# Module 4:
Queue ADT

Dr. Natarajan Meghanathan
Professor of Computer Science
Jackson State University
Jackson, MS 39217
E-mail: natarajan.meghanathan@jsums.edu

Queue ADT

End of the
Queue

- Features (Logical View)
  - A List that operates in a First In First Out (FIFO) fashion
  - Insertion can be done at the end of the list and deletion is done from the front of the list
    - The first added item has to be removed first
  - Operations:
    - Enqueue( ) – adding an item to the end of the Queue
    - Dequeue( ) – delete the item from the front of the Queue
    - Peek( ) – read the item at the front of the Queue
    - IsEmpty( ) – whether there is any element in the Queue
  - All the above operations should be preferably implemented in O(1) time.

15
24
13
42
71
30

Front of
the Queue

End

35		35
15		15
24		24
13		13
42		42
71		71
30		30
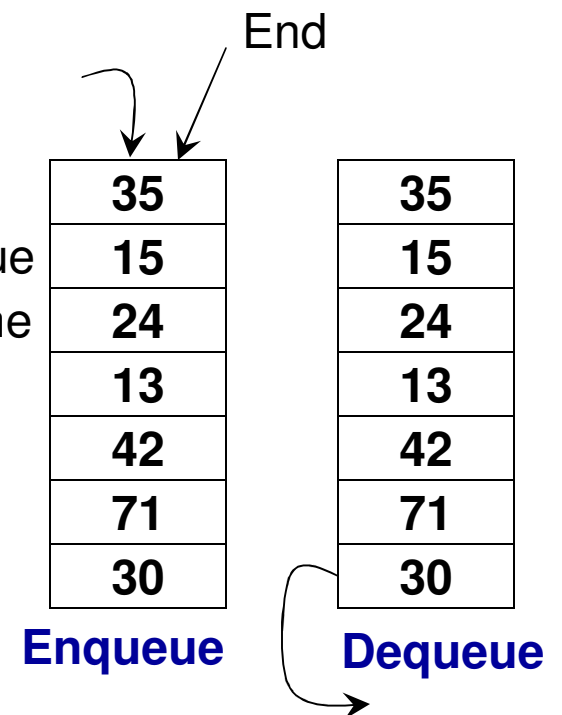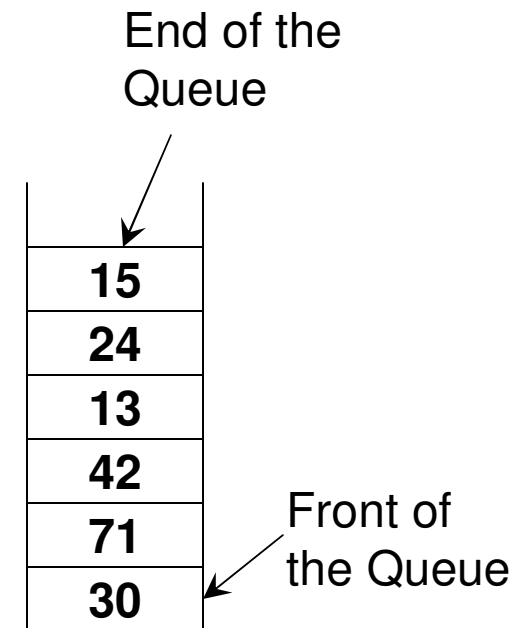
Enqueue   **Dequeue**

Implementation of Queue
Dynamic Array vs. Singly/Doubly Linked List

- Enqueue
  - Array: O(n) time, due to need for resizing when the queue gets full
  - Singly Linked List: Θ(n) time: traversal of the entire list is needed
  - Doubly Linked List: O(1) time
- Dequeue
  - Array: Θ(n) time, as elements need to be shifted one position to the left
  - Singly Linked List: O(1) time, as the headPtr just needs to be adjusted
  - Doubly Linked List: O(1) time
- Peek
  - Array: O(1) time
  - Singly Linked List: O(1) time, as the first node info needs to be just seen
  - Doubly Linked List: O(1) time

