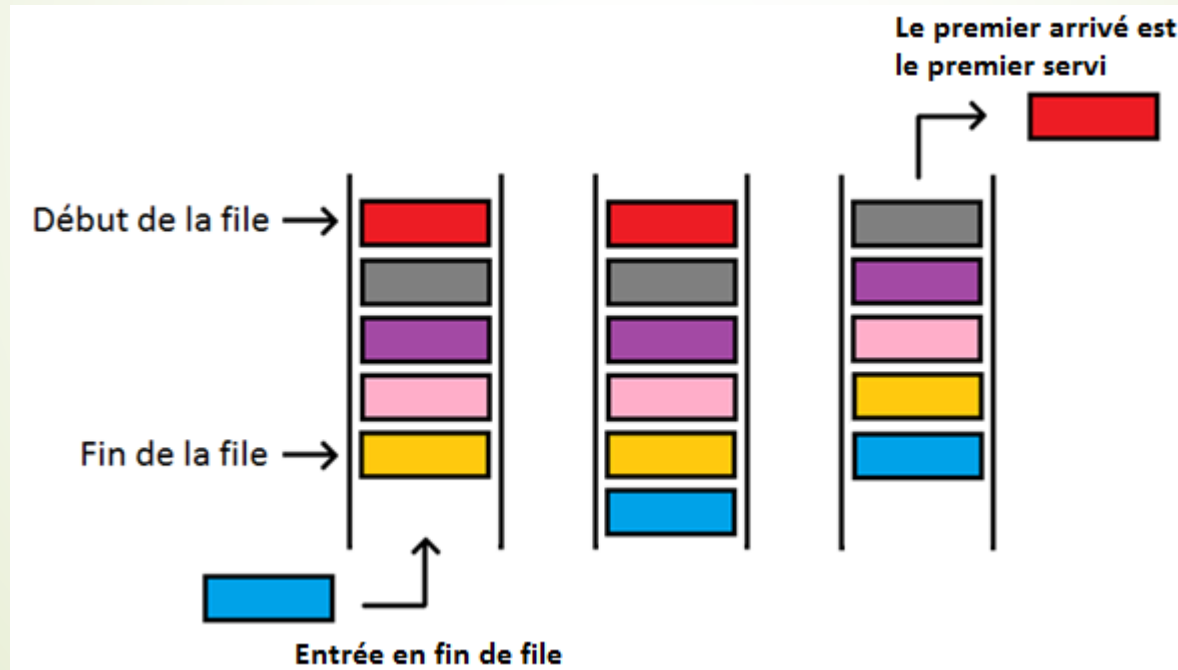


Structures dynamiques – File (Queue)

