# ITI 1521. Introduction à l'informatique II

**File :** concept

by

Marcel Turcotte

Version du 26 février 2020

# Préambule

# Préambule

## Aperçu

# Aperçu

**File : concept**

Nous nous intéressons à tous les aspects des files en programmation. Nous examinons plusieurs exemples de leur utilisation, notamment le partage de ressources, les algorithmes de simulation, et l'algorithme de fouilles en largeur. Nous verrons deux implémentations des files : soit à l'aide de tableaux circulaires ou à l'aide d'éléments chaînés.

**Objectif général :**

- Cette semaine, vous serez en mesure de décrire, appliquer, et implémenter une file.

# Préambule

## Objectifs d'apprentissage

# Objectifs d'apprentissage

- **Décrire** le concept de file en informatique.
- **Implémenter** une file à l'aide d'éléments chaînés.

**Lectures :**

- Pages 177–189 de E. Koffman et P. Wolfgang.

# Préambule

## Plan du module

# Plan

# Définitions

# Définition

Une **file** (*queue*) est un **type abstrait de données** linéaire tel que l'ajout de données se fait à une extrémité, l'**arrière** (*rear*) de la file, et le retrait à l'autre, l'**avant** (*front*).
Ces structures de données sont dites **FIFO** : *first-in first-out*.

`enqueue()` ⇒ Queue ⇒ `dequeue()`

Les deux **opérations de base** sont :

**enqueue :** l'**ajout** d'un élément à l'**arrière** de la file,

**dequeue :** le **retrait** d'un élément à l'**avant** de la file.

⇒ Les files sont donc des structures de données semblables aux files d'attente au supermarché, à la banque, au cinéma, etc.

# Type abstrait de Données (TAD) : File

```
public interface Queue<E> {
  void enqueue(E element);
  E dequeue();
  boolean isEmpty();
}
```

# Applications des files

- Gestion de **ressources partagées** :
  - Accès au processeur ;
  - Accès à un disque ou autres périphériques, ex. imprimante ;
- **Algorithmes** à base de files :
  - Simulations ;
  - Parcous en largeur («*breadth-first-search*»).

# Exemple simple d'utilisation

```
public class Test {
    public static void main(String[] args) {

        Queue<Integer> q;
        q = new LinkedQueue<Integer>();

        for (int i=0; i<10; i++) {
            q.enqueue(Integer.valueOf(i));
        }

        while (! q.isEmpty()) {
            System.out.println(q.dequeue());
        }

    }
}
```

- **Imprime ?** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

```
q = new LinkedQueue();
q.enqueue(a);
q.enqueue(b);
q.enqueue(c);
q.dequeue();
-> a
q.dequeue();
-> b
q.enqueue(d);
q.dequeue();
-> c
q.dequeue();
-> d
```

- Les éléments sont traités dans le même ordre qu'ils ont été insérés dans la file, ici l'élément **a** est le premier à joindre la file et c'est aussi le premier à quitter la file (***first-come first-serve***).

# Implémentation

# Implémentations

Tout comme les piles, il y a **deux familles** d'implémentations :

- **Listes chaînées** ;
- **À l'aide d'un tableau**.

# Implémentation à l'aide d'éléments chaînés

```
public class LinkedQueue<E> implements Queue<E> {

    public boolean isEmpty() { ... }
    public void enqueue(E o) { ... }
    public E dequeue() { ... }

}
```

# Implémentation

## Elem

# Implémentation à l'aide d'éléments chaînés

```
public class LinkedQueue<E> implements Queue<E> {

    private static class Elem<T> {
        private T value;
        private Elem<T> next;
        private Elem(T value, Elem<T> next ) {
            this.value = value;
            this.next = next;
        }
    }

    public boolean isEmpty() { ... }
    public void enqueue(E o ) { ... }
    public E dequeue() { ... }
}
```

# Implémentation

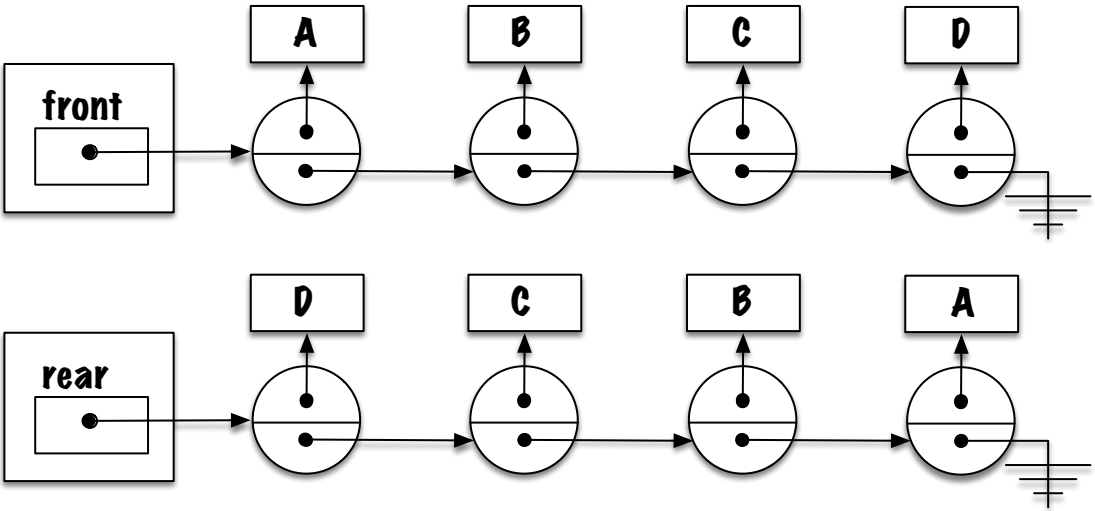
## Variables d'instance

# Implémentation à l'aide d'éléments chaînés

```
public class LinkedQueue<E> implements Queue<E> {

    private static class Elem<T> {
        private T value;
        private Elem<T> next;
        private Elem(T value, Elem<T> next) {
            this.value = value;
            this.next = next;
        }
    }

    private Elem<E> front; // rear?

    public boolean isEmpty() { ... }
    public void enqueue(E o ) { ... }
    public E dequeue() { ... }
}
```

# Implémentation à l'aide d'éléments chaînés

Quelle représentation vous semble **préférable** et **pourquoi** ?



