Lab 06 Queue Implementation with Applications

Learning Outcomes

After completing the lab students will be able to:

- Implement queues using linked lists.
- Use queues to solve certain problems such as finding the shortest path in a graph.

Pre-Lab Reading:

A queue is a special case of a singly-linked list which works according to First In First Out (FIFO) algorithm. An array implementation of a queue is also possible but it is not discussed here as it does not allow dynamic memory allocation.

Data is inserted at one end called the **'Rear'** of the queue and deleted from the other end called the **'Front'** of the queue. Operations supported in a queue are:

Enqueue (insert)	− Insert data at the **'Rear'** of the queue.
Dequeue (delete)	− Delete a data element from the **'Front'** of the queue.
Peek	− Read the element at the Front of the queue without removing it.

Read Chapter 8 from "Data Structures using C", by Reema Thareja, 2nd edition, for more information on queues, priority queues, their implementation and applications.

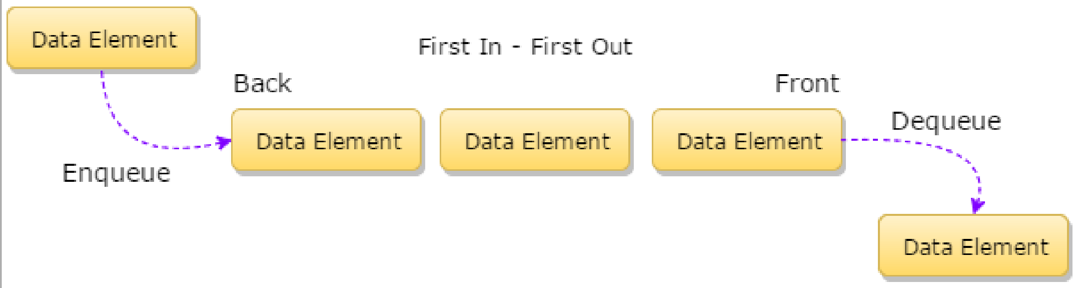


Figure 1: Queue Abstract Data Type

Skeleton code for queues is provided with this lab. As a pre-lab task implement the basic queue functions (enqueue, dequeue and peek). You will need them later on for completing the in-lab tasks. Your implementation of these functions should make sure that no restrictions (prior pointer settings for boundary cases) are enforced on the way the enqueue and dequeue functions are called.

Making a new queue should be as simple as declaring two pointers for the front and rear of the queue and call enqueue() and dequeue() functions to add/remove elements from the queue.

Pre-lab Task : Complete the functions 'enqueue()' , 'dequeue()' and peek() functions.

You are provided skeleton code for basic Queue implementation functions, enqueue(), dequeue() and peek(). Peek and enqueue are already implemented. You have to complete the ***dequeue()*** function. Also write the **main()** function to demonstrate correct working of the queue functions.

In-Lab Task 1: Implement a priority queue.

Implement a priority queue with following functions.

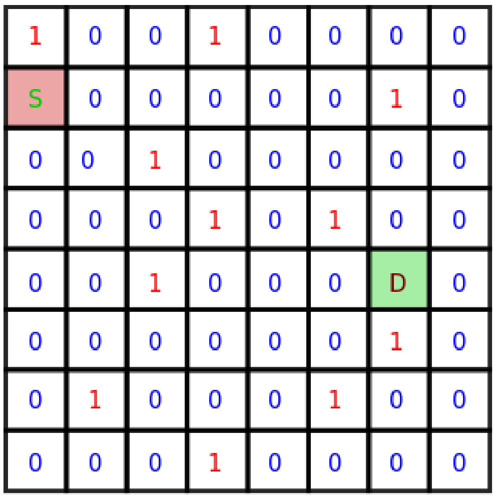
```
void pr_enqueue(struct node ** front, struct node ** rear, struct element new_data);

struct element pr_dequeue(struct node ** front);    // This function has been implemented

int pr_isEmpty(struct node ** front);               // This function has been implemented
```

This implementation constructs the queue in a sorted manner. This means that when removing items from the queue we only need to remove the first item from the front end. But when we want to add an item (using enqueue) we need to place it to its proper location based on its priority.

Skeleton code is provided. You will have to implement only the '***enqueue()***' function.

In-Lab Task 2: Find the Shortest Path in Graphs Using BFS and Queues.

For this task you are provided with the skeleton code that generates data into a 2D array populating it randomly with zeros (0) and ones (1). There will be more zeros than ones. This is by design. The code finds the shortest path between the source cell and the destination cell. The only movements allowed in the grid are top, bottom, right and left. This means only immediate four neighbors of a given cell can be visited. In the following figure the distance between the source (S) cell and the destination (D) cell is 9 steps. This calculation does not take into account the contents of the cells.



1	0	0	1	0	0	0	0
S	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	D	0
0	0	0	0	0	0	1	0
0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0

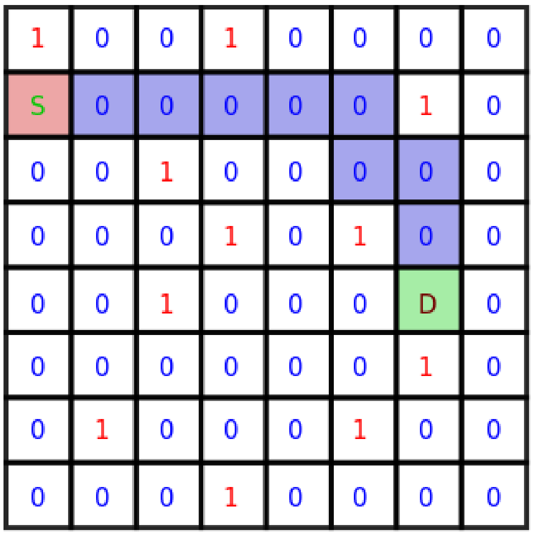
Figure 2: A 2D grid with source and destination cells marked

The skeleton code provided uses the Lee Algorithm and uses Breadth First Search. The algorithm is given below.

```
We start from the source cell and calls BFS procedure.
We maintain a queue to store the coordinates of the matrix and initialize it with the source cell.
We also maintain an integer array 'visited' of same size as our input matrix and initialize all its elements to 0.
We also maintain an integer array 'dist' of same size as our input matrix and initialize all its elements to 1000.
We LOOP till queue is not empty
    Dequeue front cell from the queue
    Return the value in dist array cell if the destination coordinates match.
For each of its four adjacent cells, if they are not visited yet, we enqueue it in the queue and also mark them as visited. We also add 1 to the dist array for the visited cell.
```

Your task is to **modify the given code** and find the cost of moving from the source cell to the destination cell **considering that only cell with a zero ‘0’ may be visited**. This way the shortest path for this particular grid will be 9 as shown in Figure 3.

Your code should return the shortest distance between the source and destination cells or (-1) if no path exists between them.



1	0	0	1	0	0	0	0
S	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	D	0
0	0	0	0	0	0	1	0
0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0

Figure 3: Shortest path shown in blue

References:

1. https://www.youtube.com/watch?v=KiCBXu4P-2Y
2. https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/
3. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/