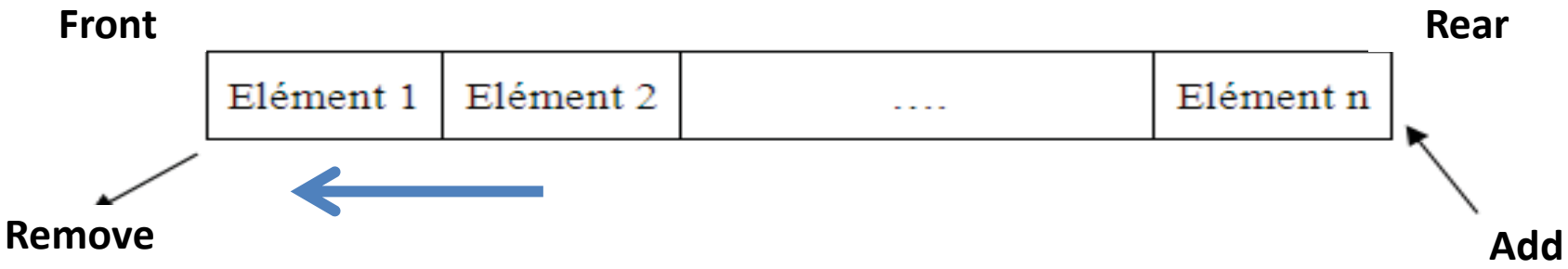
Chapter 4 : Queue

Introduction to C++ Programming

1. Definition
2. Static Queues
3. Dynamic Queues

1. Definition

- **The queue is a data structure that allows elements to be stored in a given order and removed in the same order, i.e. according to the FIFO "First In First Out" protocol.**
- **"We always add an element at the end and remove the one at the front".**



1. Definition

Depending on the type of implementation, two types of queues are distinguished:

- **Static Queue:** Implementation by an array.

- **Dynamic queue:** Implementation by a linkedd list.

1. Primitive operation

In the queue data structure, the primary or "primitive" operations are:

- **Enqueue**: Adds an element to the end of the queue.
- **Dequeue**: Removes an element from the front of the queue.
- **Front** (or **Peek**): Retrieves the front element without removing it.
- **IsEmpty**: Checks if the queue is empty.
- **Size:** returns the numbers of elements in the queue

2. Static Queues

- Implementation by an array.
- The definition of the **Queue** (static) type is as follows:

Type Structure Queue

Begin

Tab: Array[MAX] of Elements;

Head: integer; // index of the first in the Queue.

Tail: integer; // index of the last element inserted into the Queue.

END

F

Tab:

