DSA LAB 2

Queues(Linear,Circular,Dequeue)

Submitted By: Sandeep Acharya(075BCT074)
Submitted To: Department of Electronics and Computer Engineering

Introduction:

Queue is a linear data structure that is similar to stacks. In stack insertion and deletion is done from the same end while in queue insertion is done at end and deletion is done from front. One end is always used to insert data (enqueue) and the other end to remove data (dequeue). It is a FIFO (First In First Out) data structure where the element that is added first will be deleted first.

Operations in a queue:

1. enqueue() – add an item to the queue.
2. dequeue() – remove an item from the queue.
3. empty() – checks if the queue is empty.
4. size() - returns the size of queue
5. front() - returns the front element of queue.

In this lab we implemented three types of Queues:

1. Linear Queue

It's a normal queue where insertion is done at the back and deletion is done from the front. In lab we implemented linear queue using array. In this type of queue, even if we perform dequeue operation memory is not exactly freed and once we insert elements equal to the size of queue we declared. Then we cannot add any element even after performing dequeue operations.

2. Circular Queue

It addreses the drawback of linear queue. This data structured can be visualized as the circular array where we can again return to the front of the array and insert elements there provided that the space is free.

3. Dequeue

It is a double ended Queue where we we can insert and delete from either end of the queue.

Practical Examples of Queue

1. The consumer who comes first to a shop will be served first.
2. In Networking, requests are stored in buffer as queue before they are served.
3. CPU task scheduling and disk scheduling.
4. Waiting list of tickets in case of bus and train tickets.

Algorithms:

Assumptions:

Queue is implemented as a class that has ARR as the array to implement queue, FRONT and REAR index as pointers to keep track of where we are in a queue. ELEMENT is the value passed from outside by the user. We assume initally both pointer are -1 to show empty state. SIZE is the max size of ARR.

Linear Queue:

Enqueue:

Step 1: Check if the queue is full. i.e (REAR==SIZE-1)
Step 2: If the queue is full, then print "Can't Enqueue, Queue overflow" and return.
Step 3: REAR.
Step 4: Assign ARR[REAR] = ELEMENT.

Dequeue:

Step 1: Check if the queue is empty.(FRONT==-1 or FRONT>REAR)
Step 2: If the queue is empty, the print "Can't Dequeue. Queue underflow" and return.
Step 3: Copy the element at the front of the queue to some temporary variable, TEMP = ARR[FRONT].
Step 4: FRONT++
Step 5: Return temp or print it there.

Circular Queue:

Enqueue:

Check Full condition. (REAR+1)%SIZE==FRONT
Everything is same as in Linear Queue except step 3.
Step 3: REAR = (REAR+1)%SIZE

Dequeue:

Check Empty condition. (FRONT==-1)
Everything is Same as in Linear Queue except step 4.
Step 4: FRONT = (FRONT+1)%SIZE.
Also check if FRONT==REAR, if true set FRONT=REAR=-1.

Double Ended Queue(Dequeue):

A new variable COUNT is introduced to count the no of elements present in the ARR.
Full Condition Check: COUNT == SIZE
Empty Condition Check: COUNT = 0

Operations Performed:

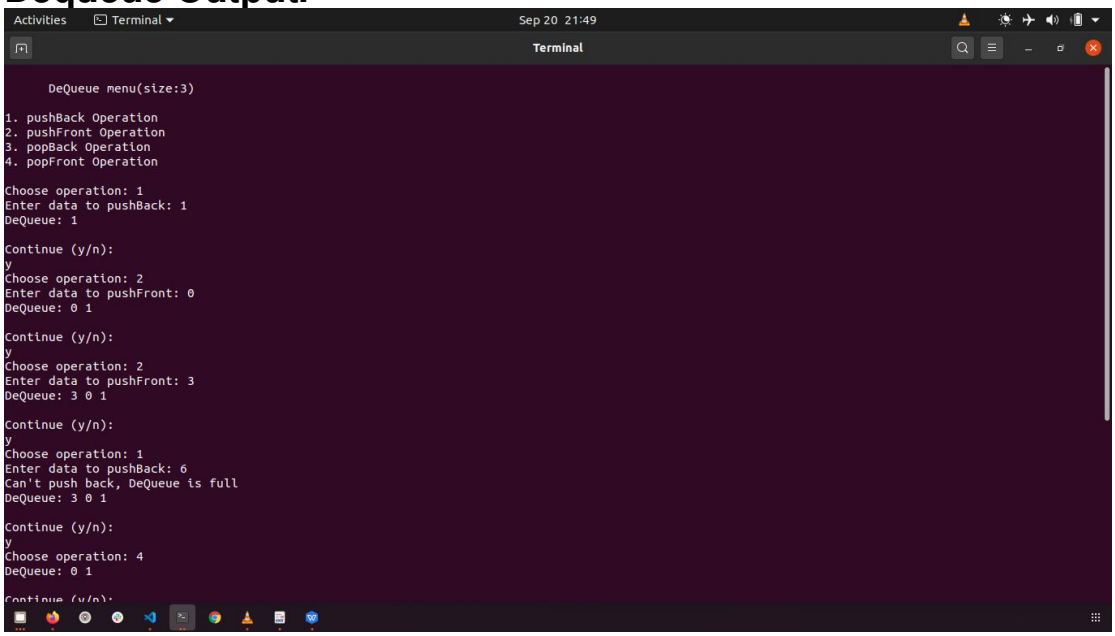
PushBack:
Step 1: Check Full condition.
Step 2: If true, print “Can’t push back, Dequeue is full” and return.
Step 3: REAR=(REAR+1)%SIZE
Step 4: ARR[REAR]=ELEMENT
Step 5: COUNT++