- With a doubly-linked list based implementation of the queue, we can enqueue by inserting the new node from the tail of the list and dequeue (or peek) by removing (or reading the value of) the node next to the head node.
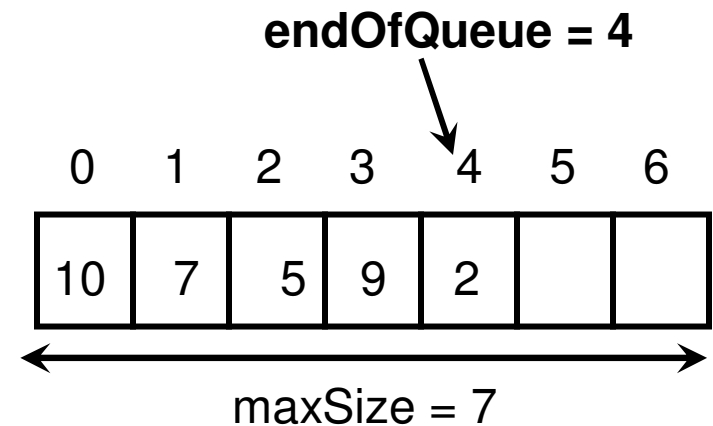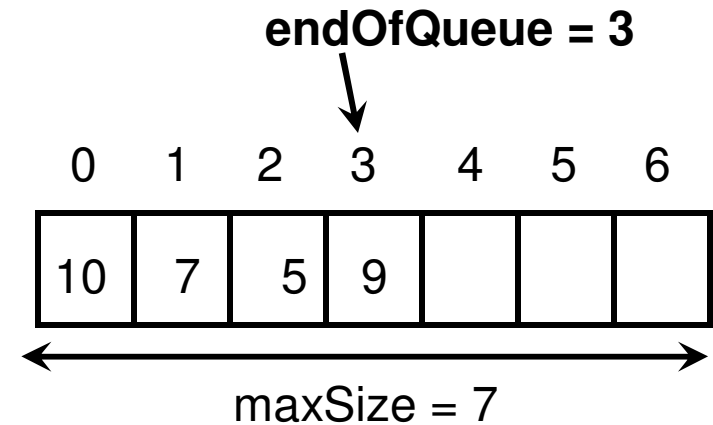
Code 4.1: Dynamic Array-based Queue

```cpp
private:                              C++
        int *array;
        int maxSize;
        int endOfQueue;

public:
        Queue(int size){
                array = new int[size];
                maxSize = size;
                endOfQueue = -1;
        }       // Same as endOfArray

        bool isEmpty(){

                if (endOfQueue == -1)
                        return true;

                return false;

        }
```

endOfQueue = 3

0	1	2	3	4	5	6
10	7	5	9			

maxSize = 7

endOfQueue = 4

0	1	2	3	4	5	6
10	7	5	9	2		

maxSize = 7

```cpp
void resize(int s){

        int *tempArray = array;

        array = new int[s];

        for (int index = 0; index < min(s, endOfQueue+1); index++){
                array[index] = tempArray[index];
        }

        maxSize = s;
}
```

Code 4.1 (C++)

```cpp
void enqueue(int data){        // same as insert 'at the end'

        if (endOfQueue == maxSize-1)
                resize(2*maxSize);

        array[++endOfQueue] = data;


}
```

```cpp
int dequeue(){

    if (endOfQueue >= 0){
            int returnVal = array[0];

            for (int index = 0; index < endOfQueue; index++)
                    array[index] = array[index+1];

            endOfQueue--;
            // the endOfQueue is decreased by one

            return returnVal;
    }
    else

            return -1000000; // an invalid value indicating
                             // queue is empty

}
```

/* Store the front value in a temporary variable
        Copy the elements from index+1 to index
starting from index = 0 to index = endOfQueue-1 */

**Code 4.1
(C++)**

```cpp
int peek(){

        if (endOfQueue >= 0)
                return array[0];
        else
                return -1000000;// an invalid value indicating
                                // queue is empty

}
```

