**Data Structures and Algorithms**

**Lab 6**

**Submitted To:**

Mr. DilshadSabir

**Submitted By:**

ManaalWaseem

FA18-BCE-074

**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 queuefunctions.**

**Program:** In this program, function “***dequeue()”*** takes queue’s front as argument. Its data part is stored in ***“temp”*** of type ***struct* element**. Pointers are set and front is freed. The function returns ***“temp”*** which is printed on console through **main**. **Main** provides the user with menu to navigate through program.

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include "node.h"

4 #include "queue\_functions.h"

5

6 **void**flush();

7

8 **int**main()

9 {

10 **struct**node \* qfront= NULL; **/// This is the top of the stack**

11 **struct**node \* qrear= NULL; **/// This is the rear of the stack**

12

13 **int**choice;

14

15 **while**(1)

16 {

17 printf("\nWhat do you want to do now?\n");

18

19 printf("\n1. Enqueue.");

20 printf("\n2. Dequeue.");

21 printf("\n3. Peak.");

22 printf("\n4. Exit the menu.\n");

23

24

25

26 scanf("%d", &choice);

27 flush();

28 **switch** (choice)

29 {

30 **case** 1: **/// Add a new node to the queue**

31 {

32 **struct**element d;

33

34 printf("\nEnterx\_index: ");

35 scanf("%d", &(d.x\_index));

36

37 printf("\nEntery\_index: ");

38 scanf("%d", &(d.y\_index));

39

40 printf("\nEntercell\_cost: ");

41 scanf("%d", &(d.cell\_cost));

42

43 printf("\nRecord entered !\n");

44

45 q\_insert(&qrear, &qfront, d);

46 printf("\n New node inserted. \n\n");

47 **break**;

48 }

49

50 **case** 2: **/// Deletes a new node to the queue**

51 {

52 **struct**element temp;

53 temp = q\_delete(&qfront);

54

55 printf("\n Data of node deleted. \n\n");

56 printf("x\_index:\t %d\n", temp.x\_index);

57 printf("y\_index:\t %d\n", temp.y\_index);

58 printf("cell\_cost:\t %d\n", temp.cell\_cost);

59 break;

60 }

61

62 case 3: /// Check what is at the front of queue

63 {

64 struct element temp;

65 temp = q\_peek(&qfront);

66

67 printf("x\_index:\t %d\n", temp.x\_index);

68 printf("y\_index:\t %d\n", temp.y\_index);

69 printf("cell\_cost:\t %d\n", temp.cell\_cost);

70 break;

71 }

72

73 case 4: /// Exit the program.

74 {

75 printf("\nExiting on user request.\n\n");

76 return(0);

77

78 break;

79 }

80 }

81 }

82 return 0;

83 }

84

85 void flush()

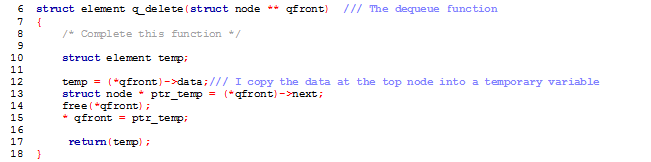
86 {

87 int c;

88 while ((c = getchar()) != '\n' && c != EOF);