Daniel Huot

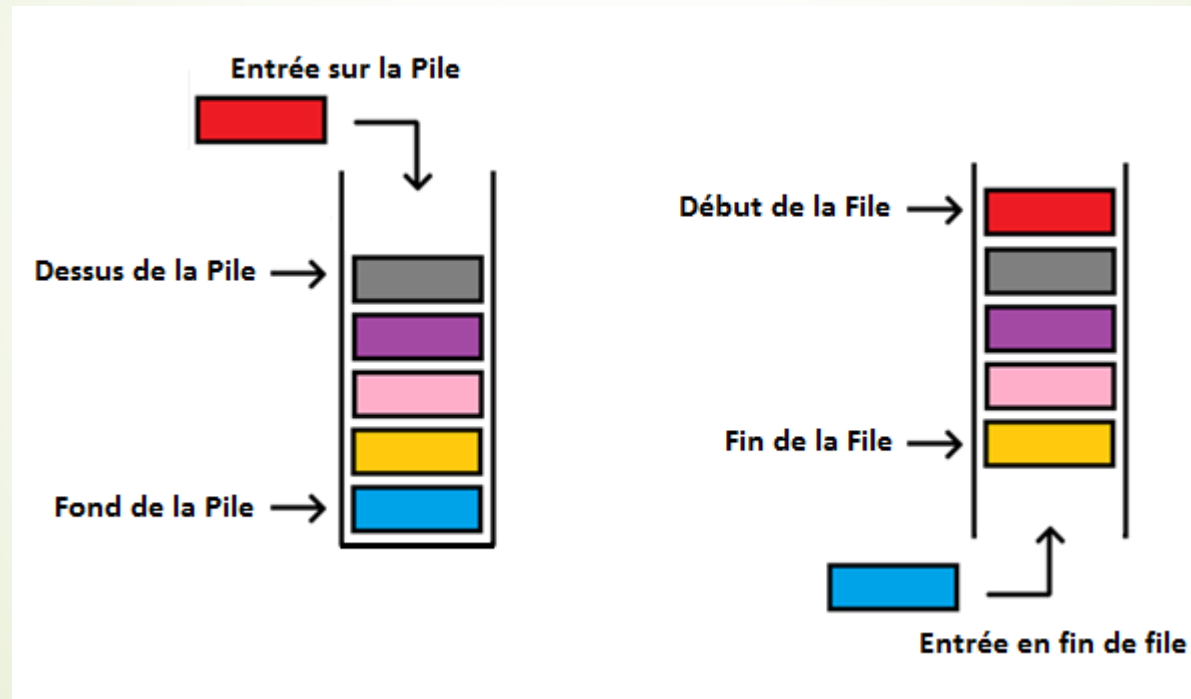
La File (Queue)

La File est une structure de données de type « premier entré, premier sorti » (First In First out)



La File (Queue)

La façon de visualiser la File est contraire à la Pile dans le sens où les contenus arriveront « par le bas ».



Pile vs File



La File (Queue)


La File est utilisée lorsqu'on doit mettre en attente des données avant leur traitement.

L'extraction d'une donnée en plein centre de la File est donc contre-nature.



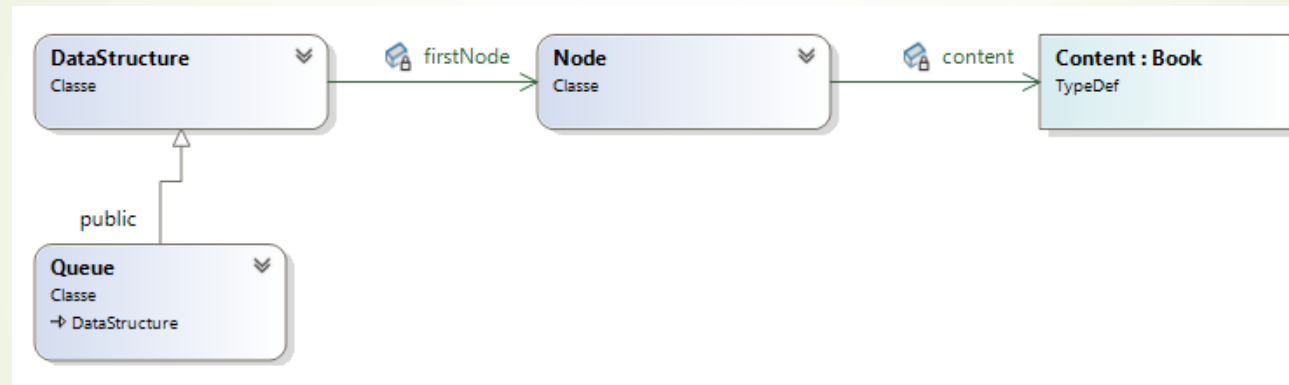
La File (Queue)

Exemples d'utilisations :

- Mise en mémoire tampon (« buffering » d'une vidéo par exemple)
 - « Spooler » d'impression (mettre les documents en attente selon l'ordre d'arrivée)
 - Priorité des tâches dans un système d'exploitation multi-tâches
 - Technique d'interpolation pour gérer le « lag » dans un jeu vidéo
- 

La File (Queue)

Selon la composition qui a été vue, le diagramme de classes pour mettre en œuvre une file est le suivant :



La File (Queue)

