**CIRCULAR QUEUES**

**LAB # 09**



**Data Structures & Algorithms**

Submitted by: **Shah Raza**

Registration No: **18PWCSE1658**

Class Section: **B**

“On my honor, as a student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to: **Dr. Khurram Shehzad Khattak**

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Lab Objectives:**

Objectives of this lab are as follows:

* Implement basic operations on Circular Queues.

**Task # 1:**

Implement Circular Queues using arrays.

**Code:**

#include<iostream>

using namespace std;

struct Queue

{

int rear, front;

int size;

int \*arr;

};

Queue Create(Queue q,int s)

{

q.front = q.rear = -1;

q.size = s;

q.arr = new int[s];

return q;

}

Queue enQueue(Queue q,int value)

{

if ((q.front == 0 && q.rear == q.size-1)

|| (q.rear == (q.front-1)%(q.size-1)))

{

cout<<"\nQueue is Full"<<endl;

}

else if (q.front == -1)

{

q.front = q.rear = 0;

q.arr[q.rear] = value;

}

else if (q.rear == q.size-1 && q.front != 0)

{

q.rear = 0;

q.arr[q.rear] = value;

}

else

{

q.rear++;

q.arr[q.rear] = value;

}

return q;

}

Queue deQueue(Queue q)

{ if (q.front == -1)

{

cout<<"\nQueue is Empty"<<endl;

}

else

{

cout<<"\n"<<q.arr[q.front]<<" Dequeued"<<endl;

q.arr[q.front] = -1;

if (q.front == q.rear)

{

q.front = -1;

q.rear = -1;

}

else if (q.front == q.size-1)

q.front = 0;

else

q.front++;

}

return q;

}

void displayQueue(Queue q)

{cout<<"\nElements in Circular Queue are: "<<endl;

if (q.rear >= q.front)

{

for (int i = q.front; i <= q.rear; i++)

{ cout<<q.arr[i]<<"\t"; }

}

else

{ for (int i = q.front; i < q.size; i++)

{ cout<<q.arr[i]<<"\t"; }

for (int i = 0; i <= q.rear; i++)

{ cout<<q.arr[i]<<"\t"; }

}

}

void isEmpty(Queue q)

{ if (q.front == -1)

{

cout<<"Queue is empty"<<endl;

}

else

{

cout<<"Queue is not empty"<<endl;

}

}

void front(Queue q)

{

cout<<"Front of the queue is: "<<q.arr[q.front]<<endl;

}

void rear(Queue q)

{

cout<<"Rear of the queue is: "<<q.arr[q.rear]<<endl;

}

int main()

{

Queue q;

int choice,item,S;

while (1)

{ cout<<"\n-------------"<<endl;

cout<<"Operations on Circular Queue"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Queue"<<endl;

cout<<"2.Enqueue"<<endl;

cout<<"3.Dequeue"<<endl;

cout<<"4.Is the Queue Empty"<<endl;

cout<<"5.Front"<<endl;

cout<<"6.Rear"<<endl;

cout<<"7.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter the size of the queue: ";

cin>>S;

q= Create(q,S);

break;

case 2:

cout<<"Enter the values to be Enqueued: ";

cin>>item;

q=enQueue(q,item);

displayQueue(q);

break;

case 3:

q=deQueue(q);

break;

case 4:

isEmpty(q);

break;

case 5:

front(q);

break;

case 6:

rear(q);

break;