Code 3.2: Doubly Linked List-based Implementation of Queue

**Class Queue (C++)**

```
private:
     Node* headPtr;
     Node* tailPtr;

public:

     Queue(){
          headPtr = new Node();
          tailPtr = new Node();
          headPtr->setNextNodePtr(0);
          tailPtr->setPrevNodePtr(0);
     }

     Node* getHeadPtr(){
          return headPtr;
     }

     Node* getTailPtr(){
          return tailPtr;
     }

     bool isEmpty(){

          if (headPtr->getNextNodePtr() == 0)
               return true;

          return false;

     }
```
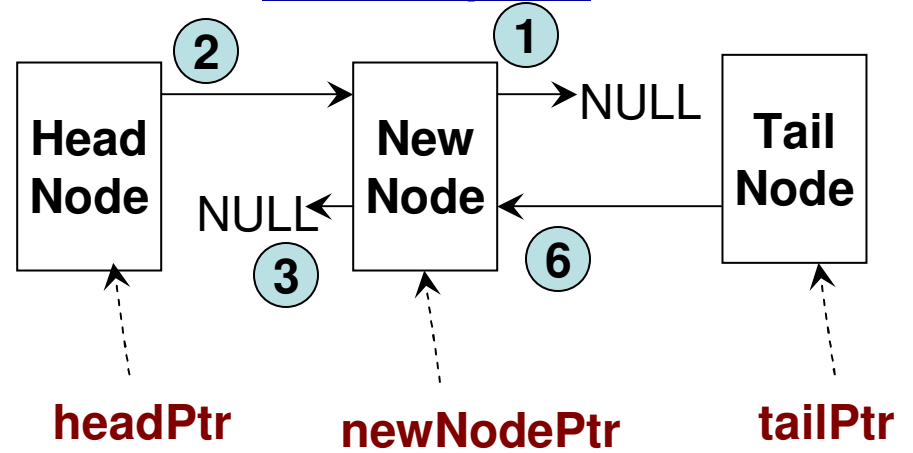
private: Class Node (C++) Overview

```
private:
     int data;
     Node* nextNodePtr;
     Node* prevNodePtr;
public:
     Node( )
     void setData(int)
     int getData()
     void setNextNodePtr(Node* )
     Node* getNextNodePtr( )
     void setPrevNodePtr(Node* )
     Node* getPrevNodePtr( )
```

