**Stack ADT**

In computer science, a Stack is an abstract data type that serves as a collection of elements, with two main principal operations: **PUSH**, which adds an element to the collection, and **POP**, which removes the most recently added element that was not yet removed.

Stack works on LIFO principle. **LIFO** stands for **Last-in-first-out**. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation.

**Stack’s pictorial representation:**



We try to implement this behavior using classes. An interface (header file only) of Stack class is given below:

**Header file of an integer stack:**

class Stack

{

int \* arr ; // dynamic array to store elements

int size; // size of Stack

int currLoc; // No. of elements currently stored in Stack (initially ZERO)

public:

Stack(); // making a Stack of maxsize = 10;

Stack(int); // making a Stack of user defined size

void push(int num); // this function will add a number into stack maintaining LIFO behavior

int pop(); // this function will remove a number from stack maintaining LIFO behavior

bool IsFull() // return true if stack is full otherwise return false

bool IsEmpty() // return true if stack is empty otherwise return false

int getTop(); // get the top most element of the stack without removing it.

~Stack(); // release all dynamic memories hold by Stack Object

};

If we implement above given class and then make object of Stack in main() as given below:

int main()

{

Stack st\_obj(5); // Stack of size 5

st\_obj.push(3);

// After inserting 3 into stack, it will be like:

|  |
| --- |
|  |
|  |
|  |
|  |
| 3 |

st\_obj.push(6);

// After inserting 6 into stack, it will be like:

|  |
| --- |
|  |
|  |
|  |
| 6 |
| 3 |

st\_obj.push(4);

// After inserting 4 into stack, it will be like:

|  |
| --- |
|  |
|  |
| 4 |
| 6 |
| 3 |

int value = st\_obj.pop();

// After removing one value from stack, it will be like:

|  |
| --- |
|  |
|  |
|  |
| 6 |
| 3 |

st\_obj.push(-9);

// After inserting -9 into stack, it will be like:

|  |
| --- |
|  |
|  |
| -9 |
| 6 |
| 3 |

}

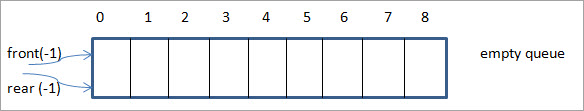
**TASK FOR YOU:**

The above class of Stack is for the integer data type only which means it is only capable of making stack of integers. You now have to implement a **class template** for Stack which can perform all those operations given above. (Both header and CPP file)

**Queue ADT**

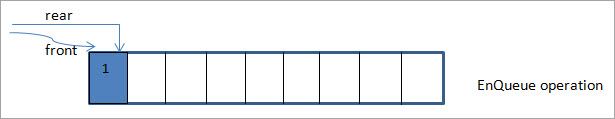
The queue is a basic data structure just like a stack. In contrast to stack that uses the LIFO approach, queue uses the FIFO (first in, first out) approach i.e the element that is inserted first is also deleted first. Basically in queue, insertion take place at one end called rear end and deletion can take place from other end called front end.

**Illustration of Queue:**

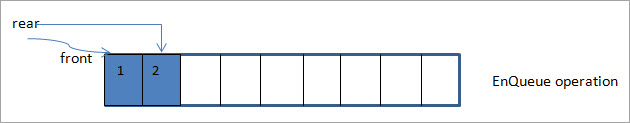


This is an empty queue and thus we have rear and front set to -1.

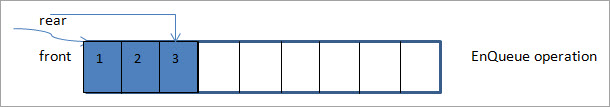
Next, we add 1 to the queue and as a result, the rear pointer moves ahead by one location.



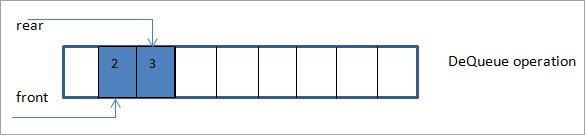
In the next figure, we add element 2 to the queue by moving the rear pointer ahead by another increment.



In the following figure, we add element 3 and move the rear pointer by 1.



Next, we delete the element pointed by the front pointer. As the front pointer is at 0, the element that is deleted is 1.



Thus the first element entered in the queue i.e. 1 happens to be the first element removed from the queue. As a result, after the first dequeue, the front pointer now will be moved ahead to the next location which is 1.

We try to implement this behavior using classes. An interface (header file only) of Queue class is given below:

**Header file of an integer queue:**

class Queue

{

int \* arr ; // dynamic array to store elements

int size; // size of Queue

int currLoc; // No. of elements currently stored in Queue (initially ZERO)

int front; // front element of the queue

int rear; // last element of the queue

public:

Queue(); // making a Queue of maxsize = 10;

Queue(int); // making a Queue of user defined size

void EnQueue (int num); // this function will add a number into queue maintaining FIFO behavior

int DeQueue(); // this function will remove a number from queue maintaining FIFO behavior

bool IsFull() // return true if queue is full otherwise return false

bool IsEmpty() // return true if queue is empty otherwise return false

~Queue(); // release all dynamic memories hold by Stack Object

};

If we implement above given class and then make object of Queue in main() as given below:

int main()

{

Queue qu\_obj(5); // Queue of size 5

qu\_obj.EnQueue(3);

// After inserting 3 into queue, it will be like:

|  |
| --- |
|  |
|  |
|  |
|  |
| 3 |

qu\_obj.EnQueue(6);

// After inserting 6 into queue, it will be like:

|  |
| --- |
|  |
|  |
|  |
| 6 |
| 3 |

qu\_obj.EnQueue(4);

// After inserting 4 into queue, it will be like:

|  |
| --- |
|  |
|  |
| 4 |
| 6 |
| 3 |

int value = qu\_obj.DeQueue();

// After removing one value from queue, it will be like:

|  |
| --- |
|  |
|  |
|  |
| 4 |
| 6 |

}

**TASK FOR YOU:**

The above class of Queue is for the integer data type only which means it is only capable of making queue of integers. You now have to implement a **class template** for Queue which can perform all those operations given above. (Both header and CPP file)