case 7:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

}

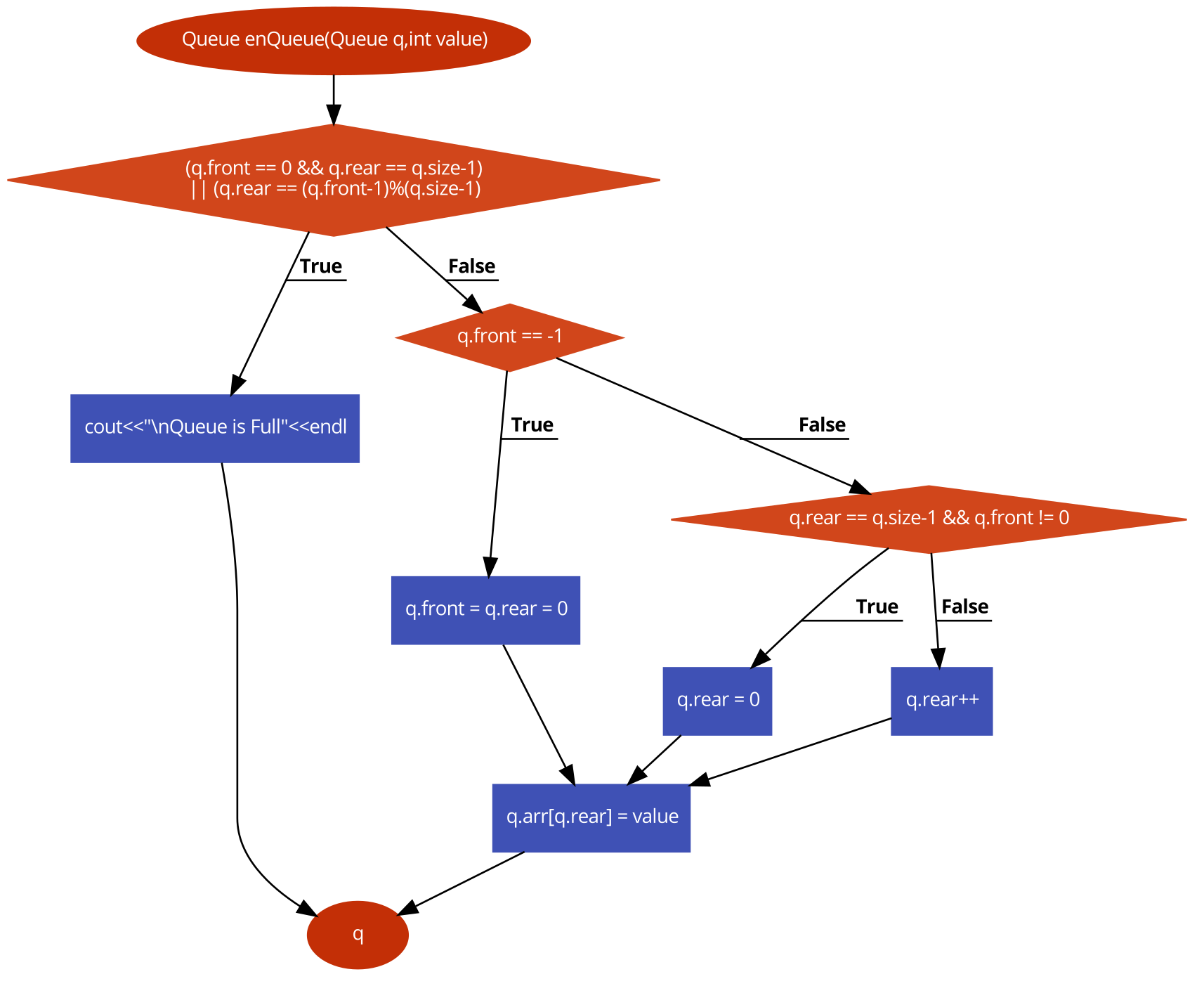
}

return 0;

