

## Review

- What is Stack
- Functions for managing Stack
  - Push
  - Pop
  - IsEmpty
  - IsFull
  - Top
- Stack Implementation with Array
- Stack Implementation with Linked List

## Preview

- How to compile a program with template classes and functions
  - Create a instantiation file
- Queue
  - What is Queue
  - Functions for managing a queue
    - enqueue
    - dequeue
    - emptyQueue
    - fullQueue
    - frontQueue
  - Queue Implementation with Array and Linked-List

## Compiling Template Classes

- Since we can use any type of data with template classes, we need provide this information to compiler to create an executable code.
- There are several way we can compile a program with template classes.
- Simple way is create a instantiation program file.
- Instantiation file include all program files and information what kind of data is used with template classes.

## Compiling Template Classes

```
// Instantiation.cpp
// Program files need to be included
#include "Students.cpp"
#include "Stack.cpp"
#include "Lab4.cpp"

//template with different data types
template struct StackNode<int>;
template class Stack<int>;
template struct StackNode < *Student >;
template class Stack < *Student >;
```

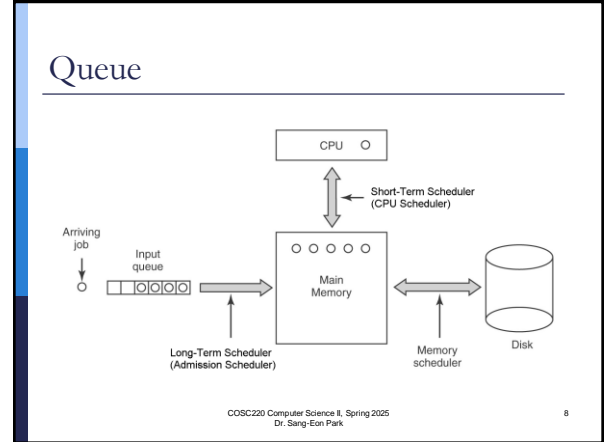
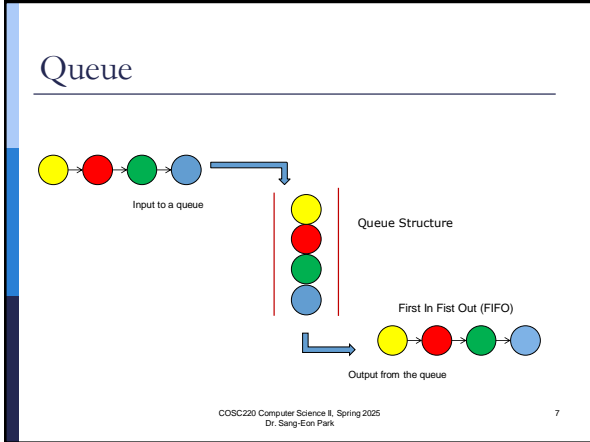
Now you can compile your program  
g++ -o stack instantiation.cpp

## Queue



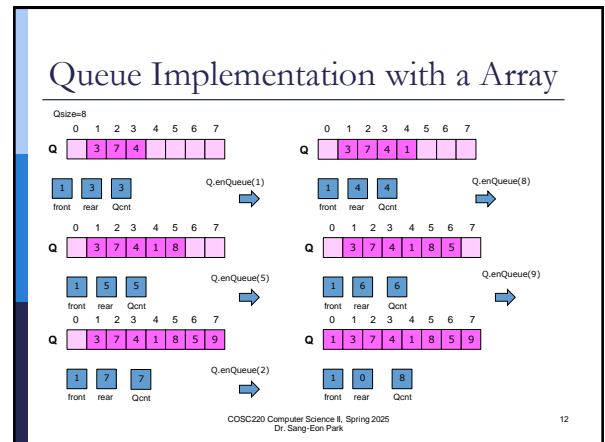
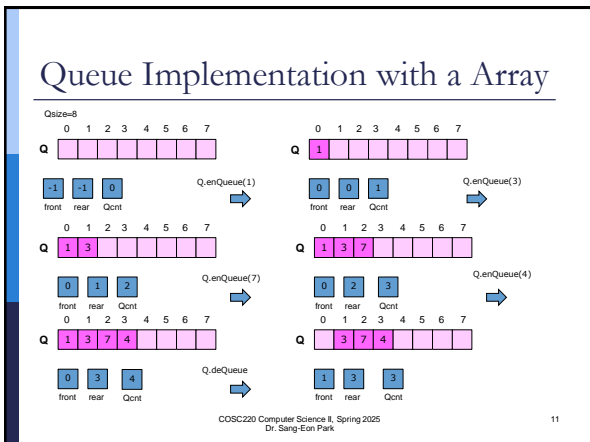
## Queue

