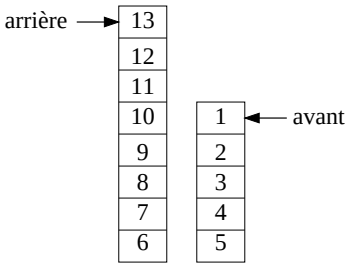
Files d'attente et suite de Hamming

1) Implémentation des files d'attente
Une file d'attente est liste dont on limite l'usage aux opérations suivantes :

- la liste est-elle vide ?
- extraire la tête de la liste
- ajouter un élément en queue de liste

On rencontre souvent des files d'attente dans la vie courante, par exemple à la caisse d'un supermarché ou à un feu rouge. En informatique une file d'attente permet de stocker des informations qui ne peuvent être traitées de suite, mais qui devront l'être en respectant leur ordre d'arrivée dans la file (structure de données FIFO : *first in, first out*).

On peut réaliser ces opérations avec le type liste chaînée fourni en standard par Caml, mais l'opération « insertion en queue » a une mauvaise complexité sur ce type de listes, aussi utilise-t-on des structures de données mieux adaptées aux fonctionnalités des files d'attente. Dans ce TP on implémentera une file d'attente par un couple de listes chaînées : l'avant de la file, classée par ordre d'arrivée et l'arrière, classée par ordre inverse d'arrivée.



Les nouveaux arrivants seront placés en tête de la liste arrière, les premiers arrivés seront extraits en tête de la liste avant. Lorsque la liste avant est épuisée on retourne la liste arrière, on la place à l'avant et on réinitialise la liste arrière à `[]`. Cette opération de retournement a une complexité linéaire en la taille de la liste arrière, donc le temps d'extraction du premier élément d'une file ainsi implémentée n'est pas constant, mais comme un élément de la file n'est retourné qu'au plus une fois, le temps cumulé de n extractions en tête la file est linéaire en n.

```
type 'a file = {
  mutable avant   : 'a list;
  mutable arrière : 'a list
};;
```

Dans la déclaration ci-contre les champs `avant` et `arrière` sont déclarés *mutables* de façon à pouvoir être modifiés sur place. Si `f` est une variable de type `'a file`, on modifie sa liste avant par une instruction du type :

```
f.avant <- qqch.
```

Recopier la définition ci-dessus. Programmer les fonctions suivantes :

– `nouv_file: unit -> 'a file`	créee une file initialement vide ;
– `est_vide : 'a file -> bool`	dit si une file est vide ;
– `premier  : 'a file -> 'a`	renvoit le premier élément de la file sans le retirer ;
– `retire   : 'a file -> 'a`	retire le premier élément de la file et le renvoit ;
– `ajoute   : 'a file -> 'a -> unit`	ajoute un élément en queue de la file.
– `longueur : 'a file -> int`	renvoit la longueur de la file.

Les fonctions `premier` et `retire` déclencheront des erreurs appropriées si la file passée en argument est vide.

2) La suite de Hamming
Un entier de Hamming est un entier naturel non nul dont les seuls facteurs premiers éventuels sont $2, 3, 5$. Le *problème de Hamming* consiste à énumérer les n premiers entiers de Hamming par ordre croissant. Pour celà, on remarque que le premier entier de Hamming est 1 et que tout autre entier de Hamming est le double, le triple ou le quintuple d'un entier de Hamming plus petit (ces cas n'étant pas exclusifs). Il suffit donc d'utiliser trois files d'attente, h_2, h_3 et h_5 contenant initialement le seul nombre 1 puis d'appliquer l'algorithme :

> *déterminer le plus petit des trois nombres en tête des files d'attente, soit x. Imprimer x, le retirer de chacune des files le contenant et insérer en queue de h_2, h_3 et h_5 respectivement $2x$, $3x$ et $5x$.*

a) Programmer cet algorithme et faire afficher les n premiers entiers de Hamming. Observer l'évolution des files d'attente à chaque étape.

b) L'algorithme précédent est très dispendieux car il place la plupart des entiers de Hamming dans les trois files alors qu'une seule suffirait. En effet, si x est un entier de Hamming divisible à la fois par 2, 3 et 5 alors x a

été placé dans h_2 au moment où l'on extrayait $x/2$, dans h_3 au moment où l'on extrayait $x/3$ et dans h_5 au moment où l'on extrait $x/5$. Modifier votre programme de sorte qu'un même entier de Hamming ne soit inséré que dans une seule des files d'attente.

c) Les entiers Caml sont limités à un milliard (environ) ce qui ne permet pas d'aller très loin dans la suite de Hamming. Pour pouvoir traiter des grands nombres on convient de représenter un entier de Hamming x par le triplet (a, b, c) tel que $x = 2^a 3^b 5^c$. Reprendre votre programme avec cette convention et calculer le millionnième entier de Hamming. Pour comparer deux entiers de Hamming x et y connus par leurs exposants il suffit de comparer les réels $\ln x$ et $\ln y$ (ln est noté `log` en Caml). On admettra que la précision des calculs sur les flottants est suffisante pour ne pas induire de comparaison erronée.

d) Déterminer *expérimentalement* la complexité mémoire de l'algorithme de recherche du n-ème nombre de Hamming. Cette complexité sera mesurée par la somme ℓ des longueurs des trois files h_2, h_3 et h_5. Pour cela on relèvera les valeurs de ℓ pour $n = 1000$, $n = 2000$, $n = 4000$, etc et on conjecturera une relation « simple » entre ℓ et n.