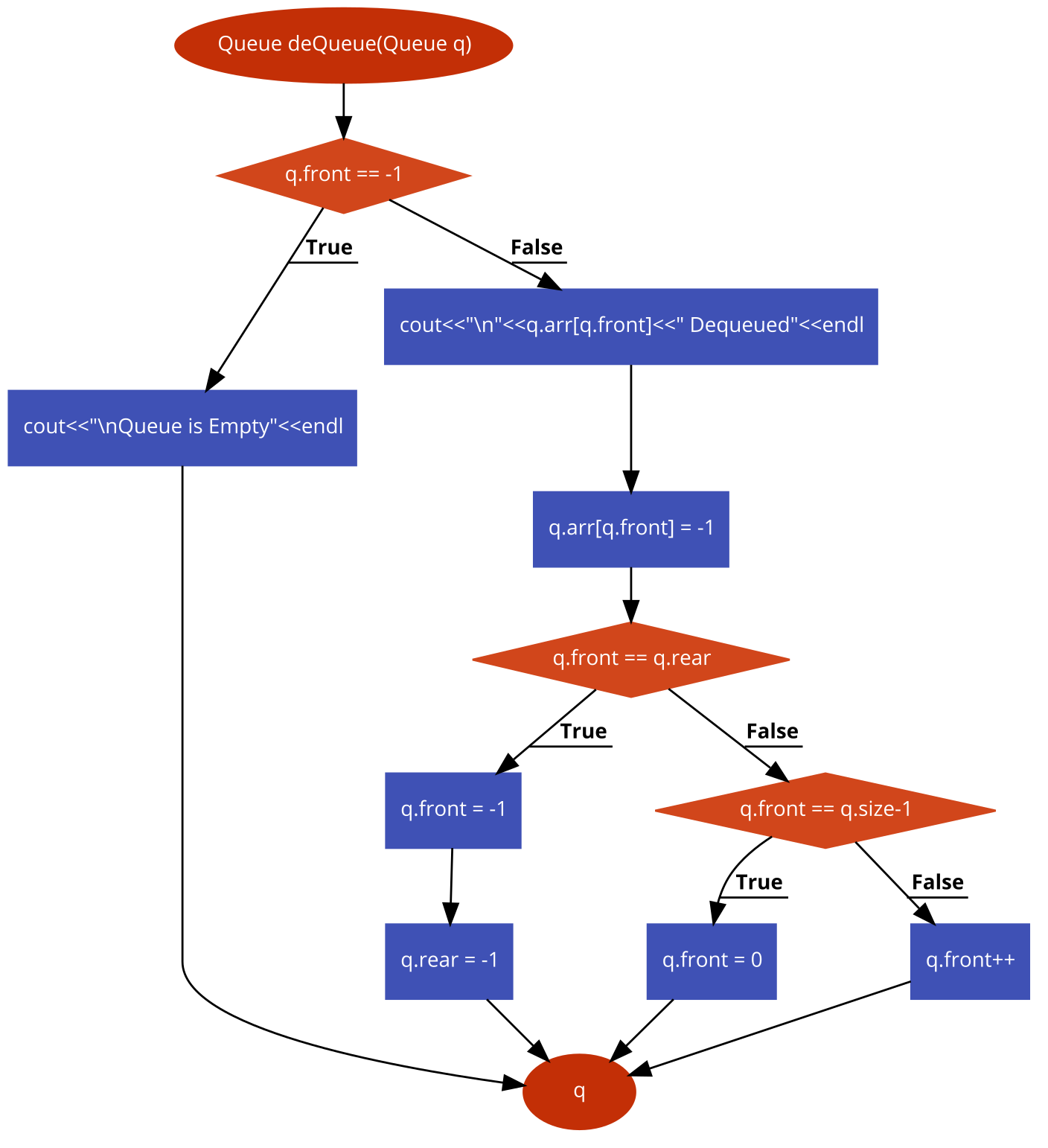
}

**Flow Chart:**

* **Enqueue Function:**

****

* **Dequeue Function:**

****

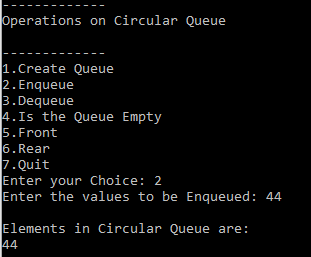
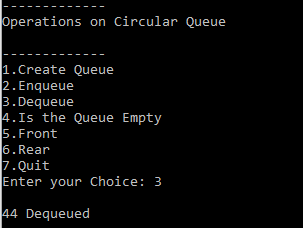
**Pseudo-Code:**