10	20	14	7				

Head : 1

Tail: 4

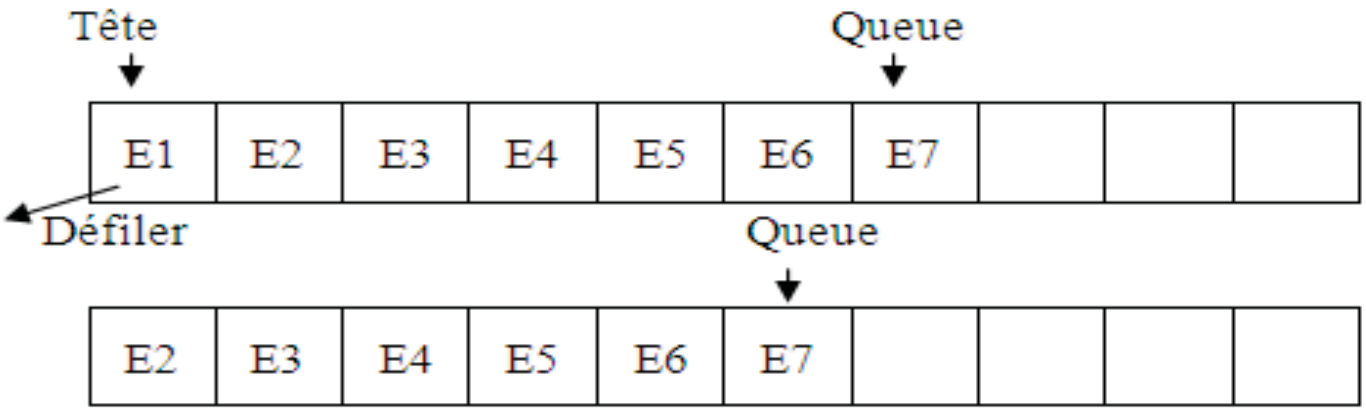
2. Static Queues

- Implementing static Queues can be done in two ways:
  - **By shifting:** using an array with a fixed head, always at 1, and a variable tail.
  - **By flow:** It can also be done by flow using a circular array where the head and tail are both variables.

2. Static Queues

Implementation by shifting

- **Fixed Head** and **Variable Tail**.

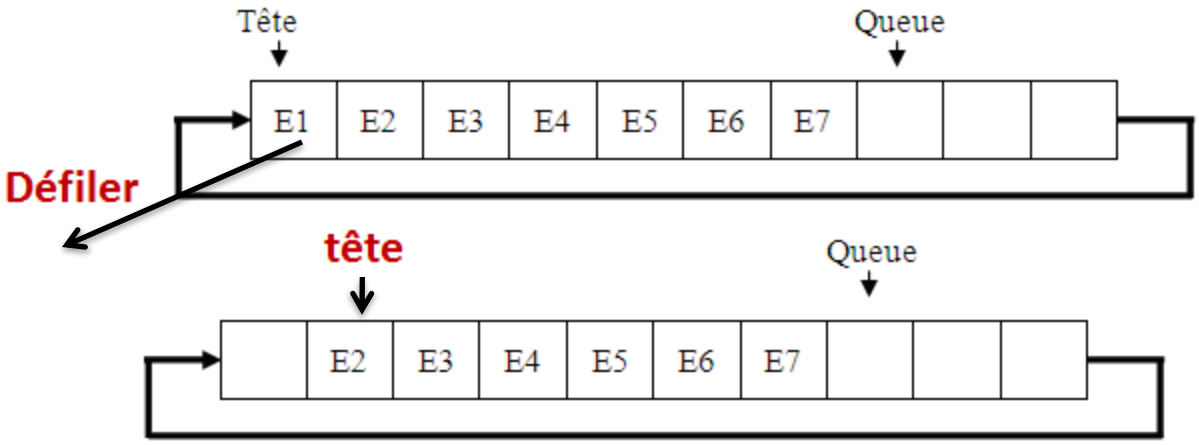


- The queue is empty if Tail= 0
- The queue is full if Tail= Max
- **It suffers from the shifting problem with each dequeue operation.**

2. Static Queues

Implementation by a circular Array

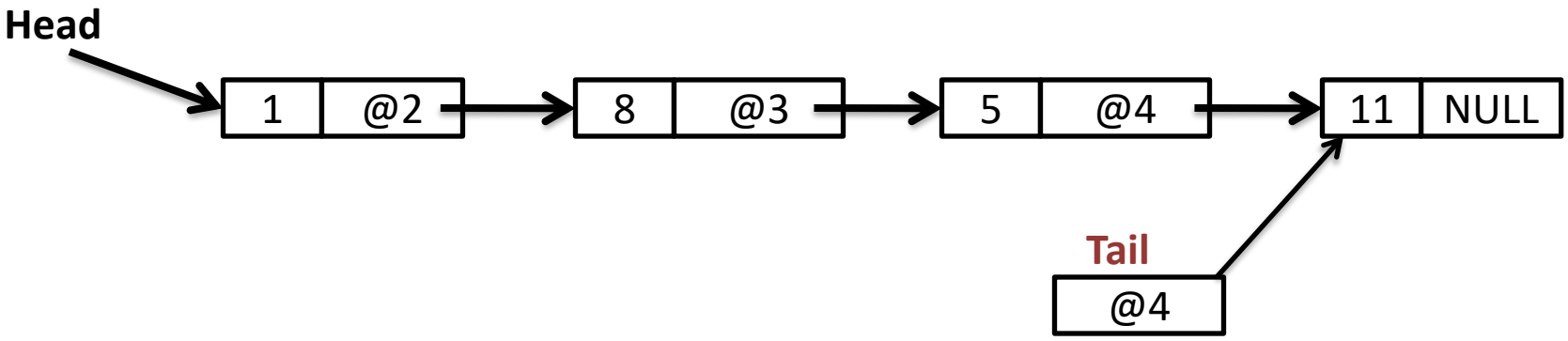
- **Variable Head** and **Variable Tail**