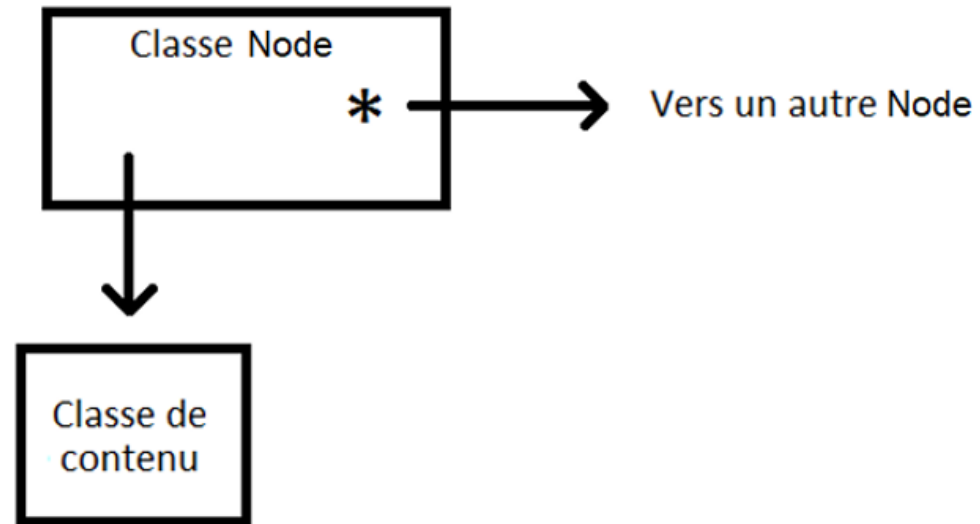
La classe de Node ne diffère pas de celle utilisée pour la Pile, car nous aurons besoin du concept de nœud également :

```
class Node
{
public:
    Node();
    ~Node();
    Node* getNext() const;
    void setNext(Node* _next);
    const Content& getContent() const;
    void setContent(const Content& _content);

private:
    Content content;
    Node* next;
};
```

La File (Queue)

Rappel : Illustration d'un noeud





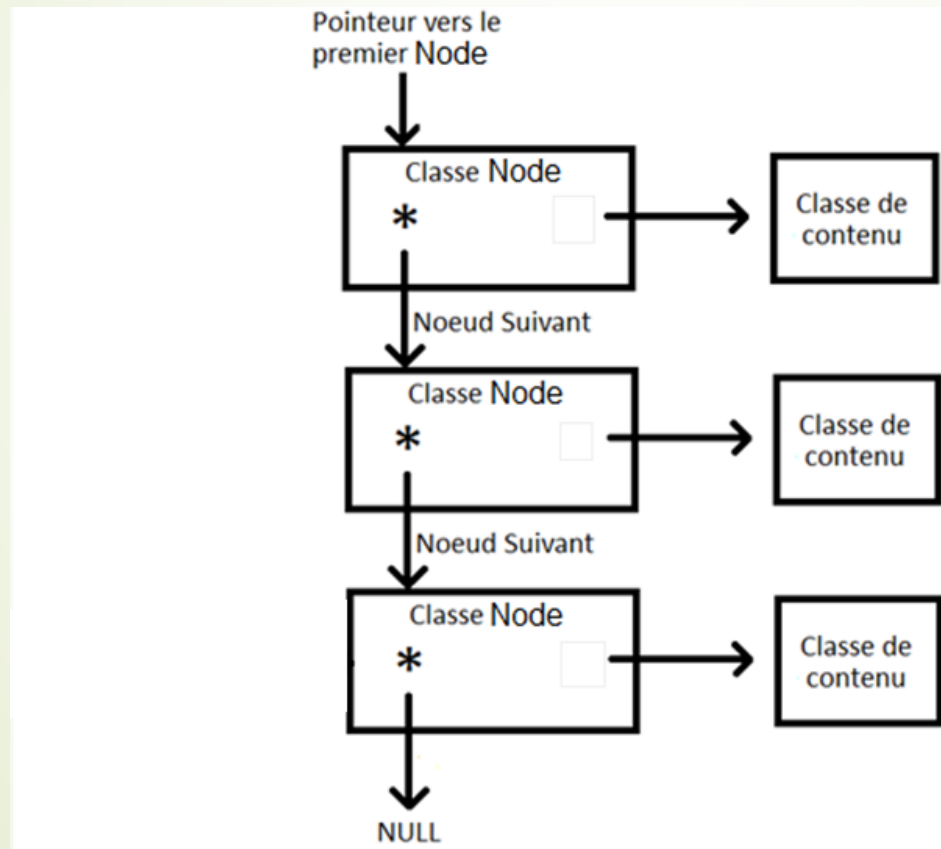
La File (Queue)