* Enqueue function:
  + if front is equal to 0 And rear is equal to size-1

Or rear is equal to front-1 mod size-1

* + - Then print “Queue is Full”
  + else if front is equal to -1
    - Then front = rear = 0
    - Insert value, array[rear] = value
* else if rear is equal to size-1And front is not equal to 0
* Then rear = 0
* Insert value, array[rear] = value
* else
  + Increment rear
  + Insert value, array[rear] = value
* Return the updated queue
* Dequeue function:
* if front is equal to -1
  + Then print “Queue is Empty”
* else
  + Print “Dequeued”
  + array[front] = -1
  + if front is equal to rear
    - Then front = -1
    - rear = -1
* else if front is equal to size-1
  + Then front = 0;
* else
  + Increment front
* Return the updated queue

**Output:**

** **

**Task # 2:**

Implement Circular Queues using Linked List.

**Code:**

#include <iostream>

using namespace std;

typedef struct Node

{

int data;

Node \*next;

}Node;

class CircularQueue

{

private:

Node \*Front,\*Rear;

int Capacity, SIZE;

public:

void Create(int S)

{

Capacity=S;

SIZE=0;

Front=Rear=NULL;

}

void Enqueue(int value)

{

if(SIZE==Capacity)

cout<<"Queue is Full.\n";

else

{

Node \*temp=new Node;

temp->data=value;

if(Front==NULL)

Front=temp;

else

Rear->next=temp;

Rear=temp;

Rear->next=Front;

cout<<"Element "<<value<<" enqueued.\n";

SIZE++;

}

}

void Dequeue()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

{

int value;

if(Front==Rear)

{

value=Front->data;

free(Front);

Front=Rear=NULL;

}

else

{

Node \*temp=Front;

value=temp->data;

Front=Front->next;

Rear->next=Front;

free(temp);

}

SIZE--;

cout<<"Element "<<value<<" Dequeued.\n";

}

}

void FRONT()

{

if(Front==NULL)

cout<<"Queue is empty.\n";

else

cout<<"Front: "<<Front->data<<endl;

}

void REAR()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

cout<<"Rear: "<<Rear->data<<endl;

}

void Size()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

cout<<"Size of the Queue: "<<SIZE<<endl;

}

void isEmpty()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

cout<<"Queue is Not Empty.\n";

}

void Display()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

{

Node \*temp=Front;

while(temp->next!=Front)

{

cout<<temp->data<<"<--";

temp=temp->next;

}

cout<<temp->data<<endl;

}

}

};

int main()

{

CircularQueue q;

int choice,item,S;

while (1)

{

cout<<"\n-------------"<<endl;

cout<<"Operations on Circular Queue"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Queue"<<endl;

cout<<"2.Enqueue"<<endl;

cout<<"3.Dequeue"<<endl;

cout<<"4.Is the Queue Empty"<<endl;

cout<<"5.Front"<<endl;

cout<<"6.Rear"<<endl;

cout<<"7.Display"<<endl;

cout<<"8.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter the size of the queue: ";

cin>>S;

q.Create(S);

break;

case 2:

cout<<"Enter the values to be Enqueued: ";

cin>>item;

q.Enqueue(item);

break;

case 3:

q.Dequeue();

break;

case 4:

q.isEmpty();

break;

case 5:

q.FRONT();

break;

case 6:

q.REAR();

break;

case 7:

q.Display();

break;

case 8:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

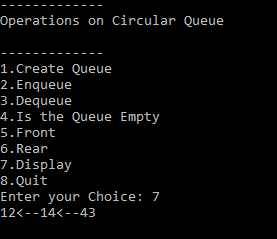
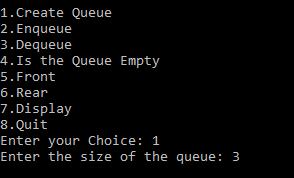
}

}

return 0;

}

**Output:**

****

****

****

****