89 }

**main.c**

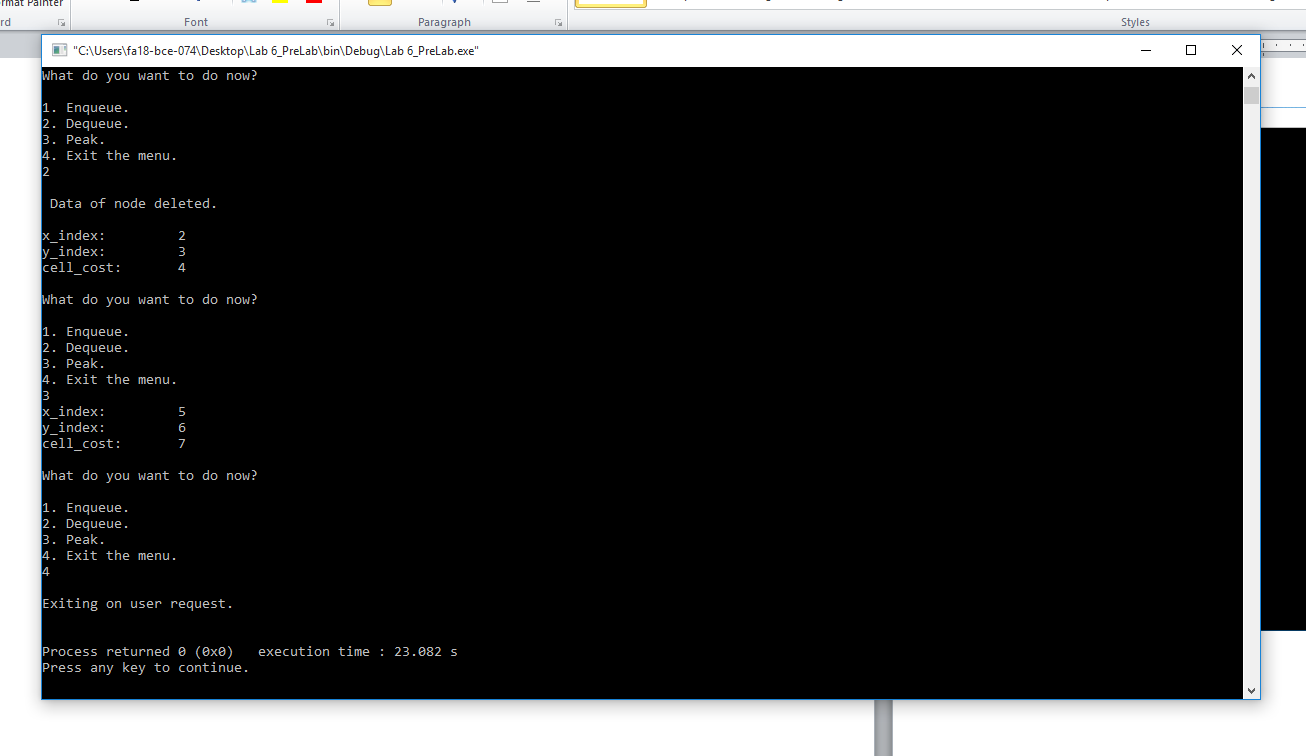


**Dequeue Functionality**

**Output:**



**a.**



**b.**

**In Lab:**

**Task 1:**

***Implement a priority queue.***

**Implement a priority queue with following functions.**

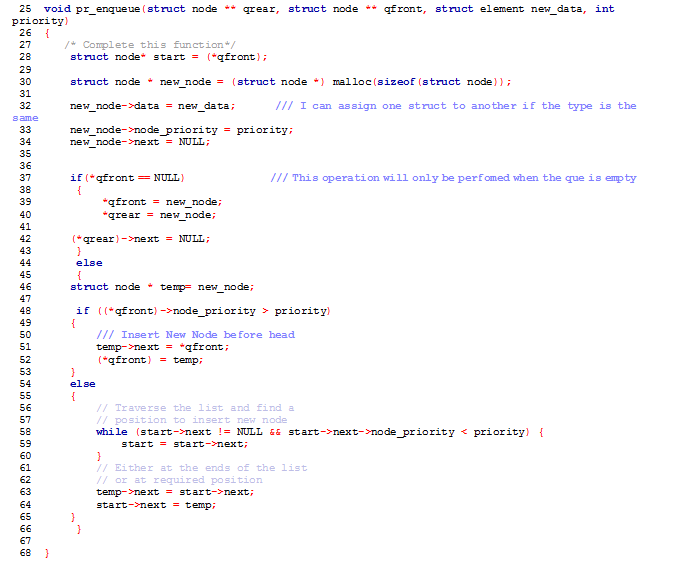
***void pr\_enqueue(struct node \*\* front, struct node \*\* rear, struct element new\_data);***