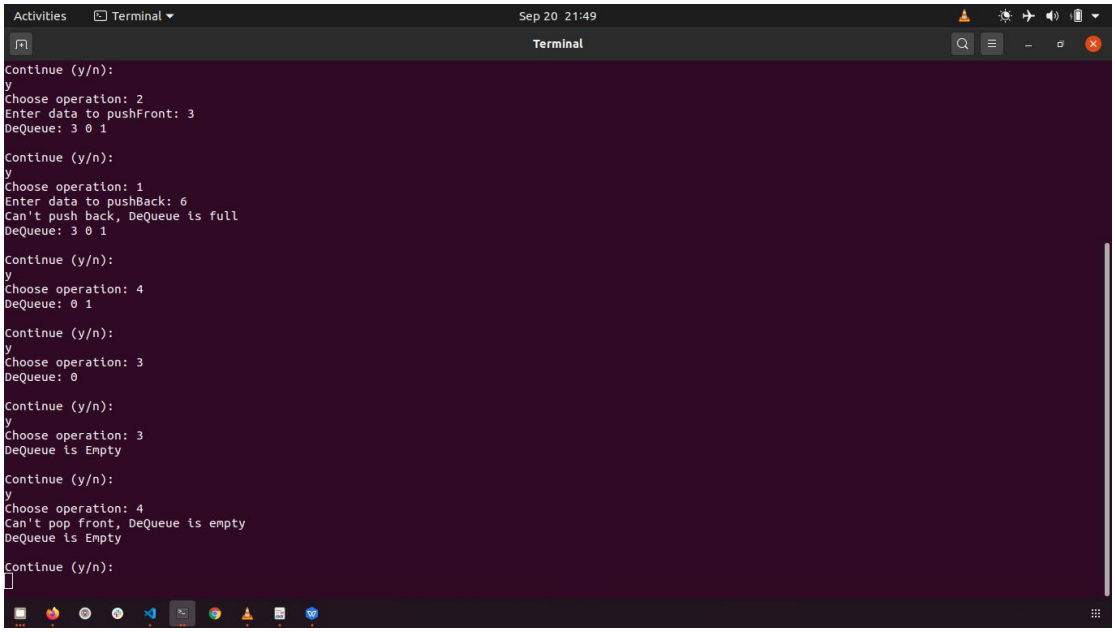
PopBack:
Step 1: Check Empty Codition.
Step 2: if true, print “Can’t pop back, Dequeue is empty” and return.
Step 3: TEMP=ARR[REAR]
Step 4: if REAR==0 then REAR=SIZE-1
Step 5: else REAR--
Step 6: COUNT--
Step 7: return TEMP

PushFront:
Step 1: Check Full Condition.
Step 2: if true, print “Can’t push front, Dequeue is full” and return.
Step 3: if FRONT==0 then FRONT=SIZE-1
Step 4: else FRONT--
Step 5: ARR[FRONT]=ELEMENT
Step 6: COUNT++

PopFront:
Step 1: Check Empty Condition.
Step 2: if true, print “Can’t pop front, Dequeue is empty” and return.
Step 3: TEMP=ARR[FRONT]
Step 4: FRONT=(FRONT+1)%SIZE
Step 5: COUNT--
Step 6: return TEMP

Dequeue Output:





Discussion and Conclusion:

In this lab, We learnt about the queue data structure and it's different types.Once Linear queue becomes full, we cannot insert the next element until all the elements are deleted from the queue. Therefore, to remove this limitation, we implemented circular queue. We implemented queue using array thus having a limitation of defining the size in the program itself. We can overcome this limitation using linked list, where queue size can be dynamically increased or decreased.