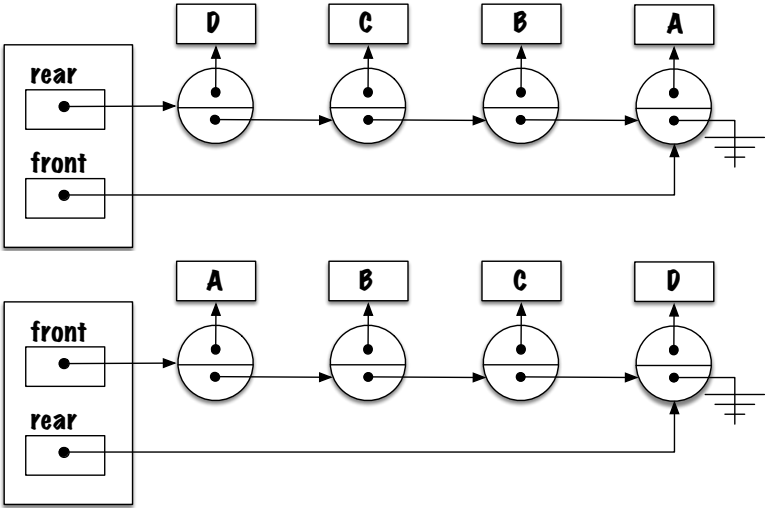
# Discussion

- Si nous choisissons la première implémentation, alors le **retrait** d'un élément sera facile (et **rapide**) mais l'**ajout** à l'arrière de la file sera difficile (et **lent**).
- L'autre implémentation ne fait qu'inverser les situations, le **retrait** devient **coûteux** alors que l'**ajout** est **rapide**.
- **Est-ce l'impasse ?**
- **Que faut-il pour faciliter le retrait ?**
- **Que faut-il pour faciliter l'ajout ?**

# Implémentation à l'aide d'éléments chaînés

# Implémentation à l'aide d'éléments chaînés

- Est-ce que ces deux implémentations seront aussi **efficaces** l'une que l'autre ?



# Discussion

- Quels seront les **impacts de cette modification** ?
  - La quantité de **mémoire** supplémentaire est **négligeable**.
  - L'**implémentation** des méthodes sera **plus complexe**.

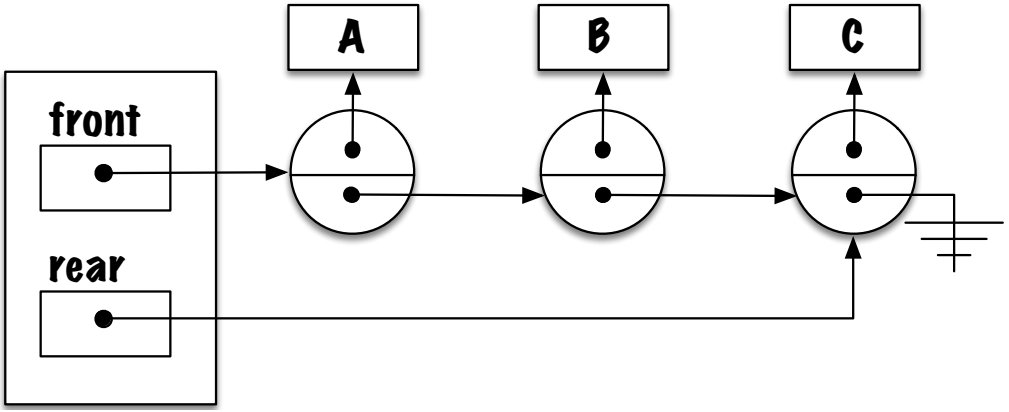
# Implémentation

## Méthodologie

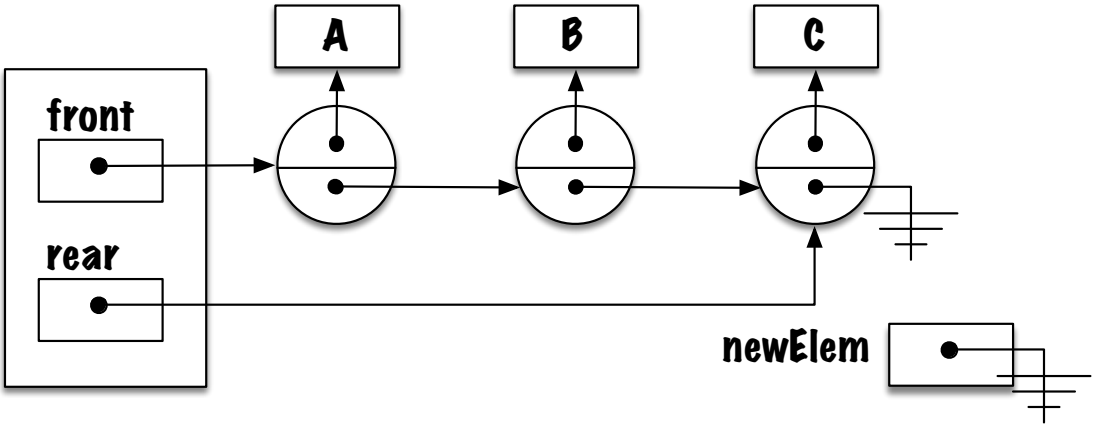
# Méthodologie

- Identifiez le **cas général** ainsi que le ou les **cas spéciaux**.
- **Cas général**, considérez un nombre suffisant d'éléments afin qu'il représente la majorité des cas.
- Les **cas spéciaux** sont les cas ou la stratégie employée pour le cas général ne s'appliquerait pas.
- Les files, les piles, et les listes **vides** ou ayant **un élément** sont souvent les cas spéciaux.