Enqueue Operation

Scenario 1: There is no node currently in the Queue
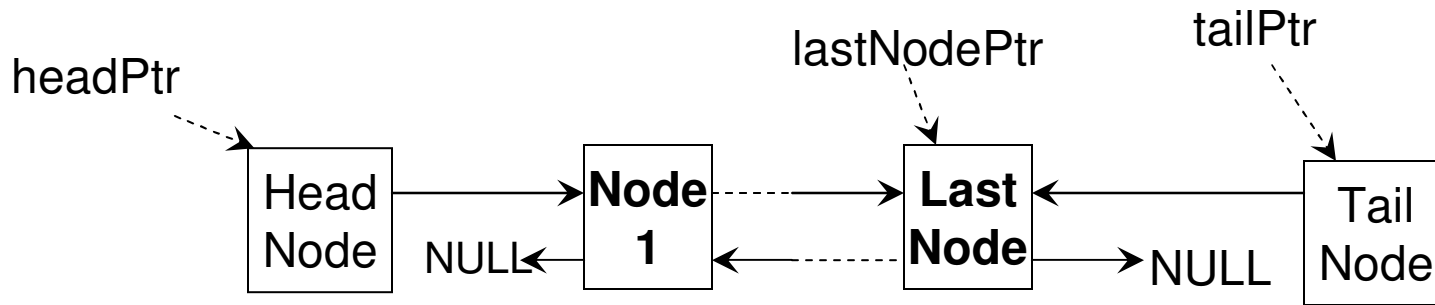
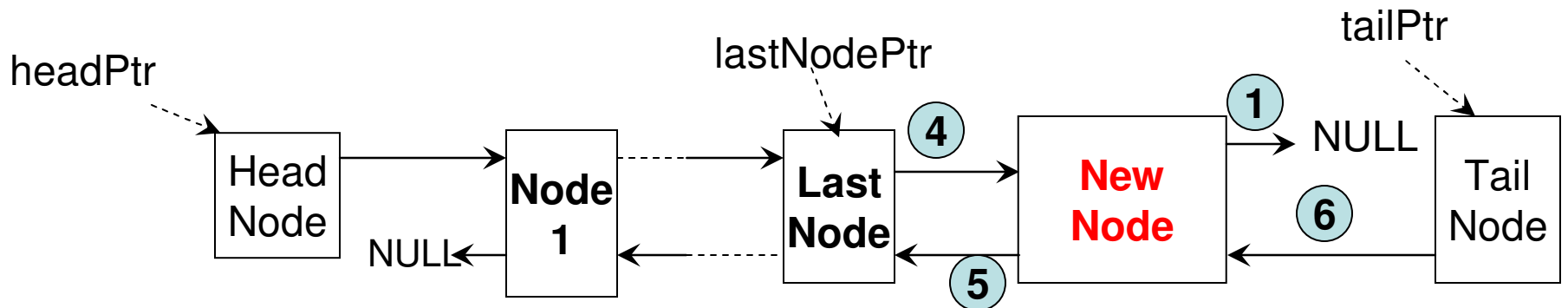**Before Enqueue**
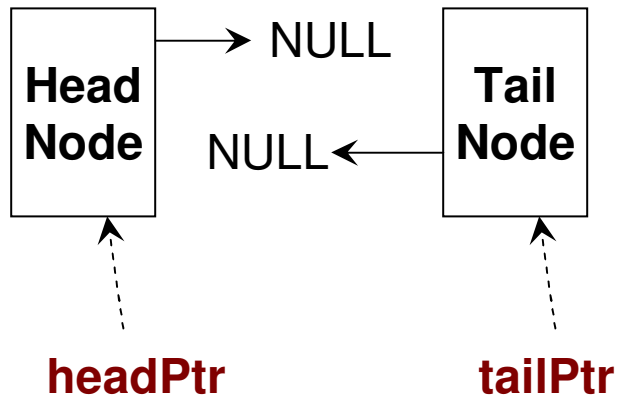
After Enqueue

Enqueue Operation

Scenario 2: There is at least one node already in the Queue

// Before the new node is inserted, the prevNodePtr for the "tail node"
// would be pointing to the last node in the queue and the nextNodePtr
// for that last node would be pointing to NULL.

Before Enqueue

After Enqueue

Code 4.2 (C++)

```cpp
void enqueue(int data){

        Node* newNodePtr = new Node();
        newNodePtr->setData(data);
        newNodePtr->setNextNodePtr(0);  (1)

        Node* lastNodePtr = tailPtr->getPrevNodePtr();

                // There is no other node in the Queue (Scenario 1)
        if (lastNodePtr == 0){

                headPtr->setNextNodePtr(newNodePtr); (2)
                newNodePtr->setPrevNodePtr(0);  (3)
        }
        else{ // There is at least one node already in the Queue (Scenario 2)

                lastNodePtr->setNextNodePtr(newNodePtr); (4)
                newNodePtr->setPrevNodePtr(lastNodePtr); (5)
        }

        tailPtr->setPrevNodePtr(newNodePtr); (6)
}
        Whatever be the case, the prevNodePtr for the tail node
        will point to the newly pushed node
```
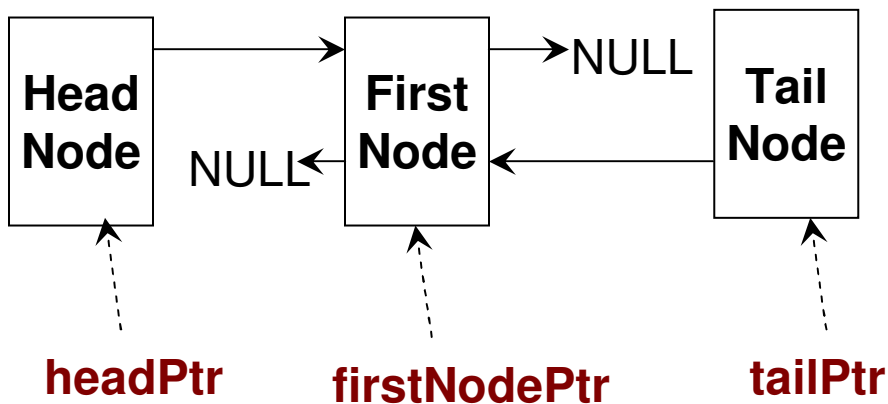
Dequeue Operation

Scenario 1: There will be no node in the Queue after Dequeue (i.e., there is just one node in the Queue before the Dequeue)
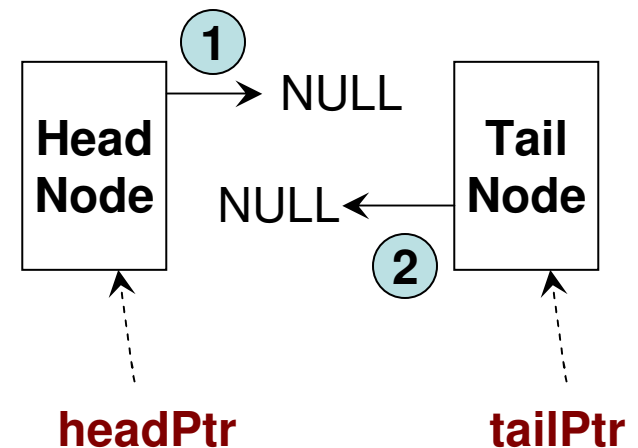
// Before Dequeue: The Head Node's nextNodePtr and the Tail Node's prevNodePtr are both pointing to the only node in the queue.
// After Dequeue: Both the Head Node's nextNodePtr and the Tail Node's prevNodePtr are set to NULL
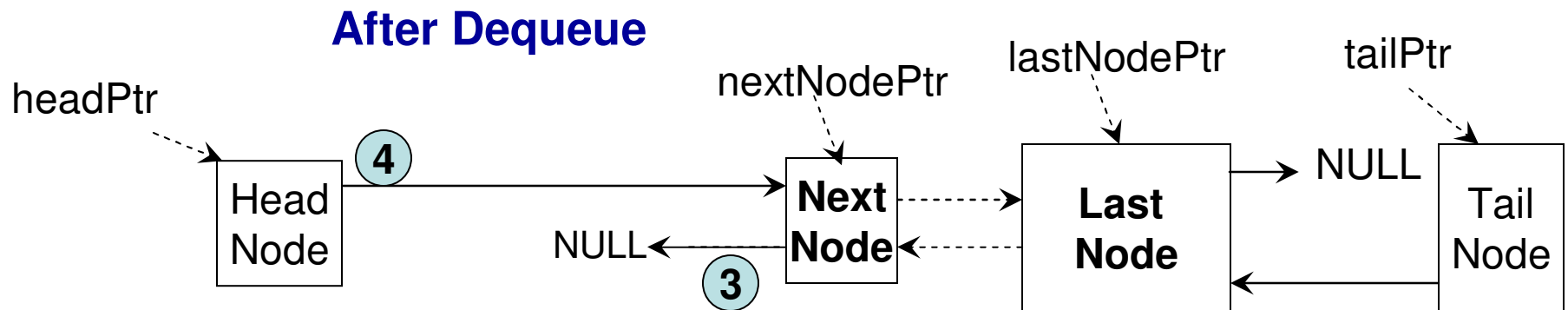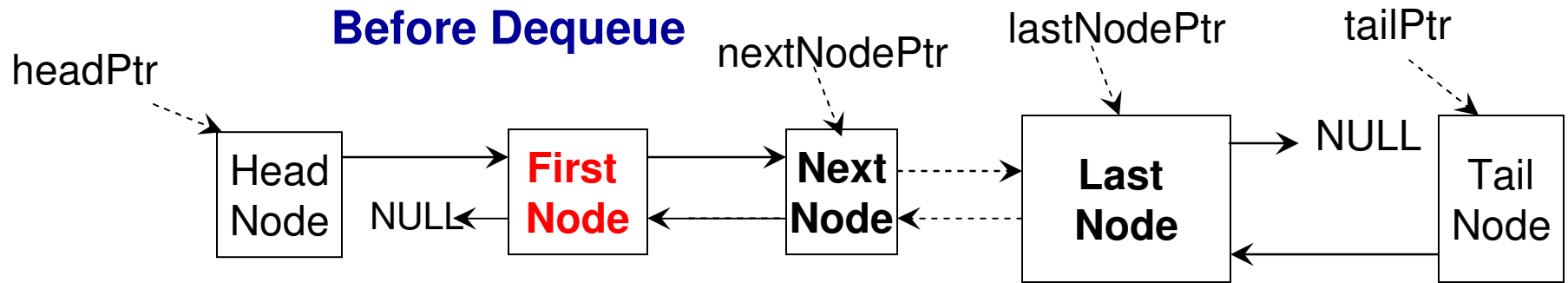
Dequeue Operation

Scenario 2: There will be at least one node in the Queue after the Dequeue operation is executed

Code 4.2 (C++)

```cpp
int dequeue(){

    Node* firstNodePtr = headPtr->getNextNodePtr();
    Node* nextNodePtr = 0;

    int poppedData = -100000; //empty queue

    if (firstNodePtr != 0){
        nextNodePtr = firstNodePtr->getNextNodePtr();
        poppedData = firstNodePtr->getData();
    }
    else
        return poppedData;

    if (nextNodePtr != 0){
        nextNodePtr->setPrevNodePtr(0);
        headPtr->setNextNodePtr(nextNodePtr);
    }
    else{
        headPtr->setNextNodePtr(0);
        tailPtr->setPrevNodePtr(0);
    }

    return poppedData;

}
```

If there is at least one node in the Queue before Dequeue

Retrieve the nextNodePtr for the First node

There is more than one node in the Queue before Dequeue. Set the next node of the first node as the new first node and make the headPtr point to it as its next node. Set the prevNodePtr of the new first node to null. (Scenario 2)

(3) (4)

(1)

(2) **There is going to be no node in the Queue after the Dequeue operation (Scenario 1).**

```cpp
int peek(){

        Node* firstNodePtr = headPtr->getNextNodePtr();

        if (firstNodePtr != 0)
                return firstNodePtr->getData();
        else
                return -100000; //empty queue

}
```

Code 4.2 (C++)

Code to get the Queue Length
(outside the Queue class)

```cpp
int getQueueLength(Queue queue){

    int length = 0;

    Node* headPtr = queue.getHeadPtr();
    Node* currentNodePtr = headPtr->getNextNodePtr();

    while (currentNodePtr != 0){
        length++;
        currentNodePtr = currentNodePtr->getNextNodePtr();
    }

    return length;

}
```

Code 4.2 (C++)

Printing the Contents of the Queue
without traversing it using the head node or tail node

- Assumption:
- The function getQueueLength(Queue queue) is available to get the length of a Queue object (say, queue).
- Idea:
  - Run iterations corresponding to the length of the queue.
  - In each iteration, first dequeue the queue, print the dequeued data, and then enqueue the dequeued data
- Example

					Printed Contents
Queue (Initial):	11	4	5	8	
Queue (It # 1):	4	5	8	11	11
Queue (It # 2):	5	8	11	4	11 4
Queue (It # 3):	8	11	4	5	11 4 5
Queue (It # 4):	11	4	5	8	11 4 5 8

Printing the Contents of the Queue
without traversing it using the head node or tail node

```cpp
void Print(Queue queue){

        int queueLength = getQueueLength(queue);

        for (int index = 0; index < queueLength; index++){
                int dataDequeued = queue.dequeue();
                cout << dataDequeued << " ";
                queue.enqueue(dataDequeued);
        }

    cout << endl;

}
```

Code 4.2 (C++)