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

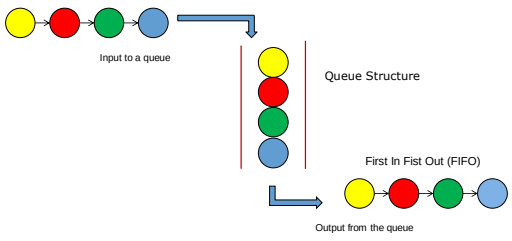
g++ -o stack instantiation.cpp

Queue

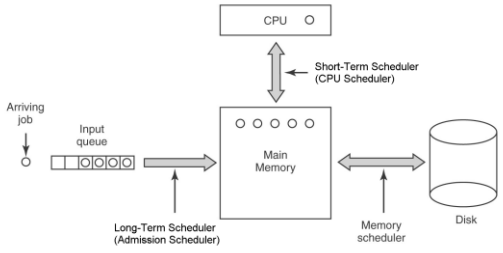
Queue

- **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



Queue



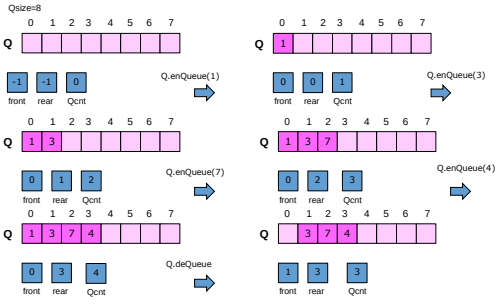
Queue

- Member Functions for managing a Queue
  - emptyQueue – return true if a queue is empty else return faluse
  - 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

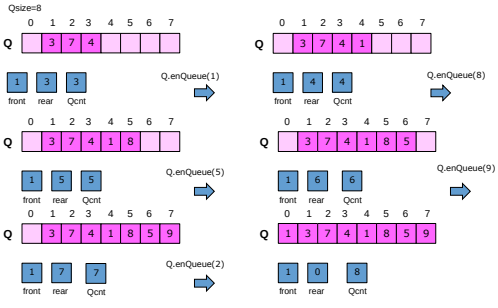
Queue Implementation with a Array

- 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.

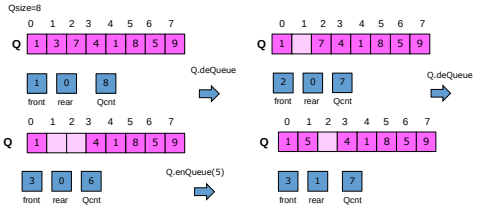
Queue Implementation with a Array



Queue Implementation with a Array

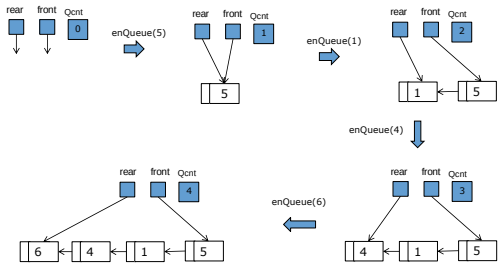


Queue Implementation with a Array

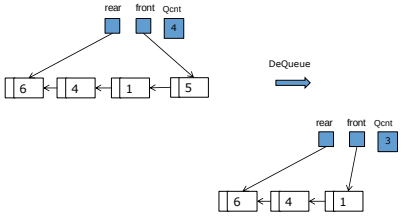


- Before enQueue new element in a Queue, need to check whether a Queue is full or not.
- Before deQueue a element from the Queue, need to check whether Queue is empty or not

Queue Implementation with a Linked List



Queue Implementation with a Linked List



- 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


```
#ifndef QUEUE_H
#include QUEUE_H

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


template <class ItemType>
class Queue
{
private:
   QueNode<ItemType> *front;
   QueNode<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
```