# Ajout d'un élément (cas général)

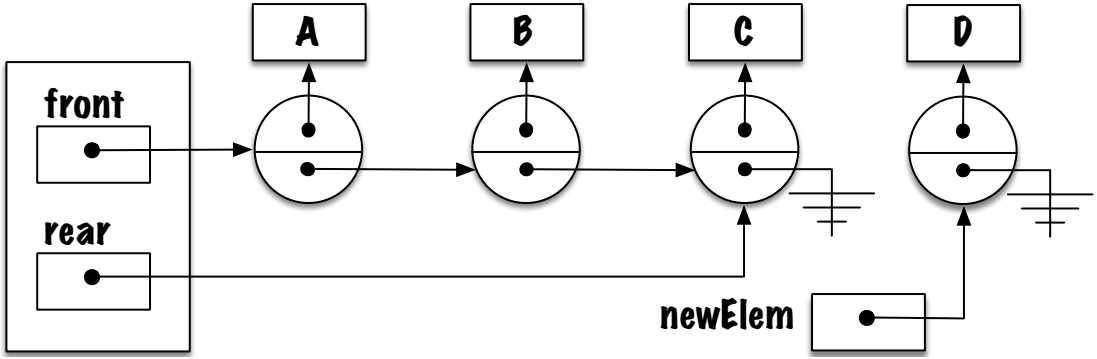


# Ajout d'un élément (cas général)

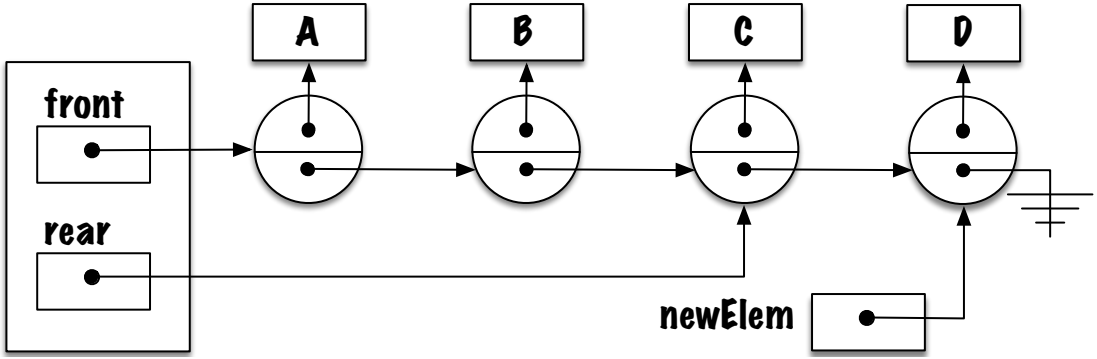



L'utilisation d'une **variable locale** facilitera le travail.

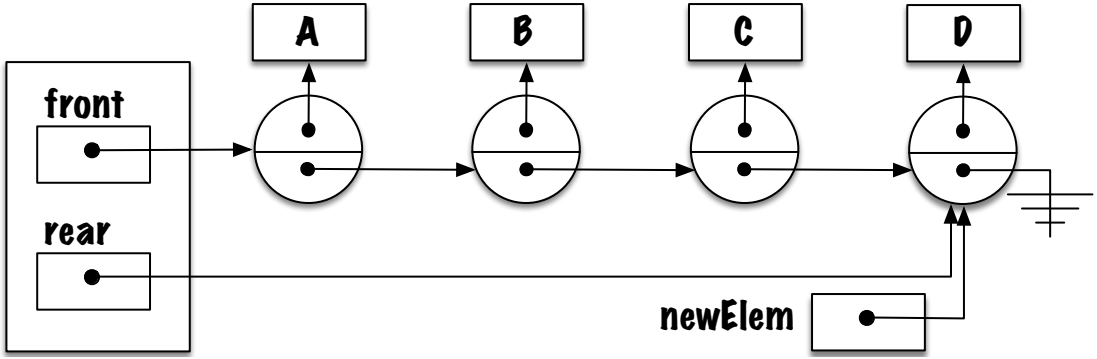
# Ajout d'un élément (cas général)



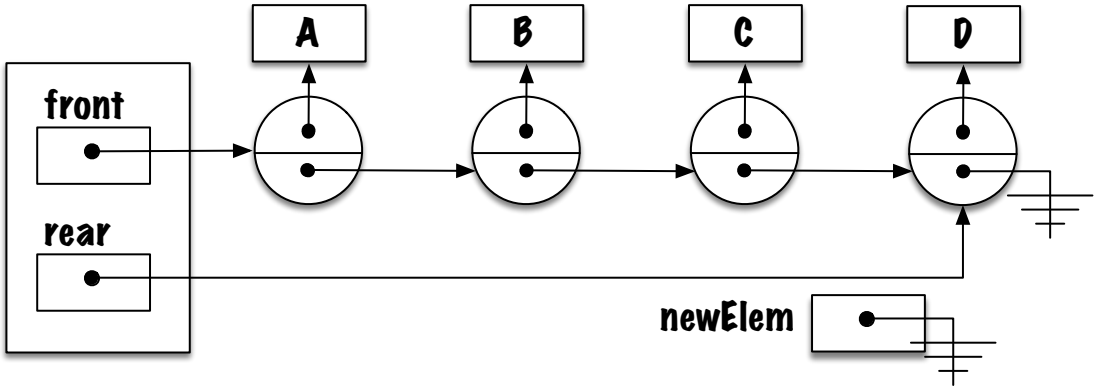
# Ajout d'un élément (cas général)



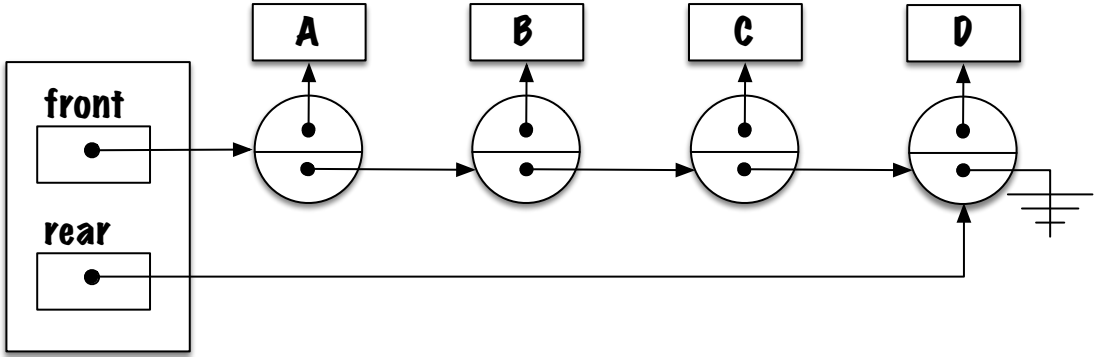
# Ajout d'un élément (cas général)



# Ajout d'un élément (cas général)



# Ajout d'un élément (cas général)

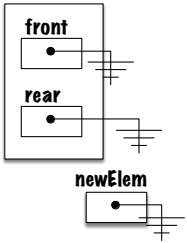


# Ajout d'un élément (cas spécial)

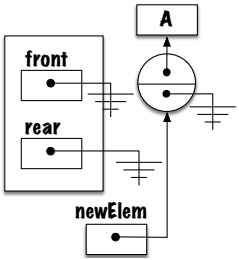
Dessinez le **diagramme de mémoire** représentant la **file vide**.

- Quelle **expression** permet d'identifier la **file vide** ?