- **Queues** are an abstract data type, specifically designed to operate in a FIFO context (first-in first-out).
- Elements are inserted into one end of the **Queue** and extracted from the other.
- It can be used for a process scheduler in operating system
- A **Queue** is used in breadth first search



- ### Queue
- Member Functions for managing a Queue
    - emptyQueue – return true if a queue is empty else return false
    - fullQueue – return true if a queue is full else return false.
    - frontQueue – return the first element in the Queue
    - enqueue – insert a new element at the end of the Queue.
    - dequeue – remove a element from the front of the Queue
- COSC220 Computer Science II, Spring 2025  
Dr. Sang-Eon Park 9

- ### Queue Implementation with a Array
- Since the characteristics of the Queue, we need extra works to keep the front and queue index.
  - It is always possible rear index is less then front index after calling dequeue and enqueue several times.
- COSC220 Computer Science II, Spring 2025  
Dr. Sang-Eon Park 10



### Queue Implementation with a Array

Queue operations using an array of size 8:

Initial state: Q: [1, 3, 7, 4, 1, 8, 5, 9], front: 1, rear: 0, Qcnt: 8

Q.dequeue: front becomes 2, rear: 0, Qcnt: 7

Q.enqueue(5): front: 2, rear becomes 1, Qcnt: 7

Q.dequeue: front becomes 3, rear: 1, Qcnt: 6

- Before enqueue new element in a Queue, need to check whether a Queue is full or not.
- Before dequeue a element from the Queue, need to check whether Queue is empty or not

COSC220 Computer Science II, Spring 2025  
Dr. Sang-Eon Park 13

### Queue Implementation with a Linked List

Queue operations using a linked list:

Initial state: rear, front, Qcnt all point to null.

enqueue(5): rear and front point to node 5, Qcnt becomes 1.

enqueue(1): rear points to node 1, front points to node 5, Qcnt becomes 2.

enqueue(4): rear points to node 4, front points to node 1, Qcnt becomes 3.

enqueue(6): rear points to node 6, front points to node 4, Qcnt becomes 4.

COSC220 Computer Science II, Spring 2025  
Dr. Sang-Eon Park 14

### Queue Implementation with a Linked List

Dequeue operation:

Initial state: rear points to node 6, front points to node 4, Qcnt: 4.

Dequeue: front moves to node 1, Qcnt becomes 3.

- Before enqueue new element in a Queue, need to check whether a Queue is full or not.
- Before dequeue a element from the Queue, need to check whether Queue is empty or not

COSC220 Computer Science II, Spring 2025  
Dr. Sang-Eon Park 15

```

#ifndef QUEUE_H
#include QUEUE_H

template <class ItemType>
struct QueueNode
{
    ItemType data;
    QueueNode<ItemType> *next;
};

template <class ItemType>
class Queue
{
private:
    QueueNode<ItemType> *front;
    QueueNode<ItemType> *back;
    int len;
    int queueCnt;
public:
    Queue(); //class constructor - initialize variables
    ~Queue(); //class destructor - return memory used by queue elements
    void enqueue(const ItemType); //add an item to the back of the queue
    ItemType dequeue(); //remove the first item from the queue and return its value
    ItemType first() const; //return the value of the first item in the queue
    bool isEmpty() const; //returns true if a Queue is empty
    bool isFull() const; //returns true if a Queue is Full
    int length() const; //returns the amount of elements in the queue
};
#endif
    
```

COSC220 Computer Science II, Spring 2025  
Dr. Sang-Eon Park 16