

# DSA LAB

## ESE

Submitted by:

**Muhammad Abdullah Zafar**

F22607010:

### Q.No.1:

Implement a c++ program for ticket booking system for an event venue utilizing circular queue. With a maximum capacity of 10 tickets, customer arrive to purchase tickets with requests limited to a maximum of 4 tickets each. If a request exceeds available space, customer wait until has queue has a room. tickets are allocated to first come first serve manner, even if split across the circular structure.

Cancellation free up space for subsequent requests

When the queue is full a waiting list is formed prioritizing customers cancellations occur.

As capacity is increased to 15 tickets the system seamlessly integrates changes informing waiting list customers. Concurrent request are handled to ensure safe and consistent ticket allocation.

### Answer:

```
#include<iostream>
#include<vector>
using namespace std;

class TicketBookingSystem {
    int *queue;
    int front, rear, capacity, bookedTickets;
```

```

    vector<int> waitingList;

public:
    TicketBookingSystem(int c) {
        front = rear = -1;
        capacity = c;
        bookedTickets = 0;
        queue = new int[capacity];
    }

    bool isFull() {
        return (rear + 1) % capacity == front;
    }

    bool isEmpty() {
        return front == -1;
    }

    void enqueue(int item) {
        if (isFull()) {
            waitingList.push_back(item);
            cout << "Queue is full. Added customer to waiting list.\n";
            return;
        }
        if (front == -1) front = 0;
        rear = (rear + 1) % capacity;
        queue[rear] = item;
        bookedTickets++;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return;
        }
        cout << "Customer " << queue[front] << " has cancelled.\n";
        if (front == rear) front = rear = -1;
        else front = (front + 1) % capacity;
        bookedTickets--;

        if (!waitingList.empty()) {
            enqueue(waitingList[0]);
            waitingList.erase(waitingList.begin());
        }
    }
}

```

```

        void increaseCapacity(int increase) {
            capacity += increase;
            int *newQueue = new int[capacity];
            for (int i = 0; i <= rear; i++) newQueue[i] = queue[i];
            delete[] queue;
            queue = newQueue;

            while (!isFull() && !waitingList.empty()) {
                enqueue(waitingList[0]);
                waitingList.erase(waitingList.begin());
            }
        }

        int getBookedTickets() {
            return bookedTickets;
        }

        int getWaitingListSize() {
            return waitingList.size();
        }

        int getAvailableTickets() {
            return capacity - bookedTickets;
        }

        void getCustomerIds() {
            if (isEmpty()) {
                cout << "No customers have booked tickets.\n";
                return;
            }
            int i = front;
            do {
                cout << queue[i] << " ";
                i = (i + 1) % capacity;
            } while (i != (rear + 1) % capacity);
            cout << "\n";
        }
    };

int main() {
    TicketBookingSystem tbs(10);
    int choice, item;

    cout<<"\nTicket booking System\n\n";

```

```

while (true) {
    cout << "\n1. Book Tickets\n2. Cancel Booking\n3. Increase Capacity\n4.
Check Booked Tickets\n5. Check Waiting List\n6. Check Available Tickets\n7. Check
Customer IDs\n8. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1:
            cout << "Enter the customer ID: ";
            cin >> item;
            tbs.enqueue(item);
            break;
        case 2:
            tbs.dequeue();
            break;
        case 3:
            cout << "Enter the increase in capacity: ";
            cin >> item;
            tbs.increaseCapacity(item);
            break;
        case 4:
            cout << "Booked Tickets: " << tbs.getBookedTickets() << "\n";
            break;
        case 5:
            cout << "Waiting List Size: " << tbs.getWaitingListSize() <<
"\n";
            break;
        case 6:
            cout << "Available Tickets: " << tbs.getAvailableTickets() <<
"\n";
            break;
        case 7:
            cout << "Customer IDs: ";
            tbs.getCustomerIds();
            break;
        case 8:
            return 0;
        default:
            cout << "Invalid choice. Please enter a valid option.\n";
    }
}

```

```
    return 0;  
}
```

## Output:

```
Ticket booking System  
  
1. Book Tickets  
2. Cancel Booking  
3. Increase Capacity  
4. Check Booked Tickets  
5. Check Waiting List  
6. Check Available Tickets  
7. Check Customer IDs  
8. Exit  
Enter your choice: 1  
Enter the customer ID: 54  
  
1. Book Tickets  
2. Cancel Booking  
3. Increase Capacity  
4. Check Booked Tickets  
5. Check Waiting List  
6. Check Available Tickets  
7. Check Customer IDs  
8. Exit  
Enter your choice: 1  
Enter the customer ID: 78  
  
1. Book Tickets  
2. Cancel Booking  
3. Increase Capacity  
4. Check Booked Tickets  
5. Check Waiting List  
6. Check Available Tickets  
7. Check Customer IDs  
8. Exit  
Enter your choice: 1  
Enter the customer ID: 48  
  
1. Book Tickets  
2. Cancel Booking  
3. Increase Capacity  
4. Check Booked Tickets  
5. Check Waiting List  
6. Check Available Tickets  
7. Check Customer IDs  
8. Exit  
Enter your choice: 4  
Booked Tickets: 3  
  
1. Book Tickets  
2. Cancel Booking  
3. Increase Capacity  
4. Check Booked Tickets  
5. Check Waiting List  
6. Check Available Tickets  
7. Check Customer IDs  
8. Exit  
Enter your choice: 1  
Enter the customer ID: 798
```