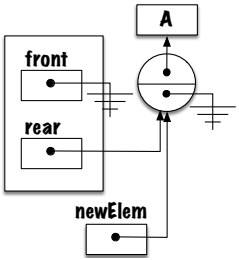
# Ajout d'un élément (cas spécial)



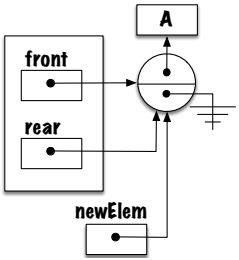
# Ajout d'un élément (cas spécial)



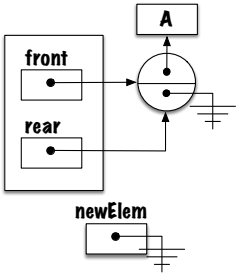
# Ajout d'un élément (cas spécial)



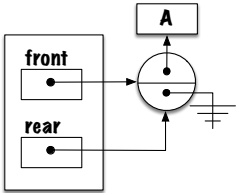
# Ajout d'un élément (cas spécial)



# Ajout d'un élément (cas spécial)



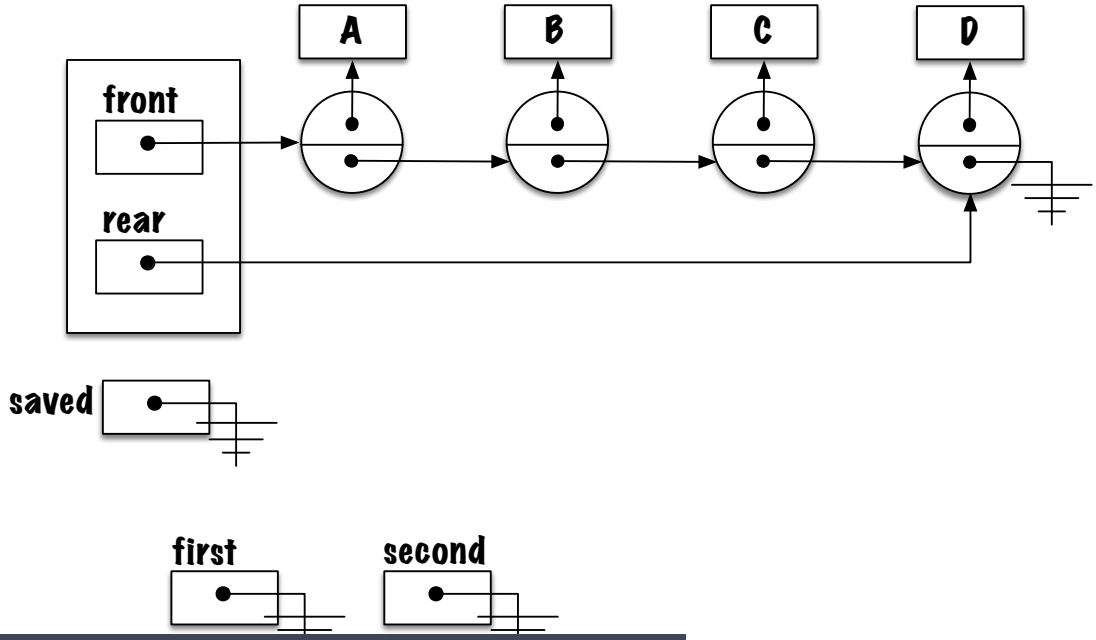
# Ajout d'un élément (cas spécial)



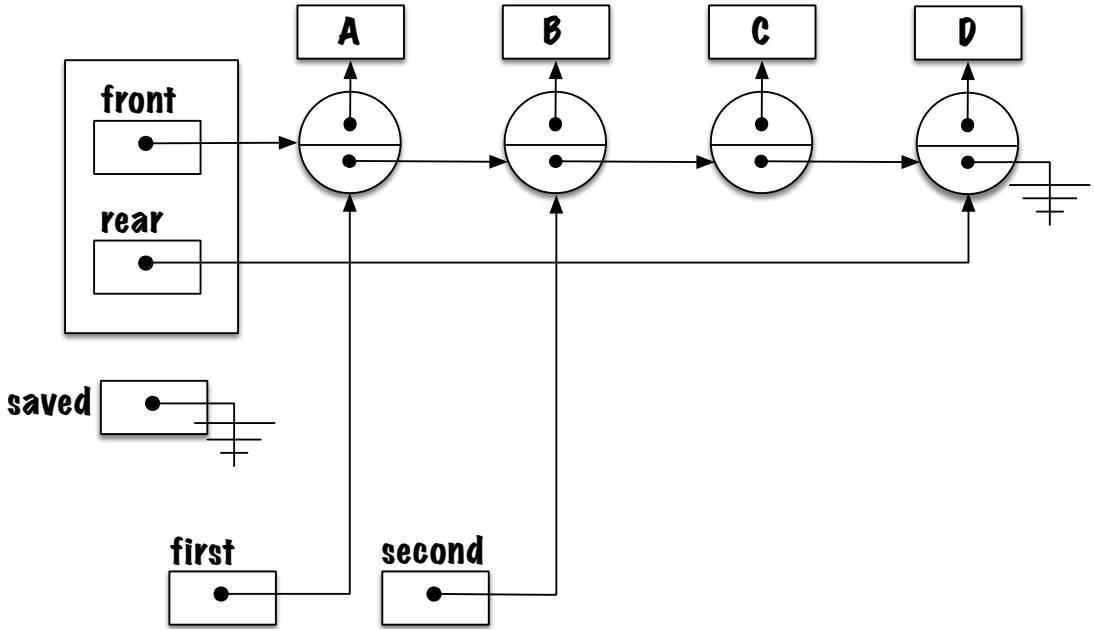
# Methodologie : retrait d'un élément

- Identifiez le **cas général** ainsi que le ou les **cas spéciaux**.
- Est-ce que la file **vide** est un **cas spécial** ?
  - **Non**, c'est un cas **illégal**, pour lequel il faudra **lancer une exception**.
- La file contenant **un seul élément** est le **cas spécial**.

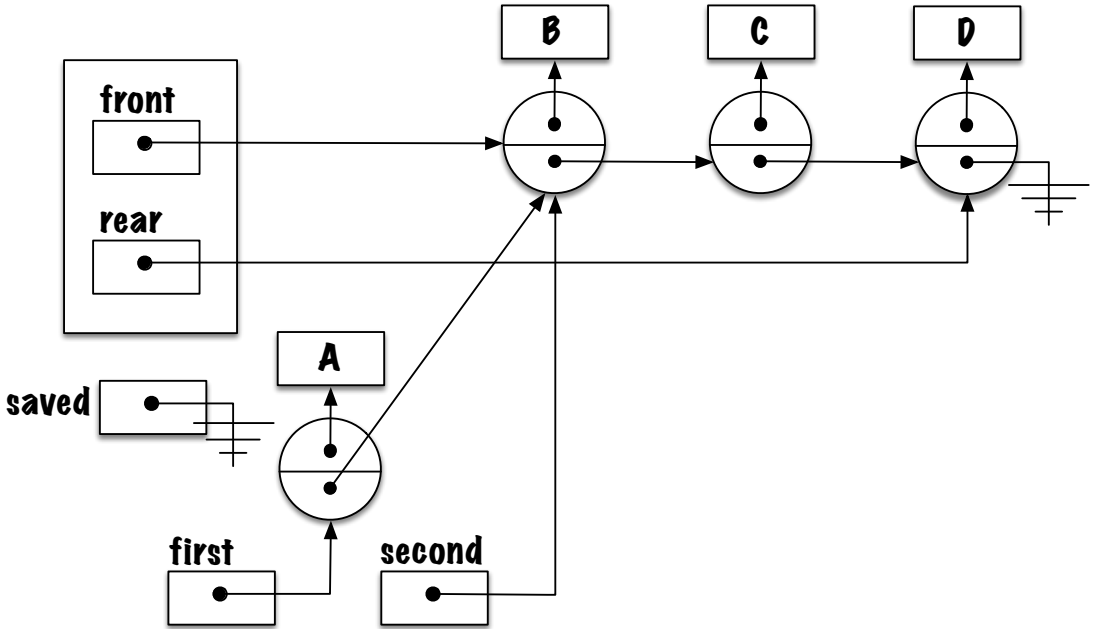
# Retrait d'un élément (cas général)



