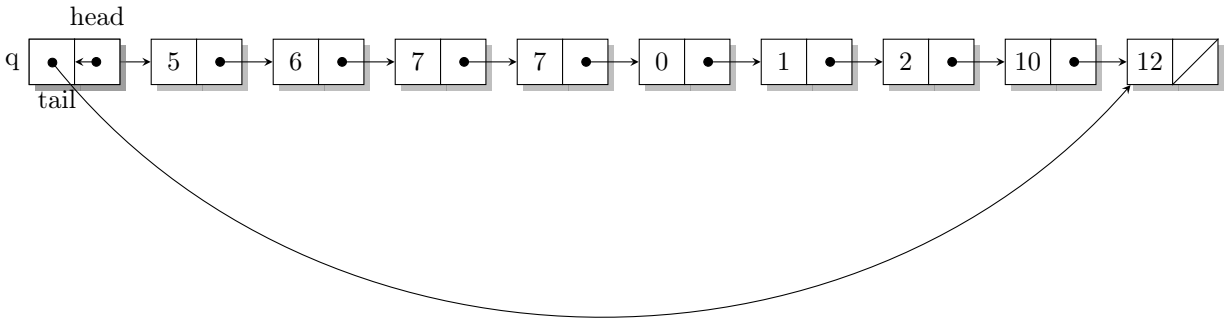


Ligne 16 : il s'agit d'ajouter la valeur 12 à la File. Soit :



Ce qui donne :




head
q
tail
5
6
7
7
0
1
2
10
12