***struct element pr\_dequeue(struct node \*\* front); // This function has been implemented***

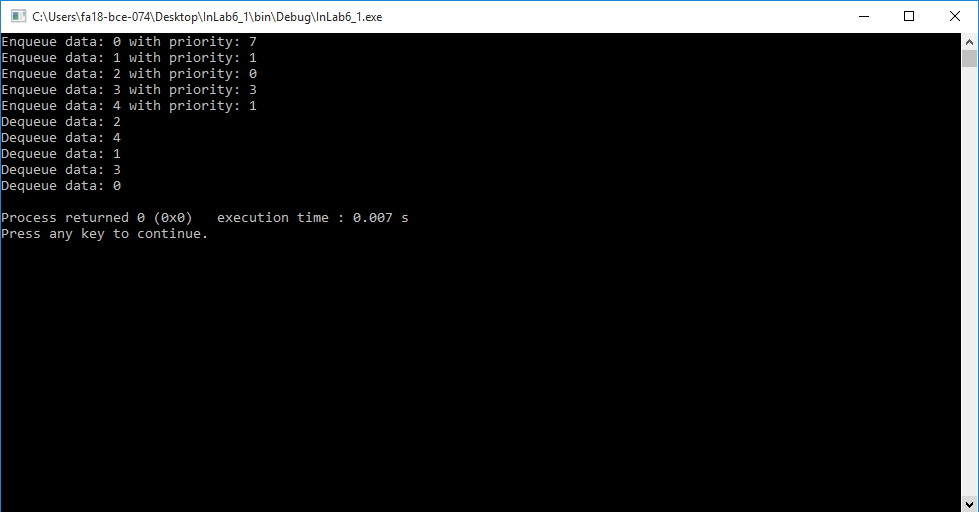
***int pr\_isEmpty(struct node \*\* front); // This function has been implemented***

**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.**

**Program:**In this program, function “***pr\_enqueue”*** takes rear, front , data for new node and priority for newly created node as argument. New node is created via. ***malloc***; if queue is empty, new node is declared as **front** otherwise list is traversed and node is inserted into queue based on its priority.



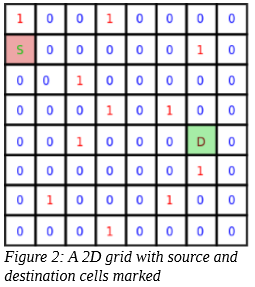
**Output:**



**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.**

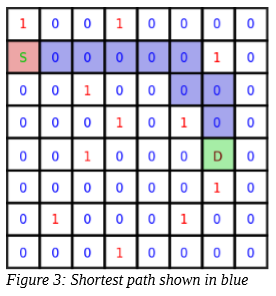


**The skeleton code provided uses the Lee Algorithm and uses Breadth First Search. The algorithm is given below.**

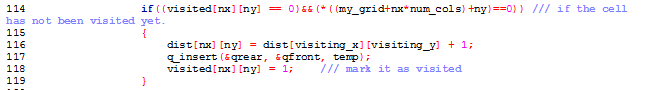
* ***We start from the source cell and calls BFS procedure.***
* ***We maintain a queue to store the coordinates of the matrix and initialize it with the source cell.***
* ***We also maintain an integer array ‘visited’ of same size as our input matrix and initialize all its elements to 0.***
* ***We 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.***

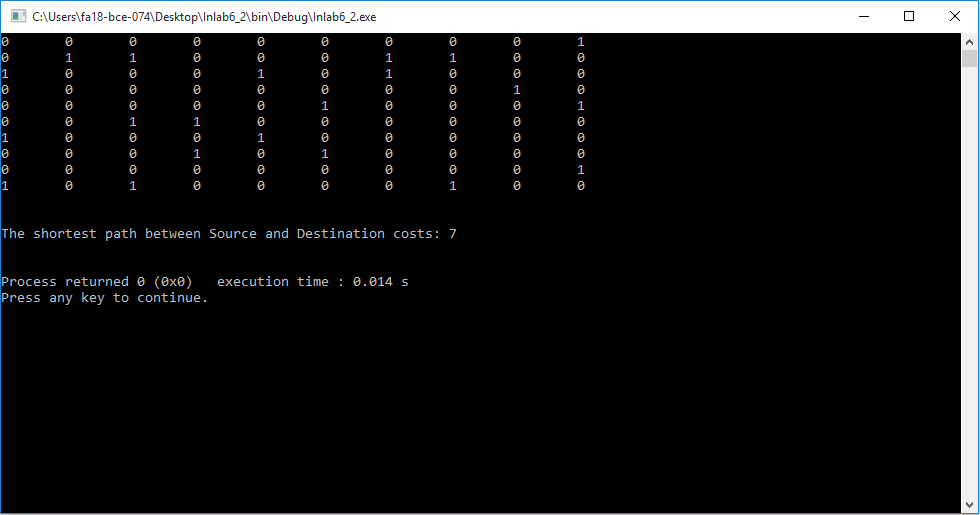
**Your task is to modify the given code and find the cost of moving from the source cell to the destination cell considering that only cell with a zero ‘0’ may be visited. This way the shortest path for this particular grid will be 9 as shown in Figure 3. Your code should return the shortest distance between the source and destination cells or (-1) if no path exists between them.**



**Program:** In this program, we have set the offset such that only cells with zeros are visited.



**Output:**



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**THE END**