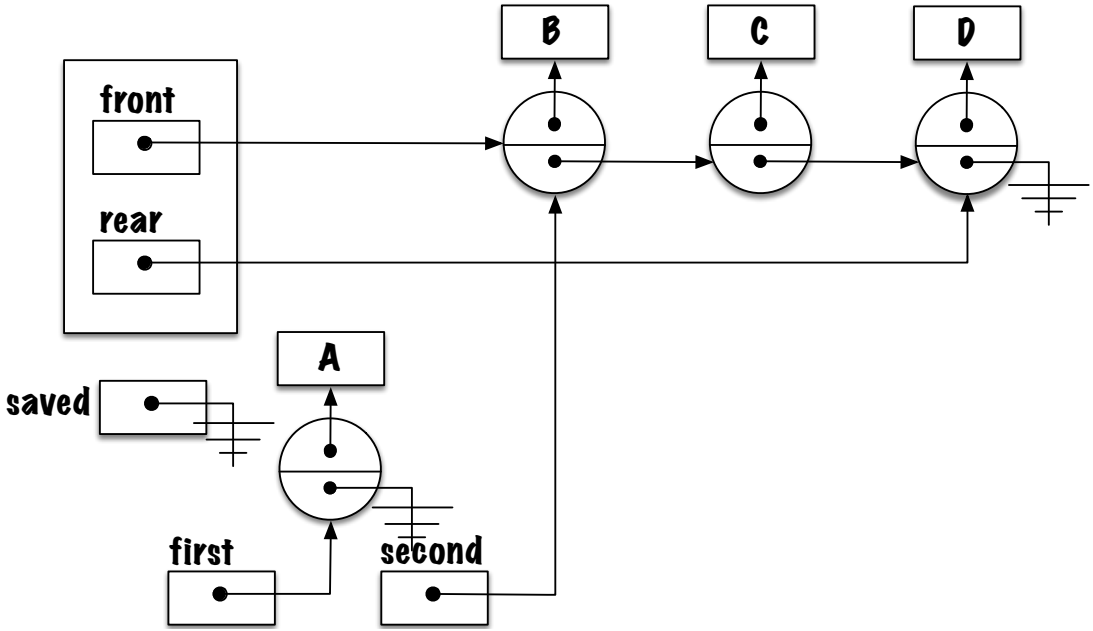
# Retrait d'un élément (cas général)



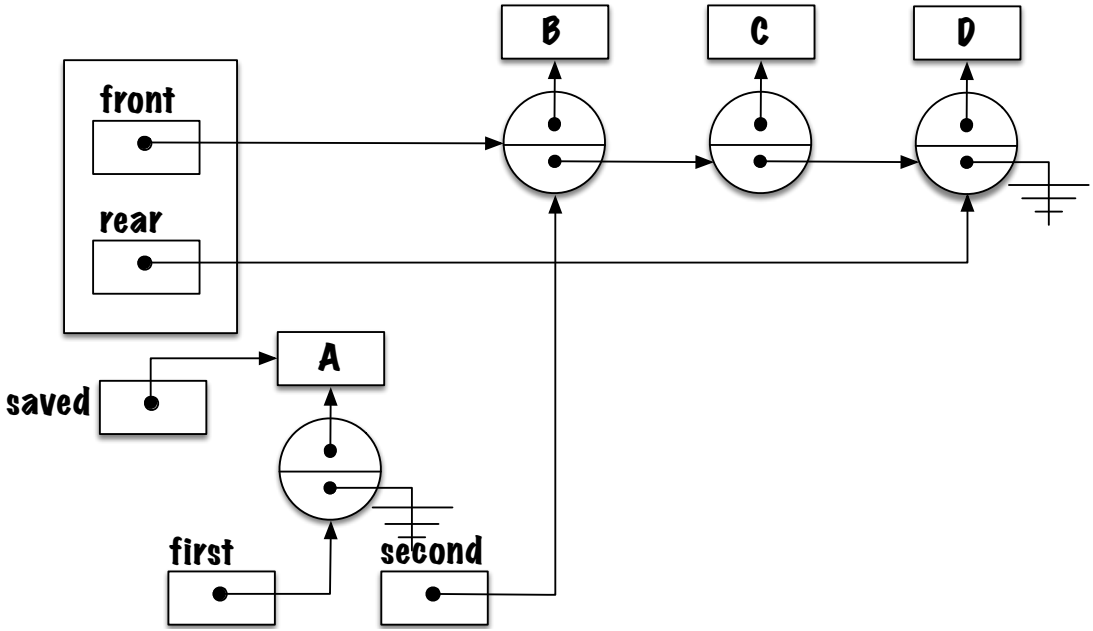
# Retrait d'un élément (cas général)