- The queue is empty if Head = Tail
- The queue is full if (Queue + 1) mod Max = Head

3. Dynamic queues

It is a linked list where dequeuing occurs only at the head, and enqueuing occurs only at the tail of the list.



3. Dynamic queues

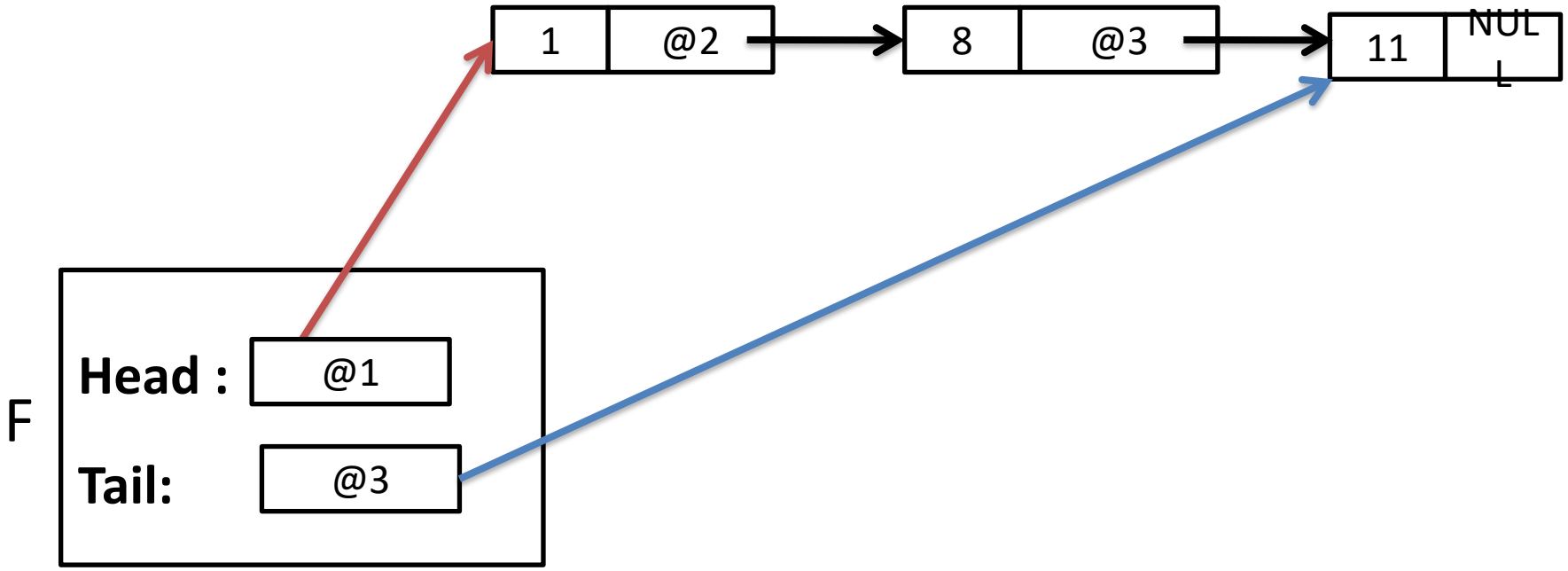
Dynamic Queue Type Definition

The definition of the **Queue** (dynamic) type is as follows:

```
Type Structure Node
    Ele : typeq ;
    next: * Node ;
END
Type Structure Queue
Begin
    Head: *Node; // Keeps the address of the head of the queue.
    Tail: *Node; // Holds the address of the last element in the
                                    queue.
END
```

