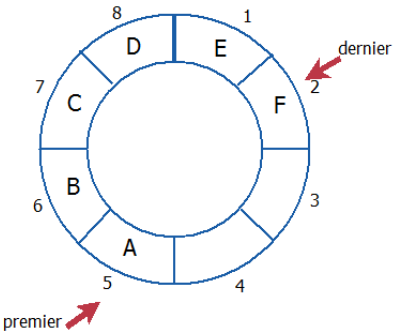
# 1 Implémentation statique / contiguë : dans un tableau

## 1.1 Le type

Nous avons besoin d'un tableau (à une dimension = vecteur) pour ranger les éléments.

Pour éviter les décalages lorsqu'il n'y a plus de places à la fin du tableau, on utilise celui-ci de manière circulaire : la position après la dernière est la première (ce dépassement est géré de façon extrêmement simple, il suffit d'utiliser un modulo la taille du tableau, soit 8 dans le cas présent). Une première solution consisterait à utiliser deux entiers `premier` qui représente la position du premier élément de la file et `dernier`, représentant la position du dernier élément.




Cela pose un problème : on ne sait plus distinguer les tests *file vide* et *vecteur plein*.
Une solution est de remplacer l'entier `dernier` par l'entier `longueur` qui nous permet :

- de savoir si la file est vide (`longueur` = 0) ;
- de savoir si le vecteur est plein (`longueur` = taille du tableau) ;
- de connaître la position du dernier (en l'ajoutant à `premier` modulo la *taille*).

Une file sera donc représentée par un enregistrement contenant le vecteur d'éléments et les deux entiers.

Exemple de déclaration algorithmique :

```
constantes
   Nbmax = 8                          /* Nombre maximum d'éléments */

types
   t_element = ...                    /* Un élément */
   t_elements = Nbmax t_element       /* Vecteur d'éléments */
   t_file = enregistrement            /* La file */
      t_elements elts
      entier premier, longueur
   fin enregistrement t_file

variables
   t_file f
```

## 1.2 Implémentation des opérations

- Tester `longueur = 0` permet de savoir si la file **est vide**.
- L'accès au **premier** élément est immédiat.
- Pour **défiler**, il suffit d'incrémenter (modulo la taille du vecteur) la position de `premier`.
- Pour **enfiler** un élément : sa position dans le vecteur est la somme `premier` + `longueur` (modulo la taille du vecteur).

À chaque modification, l'entier `longueur` est mis à jour.

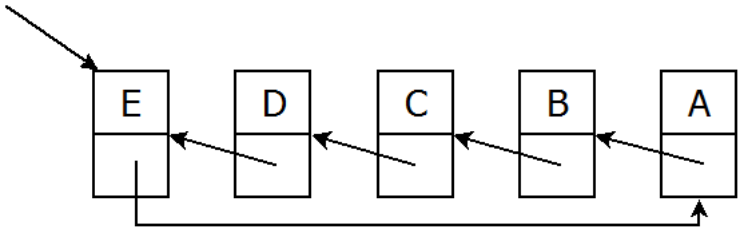
Toutes les opérations s'implémentent en *temps constant* : le nombre d'opérations exécutées ne dépend pas de la taille de la file.

# 2 Implémentation dynamique / chaînée

## 2.1 Le type

Dans ce cas, les éléments de la file sont chaînés entre eux. On peut utiliser deux pointeurs représentent les deux extrémités de la file.

Une autre solution est d'utiliser un chaînage circulaire. Le pointeur sur le dernier est alors suffisant our représenter la file : il suffit de suivre le lien `suivant` à partir du dernier pour déterminer le premier élément.




Exemple de déclaration algorithmique :

```
types
   t_element = ...                  /* un élément */
   t_pfile = ↑ t_sfile              /* la file */
   t_sfile = enregistrement          /* un enregistement */
      t_element elt
      t_pfile   suivant
   fin enregistrement t_sfile

variables
   t_pfile  F     /* F est le pointeur sur le dernier */
```

## 2.2 Implémentation des opérations

Le **file vide** sera représentée par le pointeur NUL.

Pour récupérer la valeur du **premier**, il suffit, sur une file non vide, de suivre le lien `suivant` à partir de `F` et d'accéder à l'élément pointé [1].

**Enfiler** un élément se fera de la manière suivante :
— On crée l'enregistrement pour le nouvel élément ;
— Si la file n'est pas vide : on le "chaîne" au reste : on l'insère entre le dernier et le premier ;
— il est le nouveau dernier : on déplace le pointeur `F` dessus.

Pour **défiler**, lorsque la file contient plus d'un élément :
— On garde un pointeur sur le premier (suivant du dernier) ;
— on déplace le pointeur `suivant` de `F` sur l'élément qui suit : il est le nouveau premier ;
— enfin, on libère la mémoire occupée par l'ancien premier.

Lorsque l'élément à défiler est le seul de la file, on le détruit et `F` devient NUL (file vide).

Ici aussi, toutes les opérations s'implémentent en temps constant.

1. Par abus de langage on parle de *l'élément pointé* pour *la valeur du champ elt de l'enregistrement pointé.*