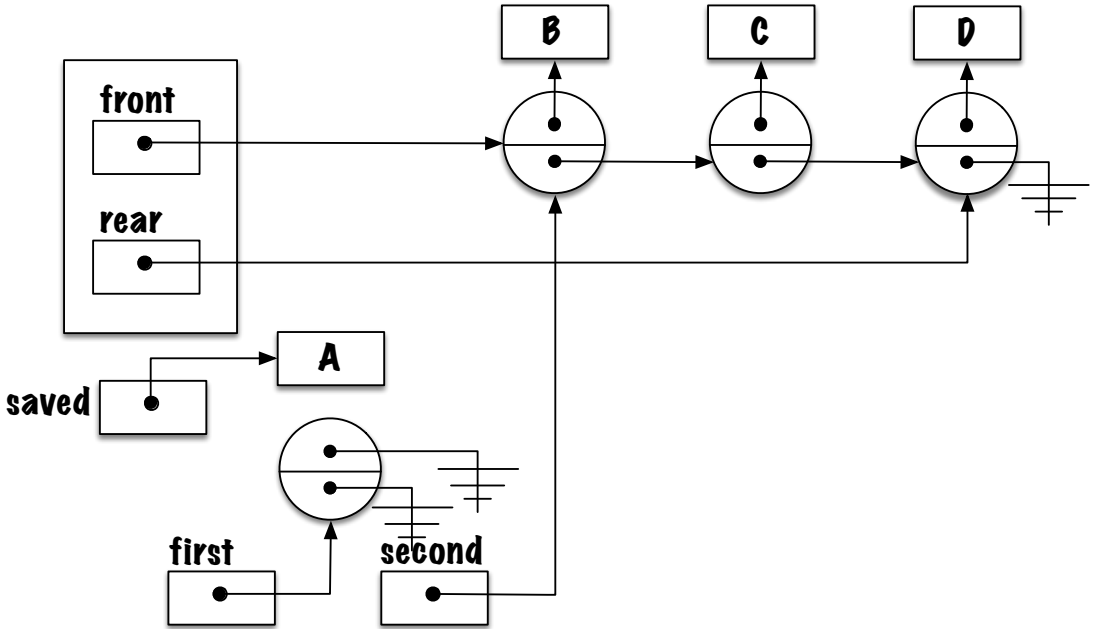
# Retrait d'un élément (cas général)



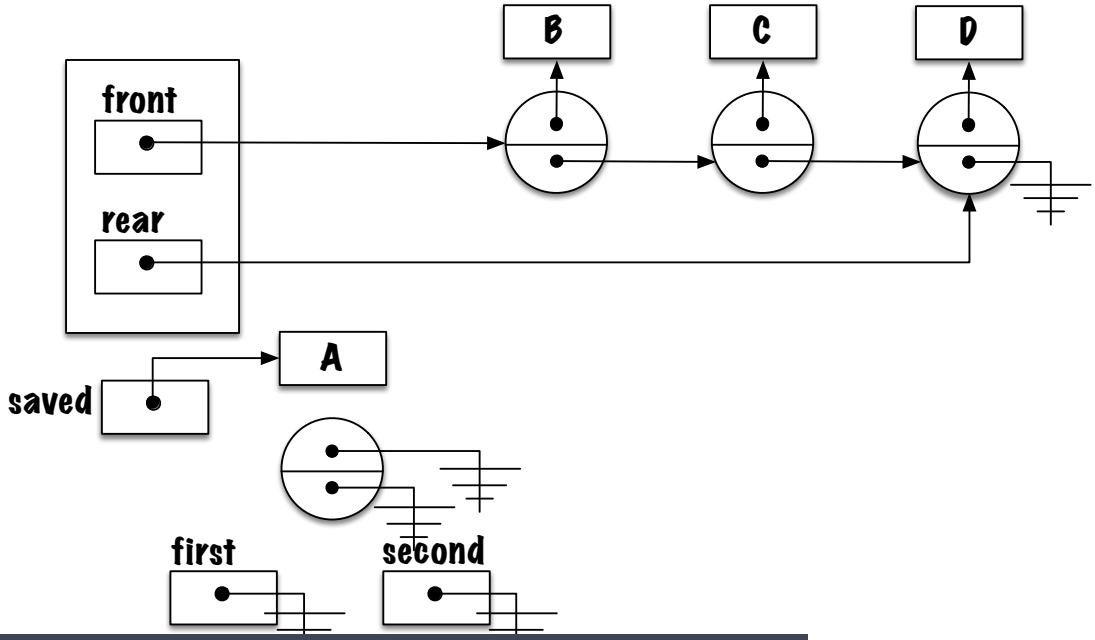
# Retrait d'un élément (cas général)



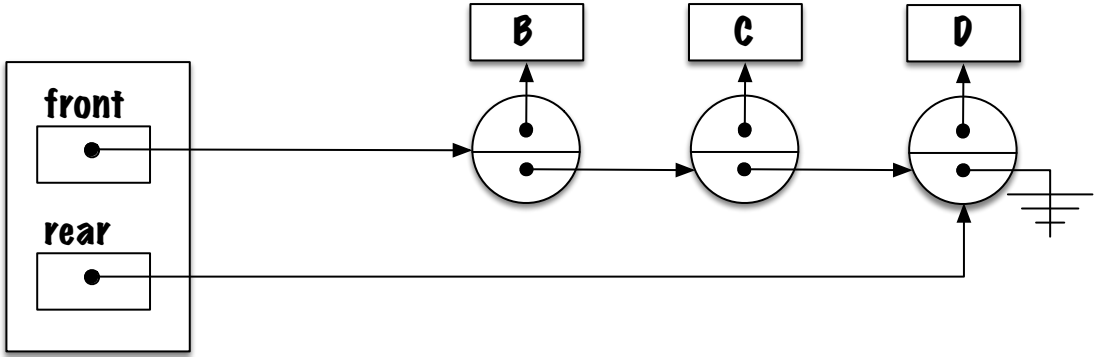
# Retrait d'un élément (cas général)



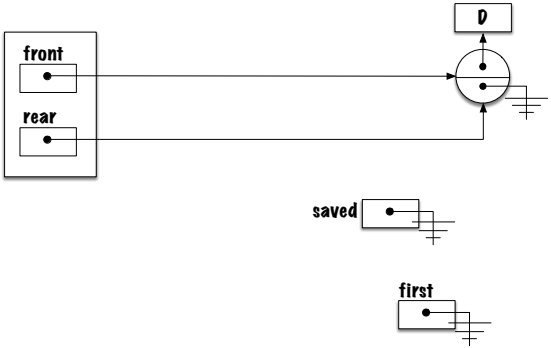
# Retrait d'un élément (cas général)