Observations :

- Il va de soi que lorsqu'il y a un nouvel arrivant dans une file, le premier nœud n'est pas modifié (imaginez que vous êtes le premier d'une file, et qu'on change votre statut au prochain arrivant !)
- Nous pourrions avoirs besoin d'un pointeur de dernier nœud afin de faire rapidement le chaînage du nouvel arrivant. Cependant, il est tout à fait possible de coder une méthode qui nous le retourne (puisque c'est un champ calculé). **C'est l'approche que nous utiliserons.**
- Par contre, si la file est vide, c'est le seul cas où le premier nœud sera modifié

La file (Queue)

Illustration d'une file avec ses nœuds :





La file (Queue)

Remarques importantes :

1. Le pointeur vers le premier nœud est essentiel, il sert à
 - Accéder directement au premier nœud pour le traitement du premier dans la file
 - Point d'entrée dans la file pour la parcourir au complet en utilisant les nœuds suivants (ex : Afficher tous les contenus de la file)
2. Le nœud suivant du dernier nœud doit pointer vers `nullptr` (indicateur de fin de file)



DataStructure : la classe mère de nos structures

Les structures que nous verrons dans le cours auront toutes :

- un premier nœud
- une méthode utilitaire pour compter le nombre de nœuds
- Un affichage particulier (polymorphisme !)

Nous allons généraliser ces comportements dans une classe mère nommée

La classe DataStructure

```
class DataStructure
{
public:
    DataStructure();
    DataStructure(const DataStructure& src);
    virtual ~DataStructure();
    virtual void display() const = 0;
    unsigned int size() const;
    virtual bool isEmpty() const;

protected:
    const Node* getFirstNode() const;
    Node* getFirstNode();
    void setFirstNode(Node* node);

private:
    Node* firstNode;
};
```



La file

Pour la file, on ne parlera pas de « `push` » ni de « `pop` » mais de « `push_back` », « `pop_front` » et « `front` » pour les contenus

La classe Queue

```
class Queue : public DataStructure
{
public:
    Queue();
    Queue(const Queue& src);
    ~Queue();
    bool operator==(const Queue& queue) const;
    void push_back(const Content& _content);
    void pop_front();
    const Content& front() const;
    void display() const override;
    const Node* getLastNode() const;
    Node* getLastNode();
    void clear();
private:
};
```

} Pas obligatoires mais peuvent être utiles pour les autres méthodes



La File

Attention :

Le codage de la file est plus technique. Il demande de faire quelques vérifications tant au niveau de l'ajout que du retrait pour garder l'état de la file dans le bon état

Conseil : Dessinez !





La File

Spécifications :

- La file est vide lorsque sa taille est zéro.
- La méthode `push_back` va créer un nœud, y insérer le contenu reçu et effectuer les chaînages pour enfiler le nœud adéquatement en fin de file
- La méthode `front` va retourner le contenu stocké dans le premier nœud sans modifier la file
- La méthode `pop_front` va détruire le premier nœud (composition !) refaire les chaînages adéquatement
- Effectuer un `pop_front` ou un `front` sur une file déjà vide lancera un `std::runtime_exception`
- Si la file est détruite et qu'il reste des nœuds, la file devra respecter sa composition forte. Indice : faire des appels répétés de `pop_front`.