3. Dynamic queues

The structure of a dynamic queue



3. Dynamic Queues (Primitive Operations)

1. Initialize : Initializes the Head and Tail variables of the queue to NULL.

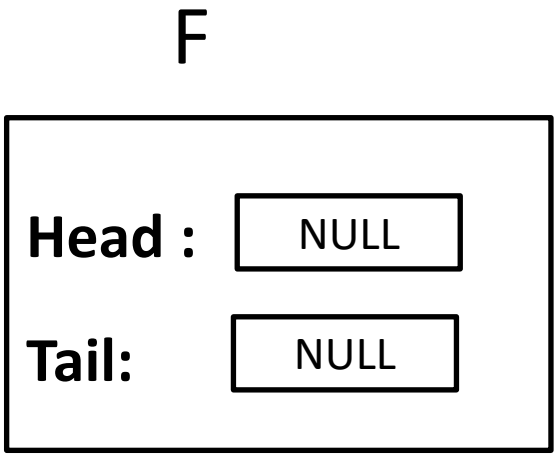
Procedure Initialise(var F: Queue)

Begin

F.Head ← NULL;

F.Tail ← NULL;

END



3. Dynamic Queues (Primitive Operations)

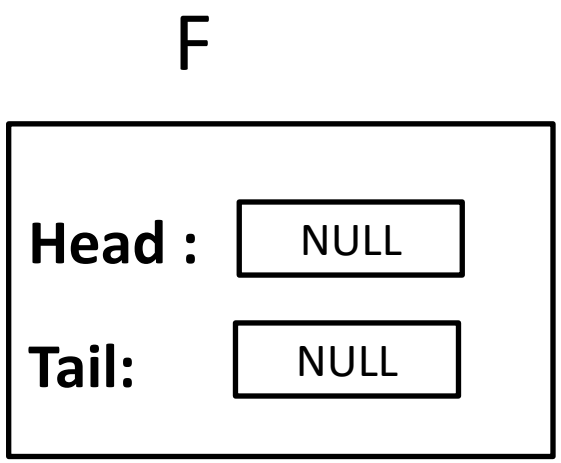
2. Is_empty : returns true if the queue is empty otherwise returns false.

Function is_empty (F: Queue): boolean

Begin

Returns (F. Head = NULL);