# Retrait d'un élément (cas général)



# Retrait d'un élément (cas spécial)

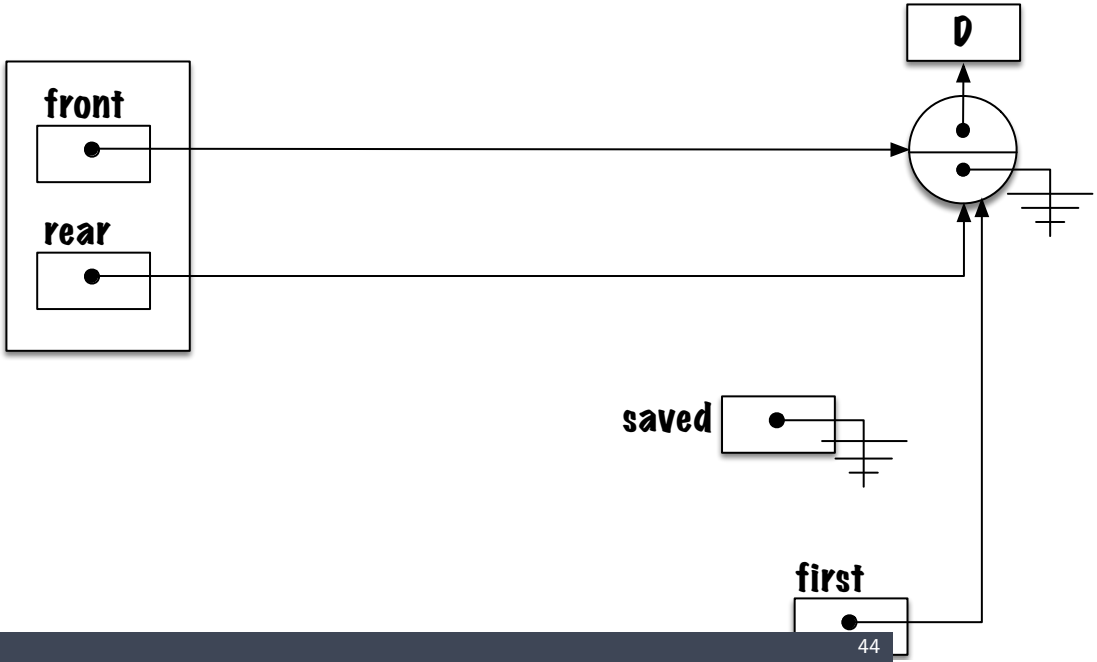



- Quelle **expression** permet de reconnaître une file contenant **un seul élément** ?
  - **Que pensez-vous** de ce qui suit ?

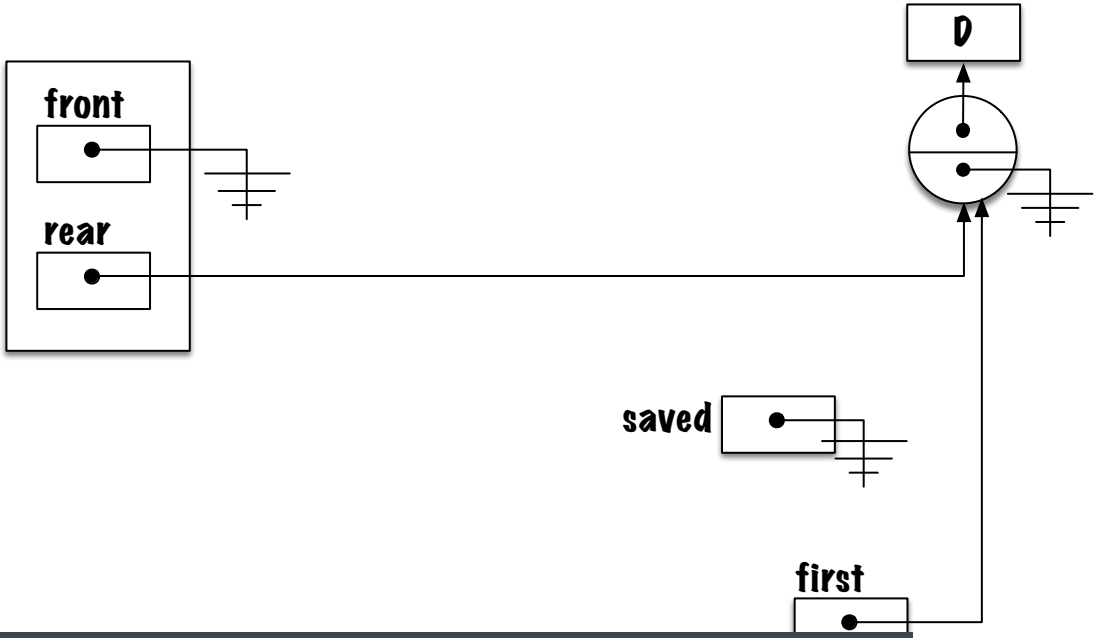
```
front == rear
```

# Retrait d'un élément (cas spécial)

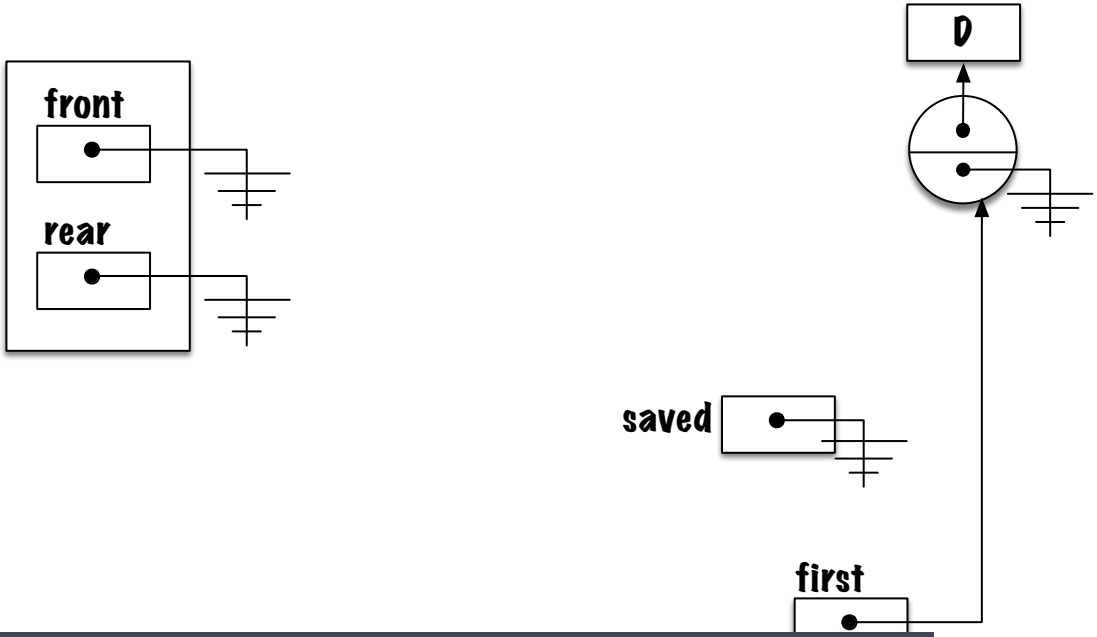
# Retrait d'un élément (cas spécial)



