Données 16 - Le type abstrait File – Queue - FIFO

I – Qu'est-ce que le type abstrait File ?

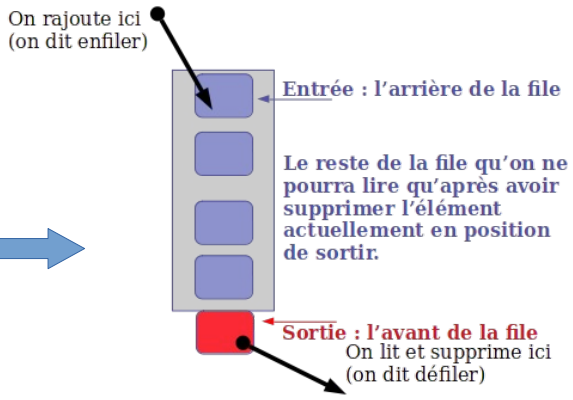
1.1 Idée générale

Une **File** est un **type linéaire de données** ayant les propriétés suivantes :

- elle est composée d'une séquence _____
- on s'impose strictement d'agir ainsi

1. **insérer un élément** du côté de l'entrée **(Arrière / Back)** se nomme **l'enfilement**

2. **supprimer l'élément** du côté de la sortie (**Avant / Front)** se nomme **le défilement**



Exemple : si on insère d'abord 0, puis 5 puis 10 puis 15 obtient :

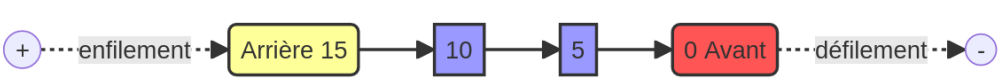
FIFO :_____

1.2 Interface du type abstrait FILE

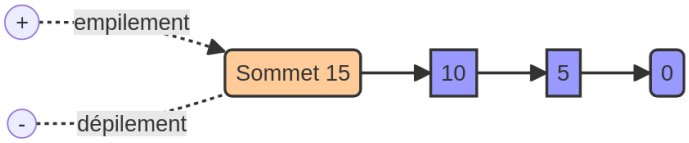
Prototype de la fonctions d'interface	Description (cas non mutable)
`nouvelleFile() -> File`	On crée une nouvelle file vide `()`. En anglais, cela pourrait donner `newQueue`.
`estFileVide(f:File) -> bool`	Renvoie un booléen qui vaut True si la file f transmise est une file vide. En anglais, `isEmpty`
`enfiler(f:File, x:Element) -> File NON VIDE`	Renvoie une nouvelle pile où l'élément arrière est maintenant l'élément x. En anglais, *`enqueue.`*
`defiler(f:File NON VIDE) -> File`	Renvoie une nouvelle file après avoir supprimé l'avant. En anglais, *`dequeue.`*
`lireAvant(f:File NON VIDE) -> Element`	Renvoie la valeur en avant de la file f. Attention, on ne modifie pas la file. En anglais, *`front.`*

1.3 Différence fondamentale entre Pile et File

Une File a deux extrémités (Arrière et Avant)



Une Pile n'a qu'une extrémité : le Sommet



Exercices 01° Fournir les contenus successifs de la pile **`f1`** après exécution de chacune des instructions

```
1  f1 ← nouvelleFile()
2  f1 ← enfiler(f1, 5)
3  f1 ← enfiler(f1, 7)
4  x ← lireAvant(f1)
5  f1 ← enfiler(f1, x)
6  f1 ← enfiler(f1, 4)
7  f1 ← defiler(f1)
```

Exercices 02° On veut stocker séquentiellement 1, 2, 3 dans une file. Donner les instructions à utiliser et fournir la représentation finale de la File. Que va renvoyer la fonction lireAvant ? La file est-elle modifiée après cette lecture ?

1.4 Type mutable

- **`enfiler(f:File, x:Element) -> None`**
  On rajoute un élément à l'arrière, sur place
- **`defiler(p:File NON VIDE) -> Element`**
  on supprime l'avant en modifiant la file en place et on renvoie l'élément qu'on vient de sortir de la file

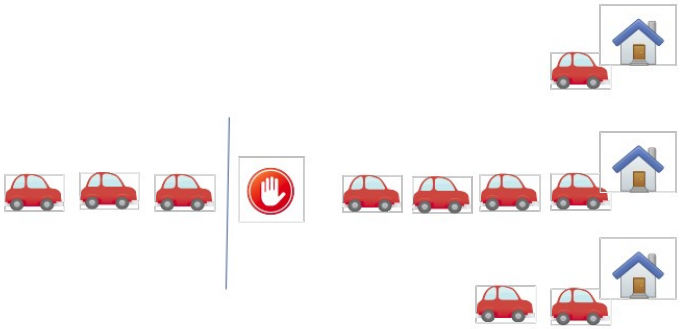
Exercices 03° Fournir les contenus successifs de la file **`f3`** après exécution de chacune des instructions

```
1  f3 ← nouvelleFile()
2  enfiler(f3, 5)
3  enfiler(f3, 7)
4  x ← defiler(f3)
5  enfiler(f3, x)
6  enfiler(f3, 4)
7  defiler(f3)
```

1.5 Fonctions optionnelles construites à partir des primitives

Prototype de la fonctions d'interface	Description (cas non mutable)
`taille``(f:File) -> int`	On renvoie le nombre d'éléments dans la File. Cela peut être pratique pour éviter de dépasser la taile limite
`vider``(f:File) -> File`	On vide la file. Pour une file immuable, on renvoie donc une file vide. Cela peut être utile pour signaler qu'on se sépare d'éléments non traités. En anglais, clear.
`mettreEnAttente``(f:File NON VIDE) -> File NON VIDE`	On défile l'avant et on enfile à nouveau la valeur dans la File

Exercices 04° Imaginer un algorithme permettant de gérer trois guichets f1, f2 et f3 d'un péage routier. Les clients font la queue physiquement et à partir d'un endroit un afficheur doit leur dire de se diriger vers un guichet particulier 1, 2 ou 3 en fonction de la file d'attente aux trois guichets.



II – Choix d'utilisation d'une FILE

Exercices 05° Même énoncé que la leçon 15 en replaçant empilement par enfilement…

III – Choisir sa structure linéaire