1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit

Enter your choice: 1

Enter the customer ID: 78

1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit

Enter your choice: 1

Enter the customer ID: 89

1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit

Enter your choice: 1

Enter the customer ID: 78

1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit

Enter your choice: 1

Enter the customer ID: 56

1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit

Enter your choice: 1

Enter the customer ID: 89

Queue is full. Added customer to waiting list.

```
1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit
Enter your choice: 3
Enter the increase in capacity: 2
```

```
1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit
Enter your choice: 1
Enter the customer ID: 126
```

```
1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit
Enter your choice: 7
Customer IDs: 54 78 48 798 49 45 78 89 78 56 89 126
```

```
1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit
Enter your choice: 6
Available Tickets: 0
```

```
1. Book Tickets
2. Cancel Booking
3. Increase Capacity
4. Check Booked Tickets
5. Check Waiting List
6. Check Available Tickets
7. Check Customer IDs
8. Exit
```

**Q.no.2:**

Apply Depth first search:

Implement graph using adjacency list with int datatype as key values.

1→2

1→3

3→6

3→5

3→4

4→7

6→8

**Answer:**

```
#include<iostream>
#include<list>
using namespace std;

class Graph {
    int V;
    list<int> *adj;

public:
    Graph(int V);

    void addEdge(int v, int w);

    void DFS(int v);
};

Graph::Graph(int V) {
```



```

    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w) {
    adj[v].push_back(w);
}

void Graph::DFS(int v) {
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    list<int> stack;

    visited[v] = true;
    stack.push_back(v);

    while(!stack.empty()) {

        v = stack.back();
        cout << v << " ";
        stack.pop_back();

        for (auto i = adj[v].begin(); i != adj[v].end(); ++i) {
            if (!visited[*i]) {
                stack.push_back(*i);
                visited[*i] = true;
            }
        }
    }
}

int main() {
    Graph g(9);

    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(3, 6);
    g.addEdge(3, 5);
    g.addEdge(3, 4);
    g.addEdge(4, 7);
    g.addEdge(6, 8);

    cout << "Depth First Traversal (starting from vertex 1): ";

```

```

    g.DFS(1);

    return 0;
}

```

Output:

```

Depth First Traversal (starting from vertex 1): 1 3 4 7 5 6 8 2

```

**Q.no.2(b):**

```

#include<iostream>
using namespace std;

class Graph{
private:
    bool** adjMatrix;
    int numVtx;

public:
    Graph(int numVtx){
        this->numVtx = numVtx;
        adjMatrix = new bool*[numVtx];
        for(int i=0; i<numVtx; i++){
            adjMatrix[i]= new bool[numVtx];
            for(int j=0; j<numVtx; j++){
                adjMatrix[i][j]= false;
            }
        }
        void addEdge(int i , int j){
            adjMatrix[i][j]= true;
            adjMatrix[j][i]= true;
        }
        void deleteEdge(int i , int j){
            adjMatrix[i][j]= false;
            adjMatrix[j][i]= false;
        }

        void toString(){
            for(int i=0; i<numVtx; i++){
                cout<<i<<" : ";
                for(int j=0; j<numVtx; j++){

```


```

        cout<<adjMatrix[i][j]<<" ";
    }
    cout<<endl;
}
}
~Graph(){
    for(int i=0; i<numVtx; i++){
        delete[] adjMatrix[i];
    }
    delete[] adjMatrix;
}
};

int main(){
    Graph g(7);
    g.addEdge(0,2);
    g.addEdge(0,3);
    g.addEdge(0,4);
    g.addEdge(1,4);
    g.addEdge(1,5);
    g.addEdge(1,6);
    g.addEdge(2,0);
    g.addEdge(2,4);
    g.addEdge(3,0);
    g.addEdge(3,5);
    g.addEdge(3,6);
    g.addEdge(4,0);
    g.addEdge(4,1);
    g.addEdge(4,2);
    g.addEdge(4,6);
    g.addEdge(5,1);
    g.addEdge(5,3);
    g.addEdge(5,6);
    g.addEdge(6,1);
    g.addEdge(6,3);
    g.addEdge(6,4);
    g.addEdge(6,5);
    g.toString();
    return 0;
}

```

## Output:

 C:\Users\Student.CL05-L3-PC09\Desktop\M.AbdullahF22607010\q3.exe

```
0 : 0 0 1 1 1 0 0
1 : 0 0 0 0 1 1 1
2 : 1 0 0 0 1 0 0
3 : 1 0 0 0 0 1 1
4 : 1 1 1 0 0 0 1
5 : 0 1 0 1 0 0 1
6 : 0 1 0 1 1 1 0
```

-----