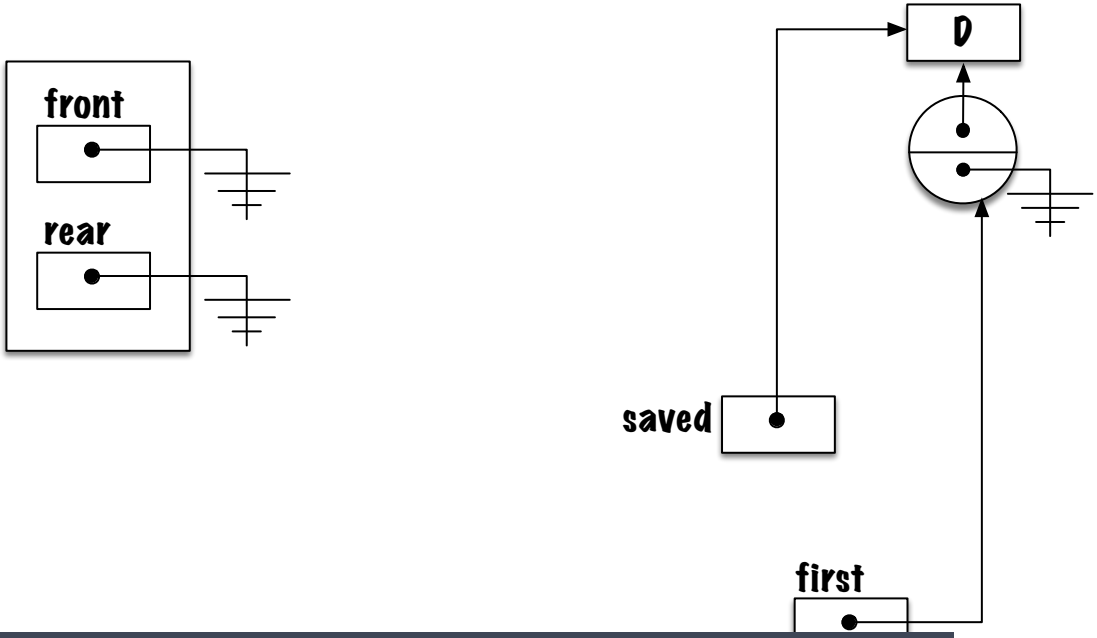
# Retrait d'un élément (cas spécial)



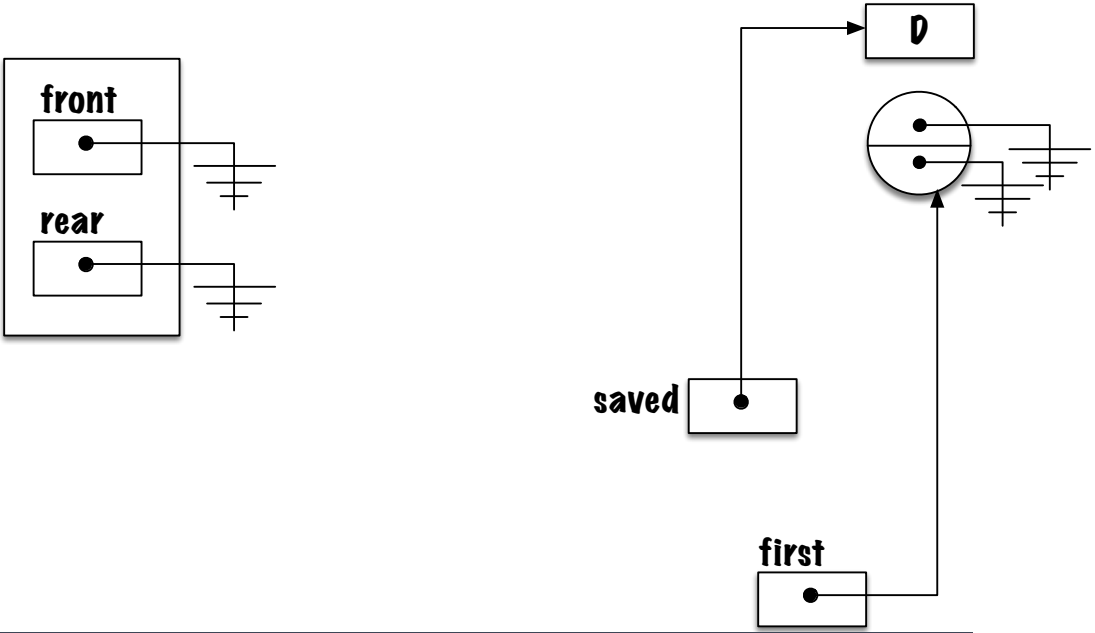
# Retrait d'un élément (cas spécial)



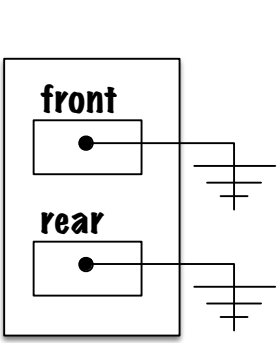
# Retrait d'un élément (cas spécial)

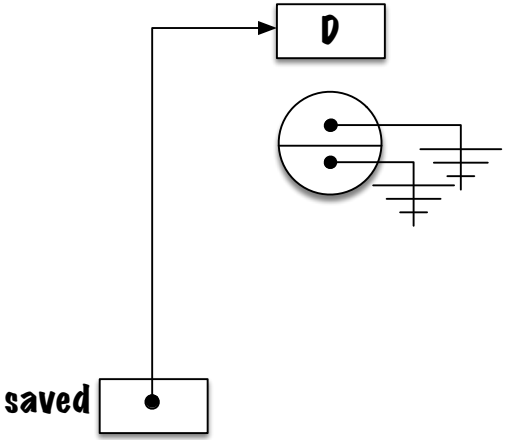


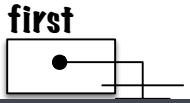
# Retrait d'un élément (cas spécial)



# Retrait d'un élément (cas spécial)



# Piège

# Piège !


- Voici un problème commun, **le lien vers l'élément arrière n'a pas été brisé**.
- Quels **types de problèmes** pourraient survenir ?
- **Qu'arrivera-t-il** si l'on utilise l'expression suivante afin de détecter la file vide ? `front == null && rear == null`.

# Prologue

# Résumé

- Une **file** (*queue*) est un **type abstrait de données** linéaire tel que l'ajout de données se fait à une extrémité, l'**arrière** (*rear*) de la file, et le retrait à l'autre, l'**avant** (*front*).
- L'**implémentation chaînée** nécessite une référence vers l'**élément avant** ainsi que l'**élément arrière**.

# Prochain module

- **Queue** : applications

# References I


E. B. Koffman and Wolfgang P. A. T.
***Data Structures : Abstraction and Design Using Java.***
John Wiley & Sons, 3e edition, 2016.


Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

**Université d'Ottawa**