END



3. Dynamic queues

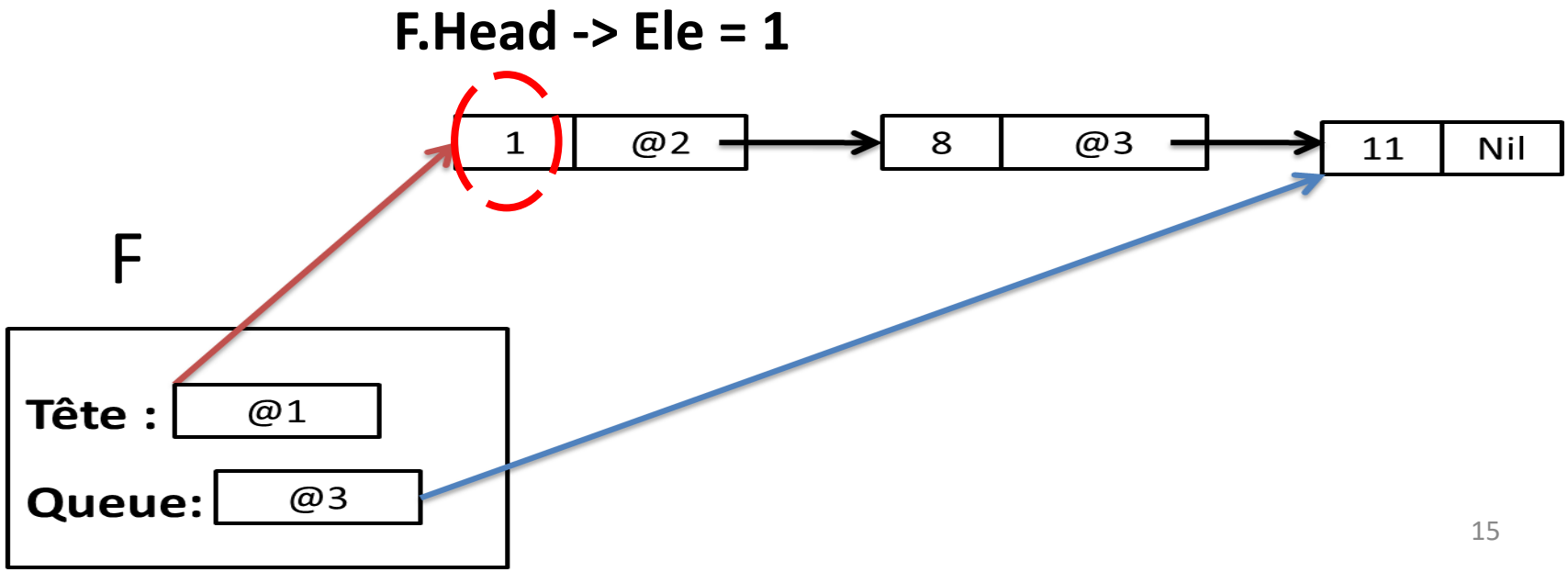
3. Head : Returns the element that exists in the header of the Queue (The first element threaded).

function Head (F: Queue): integer

Begin

Returns (**F.Head -> Ele**);

END



3. Dynamic Stacks

Enqueue : Adds an element to the tail of the queue.

Procedure **Enqueue** (var F: Queue, x: typeq)

P: *Node

Begin

P ← create_Node (x);

If (is_empty(F)) then

F.Head ← P

F.Tail ← P;

Else

F. Head-> next ← P;

F.Tail← P;

End if

END

3. Dynamic Stacks

Dequeue: Removes the first element (the Front) from the queue.

```
Procedure Dequeue (var F: Queue)
    P: *Node
Begin
    If( ! is_empty (f))
            P ← F.head ;
            F.Head ← f.Head -> next;
            Desallocate (p);
            If ( F.Head =NULL)
                    F.Tail ← NULL;
            End if
  Endif
END
```

3. Dynamic Stacks

- **The size function** : returns the number of elements already enqueue into the queue.

```
Function Size (F: Queue): integer
    Current: *Node
    Nb: integer;
Begin
    Current ← F.Head; Nb ←0 ;
    While (current != NULL)
        Nb ← Nb +1;
        Current ← Current->next ;
    Endwhile
        Return Nb;
END
```

4. Queue usage examples:

Queues are widely used in various applications where data needs to be processed in a **First-In-First-Out (FIFO)** order such as:

- **Task Scheduling in Operating Systems**:
  - Queues manage tasks, processes, or threads that need CPU time, ensuring that the first task to enter is the first one to be executed.
  - Print queues also work similarly, where documents are printed in the order they are submitted.
- **Breadth-First Search (BFS) in Graph Algorithms**: Queues are essential in BFS to keep track of nodes to visit next, allowing each level of nodes to be processed sequentially.

4. Queue usage examples:

- **Handling Requests in Web Servers**: When multiple users make requests to a server, requests are stored in a queue, ensuring they are processed in the order received.

- **Data Buffers**: Queues are used in buffering for streaming or real-time applications, like video streaming, where data needs to be processed in the order it arrives.

- **Messaging Systems**: Messaging queues (like those used in RabbitMQ) store messages for distributed systems, ensuring messages are processed in the order they were received.

- **Customer Service Systems**: Call centers and help desks use queues to manage customers waiting for assistance, where customers are served in the order they called.

4. Queue usage examples: