Data Structures and Algorithms

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 5

Queue

Array Based

Course Roadmap

Part 1: Linear Data Structures

Lecture 1: Complexity Analysis and Recursion

Lecture 2: Arrays

Lecture 3: Linked List

Lecture 4: Stack

Lecture 5: Queue

Lecture 6: Deque

Lecture 7: STL in C++ (Linear Data Structures)

Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to Queue

2- Insertion Operation

3- Deletion Operation

4- Front & Back Operations

5- Time Complexity & Space Complexity


Let's
START UP

Lecture Agenda

Section 1: Introduction to Queue

Section 2: Insertion Operation

Section 3: Deletion Operation

Section 4: Front & Back Operations

Section 5: Time Complexity & Space Complexity

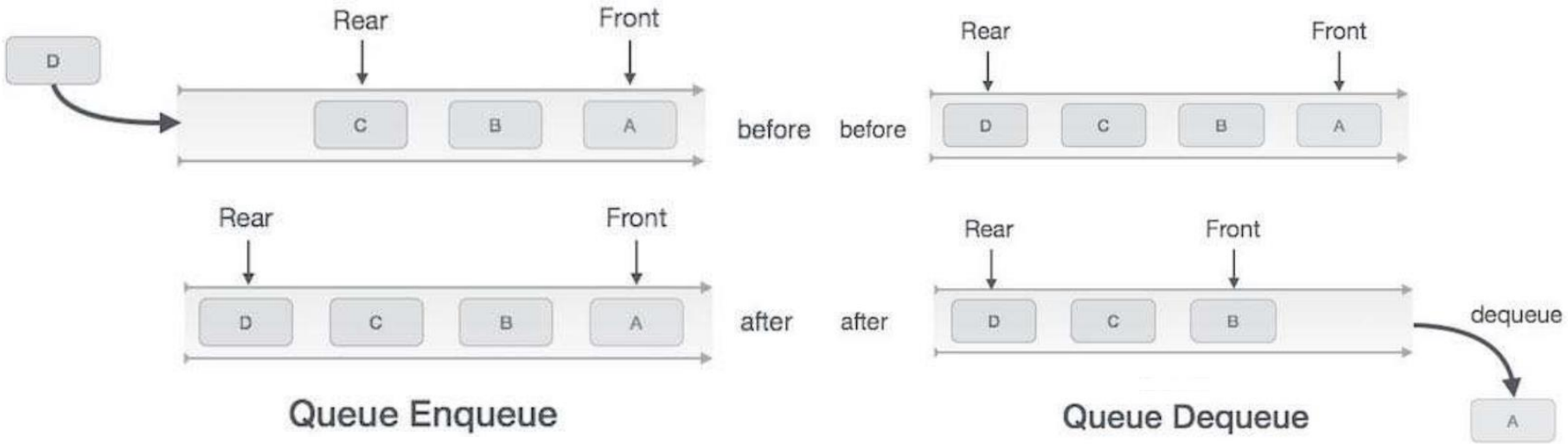
Introduction to Queue

- A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
- A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence. By convention, the end of the sequence at which elements are added is called the back, tail, or rear of the queue, and the end at which elements are removed is called the head or front of the queue, analogously to the words used when people line up to wait for goods or services.
- The operations of adding an element to the rear of the queue is known as enqueue, and the operation of removing an element from the front is known as dequeue.

Introduction to Queue

- Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues

- enqueue() - add (store) an item to the queue.
- dequeue() - remove (access) an item from the queue.



Introduction to Queue

- Queue is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the REAR(also called tail), and the removal of existing element takes place from the other end called as FRONT(also called head). This makes queue as FIFO (First in First Out) data structure, which means that element inserted first will be removed first.
- The operations of a queue make it a first-in-first-out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. A queue is an example of a linear data structure, or more abstractly a sequential collection.

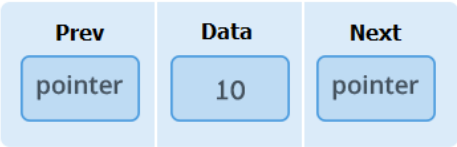
Introduction to Queue

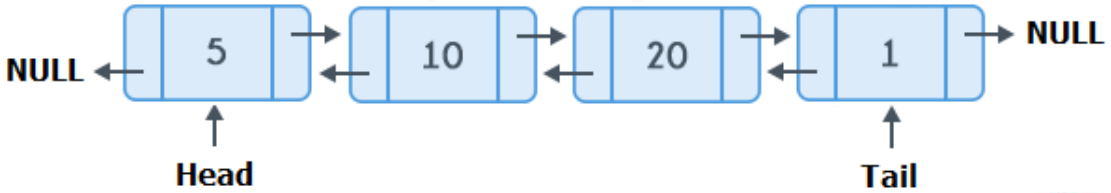
➢ Following are the basic operations supported by a queue.

- Push: which adds an element to the collection.
- Pop: which removes the most recently added element that was not yet removed.
- Front & Back: which gets the first element and the last element in the queue.

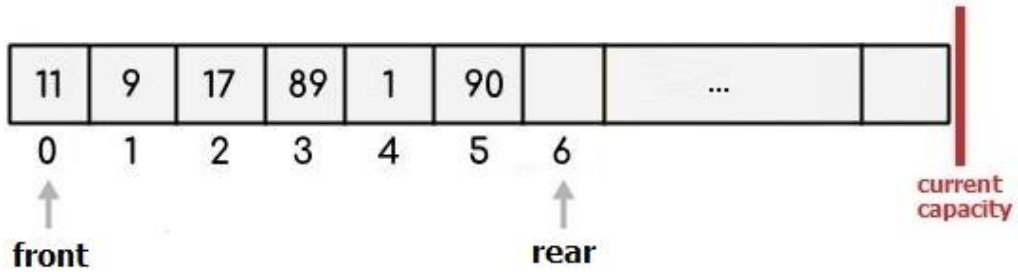
➢ Queue Types

1. Queue (Linked List Based)





2. Queue (Array Based)



Reserve Method - Queue (Array Based)

```
// Initialize a queue with dynamic length
int front_idx, rear_idx;
int capacity;
int* arr;

// This function updates the capacity of the queue
void reserve(int new_capacity) {
    // Initialize a new queue with the new capacity
    int* temp = new int[new_capacity];
    // copy the elements in the current queue to the new queue
    for (int i = 0; i < rear_idx; i++)
        temp[i] = arr[i];
    // delete the old queue
    delete[] arr;
    // set the temp queue with new capacity to be the queue
    arr = temp;
    // set the current capacity of the queue to be the new capacity
    capacity = new_capacity;
}
```

Queue-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Queue
- **Section 2: Insertion Operation**
- Section 3: Deletion Operation
- Section 4: Front & Back Operations
- Section 5: Time Complexity & Space Complexity

Insertion Operation - Queue (Array Based)

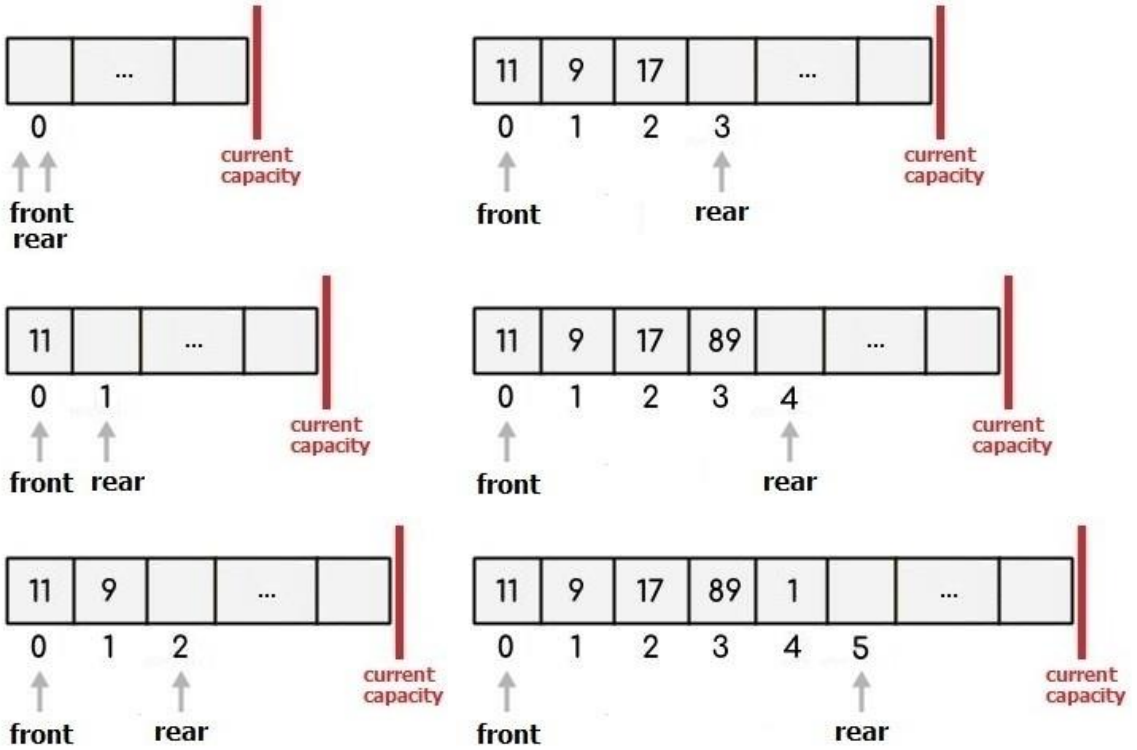
- Insert Operation is to add (store) an item to the queue.

- $\boldsymbol{Insertion\ Algorithm}$:

If the array is full increase the capacity

1. $arr[rear] = data$
2. $rear = rear + 1$

- Insert 11, 9, 17, 89, 1



Insertion Operation - Queue (Array Based)

```
// This function inserts an element at the begin of the queue
void push(int new_data) {
    // check if we need to update the capacity of the queue
    if (rear_idx == capacity)
        reserve(2 * capacity + 1);
    // insert the new element
    arr[rear_idx] = new_data;
    // update the rear_idx of the queue
    rear_idx = rear_idx + 1;
}
```

Queue-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- **Section 3: Deletion Operation**
- Section 4: Front & Back Operations
- Section 5: Time Complexity & Space Complexity

Deletion Operation - Queue (Array Based)

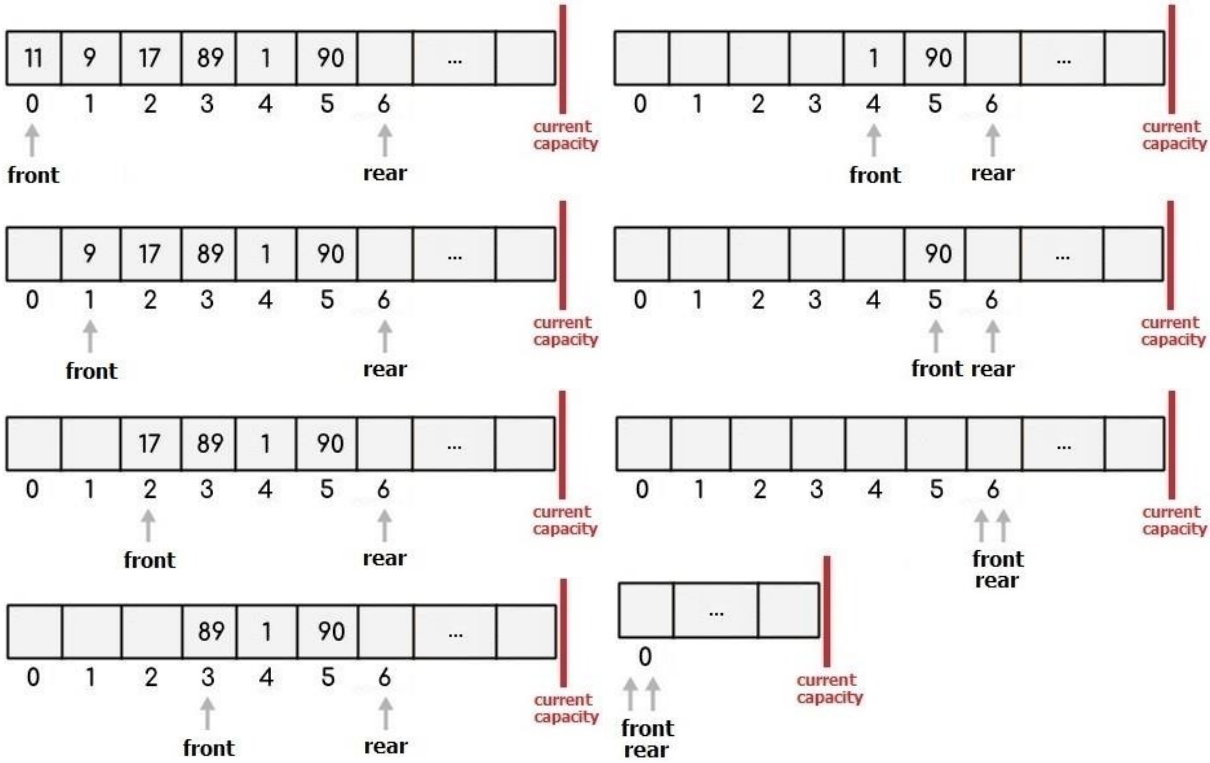
- Delete Operation remove (access) an item from the queue.

- ***Deletion Algorithm***:

1. $front = front + 1$
2. $if\ front == rear\ then$
3. $front = rear = 0$

If the array is half full decrease the capacity

- Delete 11, 9, 17, 89, 1, 90



Deletion Operation - Queue (Array Based)

```cpp
// This function deletes the first element in the queue
void pop() {
    // check if the queue is empty
    if (rear_idx <= front_idx)
        return;
    // update the front_idx of the queue
    front_idx = front_idx + 1;
    // if the queue is empty reset front_idx & rear_idx
    if (front_idx == rear_idx)
        front_idx = rear_idx = 0;
    // check if we need to update the capacity of the queue
    if (rear_idx < capacity / 2)
        reserve(capacity / 2);
}
```

Queue-Array-Based.cpp

Lecture Agenda

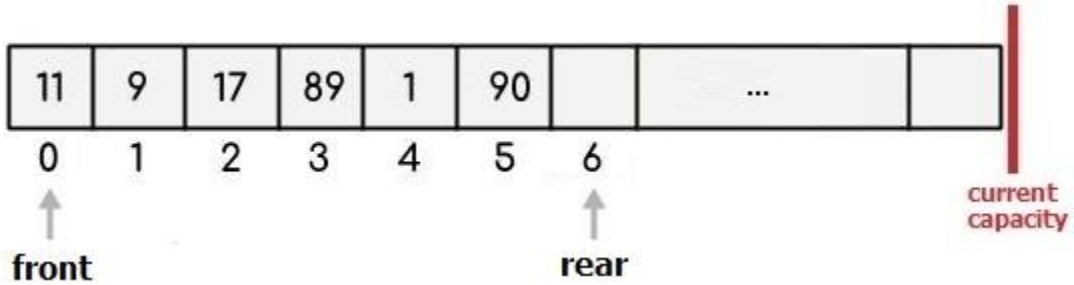
- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- **Section 4: Front & Back Operations**
- Section 5: Time Complexity & Space Complexity

Front & Back Operations - Queue (Array Based)

- Front Operation gets the first item in the queue.
- Front 11

- ***Front Algorithm***:

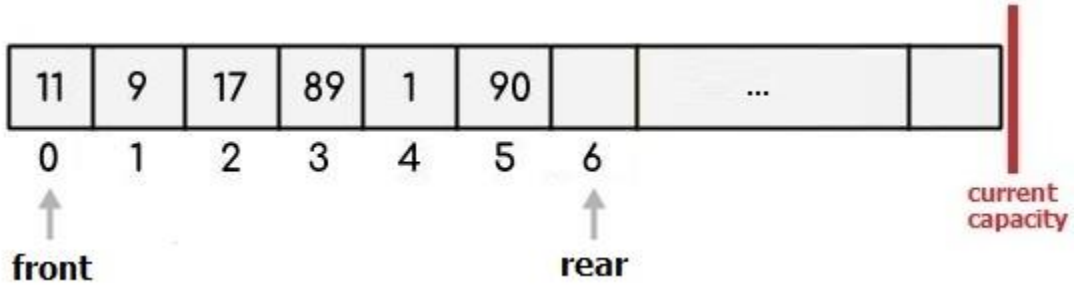
1. ***return arr[front]***



- Back Operation gets the last item in the queue.
- Back 90

- ***Back Algorithm***:

1. ***return arr[rear − 1]***



Front & Back Operations - Queue (Array Based)

```cpp
// This function returns the value of the first element in the queue
int front() {
    // check if the queue is empty
    // to return the biggest integer value as an invalid value
    if (rear_idx <= front_idx)
        return INT_MAX;
    // otherwise return the real value
    else
        return arr[front_idx];
}
// This function returns the value of the last element in the queue
int back() {
    // check if the queue is empty
    // to return the biggest integer value as an invalid value
    if (rear_idx <= front_idx)
        return INT_MAX;
    // otherwise return the real value
    else
        return arr[rear_idx - 1];
}
```

Queue-Array-Based.cpp

Functionality Testing - Queue (Array Based)

- Initialize a global array

```
#include <bits/stdc++.h>
using namespace std;

// Initialize a queue with dynamic length
int front_idx, rear_idx;
int capacity;
int* arr;
```

- In the Main function:

```
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
```

- Expected Output:

```
Queue front: 2147483647 and Queue back: 2147483647
```

Queue-Array-Based.cpp

Functionality Testing - Queue (Array Based)

- In the Main function:

```
push(10);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(20);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(30);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(40);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(50);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
```

- Expected Output:

```
Queue front: 10 and Queue back: 10
Queue front: 10 and Queue back: 20
Queue front: 10 and Queue back: 30
Queue front: 10 and Queue back: 40
Queue front: 10 and Queue back: 50
```

Queue-Array-Based.cpp

Functionality Testing - Queue (Array Based)

- In the Main function:

```
while (rear_idx > 0) {
    cout << "Queue front: " << front()
         << " and Queue back: " << back() << '\n';
    pop();
    cout << "queue front has been deleted\n";
}
cout << "Queue is empty now\n";
```

- Expected Output:

```
Queue front: 10 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 20 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 30 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 40 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 50 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue is empty now
```

Queue-Array-Based.cpp

Functionality Testing - Queue (Array Based)

- In the Main function:

```
push(10);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(20);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(30);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
```

- Expected Output:

```
Queue front: 10 and Queue back: 10
Queue front: 10 and Queue back: 20
Queue front: 10 and Queue back: 30
```

Queue-Array-Based.cpp

Functionality Testing - Queue (Array Based)

- In the Main function:

```
while (rear_idx > front_idx) {
    cout << "Queue front: " << front()
         << " and Queue back: " << back() << '\n';
    pop();
    cout << "queue front has been deleted\n";
    cout << "-----------------------------------\n";
}
cout << "Queue is empty now\n";
```

- Expected Output:

```
Queue front: 10 and Queue back: 30
queue front has been deleted
-----------------------------------
Queue front: 20 and Queue back: 30
queue front has been deleted
-----------------------------------
Queue front: 30 and Queue back: 30
queue front has been deleted
-----------------------------------
Queue is empty now
```

Queue-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations

Section 5: Time Complexity & Space Complexity

Time Complexity & Space Complexity

- Time Analysis

	Worst Case	Average Case
• Push	Θ(n)	θ(1)
• Pop	Θ(n)	θ(1)
• Front	Θ(1)	θ(1)
• Back	Θ(1)	θ(1)

Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations
- ✓ Section 5: Time Complexity & Space Complexity

Practice

Practice

1- Sliding Window Maximum summation of all sub-arrays of size k
2- Check string is palindrome or not
3- Generate Binary Numbers from 1 to n
4- Reversing a queue using recursion
5- Reversing the first K elements of a Queue
6- Find the largest multiple of 3
7- Smallest multiple of a given number made of digits 0 and 9 only
8- Delete all elements in the queue
9- Sum of minimum and maximum elements of all subarrays of size k
10- First negative integer in every window of size k

Assignment

Implement STL Queue

- queues are a type of container adaptor, specifically designed to operate in a (first-in first-out) context, where elements are inserted into one end of the container and extracted from the other.
- queues are implemented as containers adaptors, which are classes that use an encapsulated object of a specific container class as its underlying container, providing a specific set of member functions to access its elements. Elements are pushed into the "back" of the specific container and popped from its "front".
- Queues are a type of container adaptors which operate in a first in first out type of arrangement. Elements are inserted at the back (end) and are deleted from the front.

More Info: cplusplus.com/reference/queue/queue/
More Info: en.cppreference.com/w/cpp/container/queue
More Info: geeksforgeeks.org/queue-cpp-stl/

Implement STL Queue

- Member functions: (constructor) Construct queue (public member function)
  (empty) Test whether container is empty (public member function)
  (size) Return size (public member function)
  (front) Access next element (public member function)
  (back) Access last element (public member function)
  (push) Insert element (public member function)
  (pop) Remove top element (public member function)
  (swap) Swap contents (public member function)

More Info: cplusplus.com/reference/queue/queue/
More Info: en.cppreference.com/w/cpp/container/queue


DO
MORE.