Assignment – 31 (Double ended queue) – Write-up

> This is pre-production content, posted in rush due to high demand from R-batch. **Does not have flowcharts.**

Algorithm

Class Dequeue

Data members:
- a: an integer array of size 10
- front: an integer to track the front index, initialized to -1
- rear: an integer to track the rear index, initialized to -1
- count: an integer to track the number of elements in the Dequeue, initialized to 0

Constructor:
- Initialize front, rear, and count to -1.

Method addBegin(item):
- If front is -1:
  - Set front and rear to 0.
  - Assign item to a[rear].
  - Increment count by 1.
- Else if rear is greater than or equal to SIZE - 1:
  - Print "Insertion is not possible, overflow!!!"
- Else:
  - Shift elements in the array a to make space for the new item.
  - Insert item at the front of the Dequeue.
  - Increment count by 1.
  - Increment rear by 1.

Method addEnd(item):
- If front is -1:
  - Set front and rear to 0.
  - Assign item to a[rear].
  - Increment count by 1.
- Else if rear is greater than or equal to SIZE - 1:
  - Print "Insertion is not possible, overflow!!!"
- Else:
  - Increment rear by 1.
  - Assign item to a[rear].
  - Increment count by 1.

Method deleteFront():
- If front is -1:
  - Print "Deletion is not possible: Dequeue is empty."
- Else:
  - If front is equal to rear:
    - Set front and rear to -1.

- Print "The deleted element is " followed by a[front].
- Increment front by 1.

Method deleteEnd():
- If front is -1:
  - Print "Deletion is not possible: Dequeue is empty."
- Else:
  - If front is equal to rear:
    - Set front and rear to -1.
  - Print "The deleted element is " followed by a[rear].
  - Decrement rear by 1.

Method display():
- Loop from i = front to i <= rear:
- Print a[i] followed by a space.

Function main():

Declare variables: c, item as integers
Create an instance of the Dequeue class and name it d1

Repeat the following loop until c is equal to 6:
Display a menu for Dequeue operations:
- "****DEQUEUE OPERATION****"
- "1-Insert at beginning"
- "2-Insert at end"
- "3-Display"
- "4-Deletion from front"
- "5-Deletion from rear"
- "6-Exit"
- "Enter your choice (1-6):"

Read the user's choice into variable c

Switch on the value of c:
Case 1:
Prompt the user to enter an element
Read the element into variable item
Call the addBegin(item) method on d1 (Dequeue object)
Break
Case 2:
Prompt the user to enter an element
Read the element into variable item
Call the addEnd(item) method on d1
Break
Case 3:
Call the display() method on d1 to show the elements in the Dequeue
Break
Case 4:
Call the deleteFront() method on d1 to remove an element from the front
Break

```
        Case 5:
          Call the deleteEnd() method on d1 to remove an element from the rear
          Break
        Case 6:
          Exit the program
          Break
        Default:
          Display "Invalid choice"
          Break

    End of loop

    Return 0 to indicate successful program completion
End of Function main()
```

Pseudocodes

class Dequeue

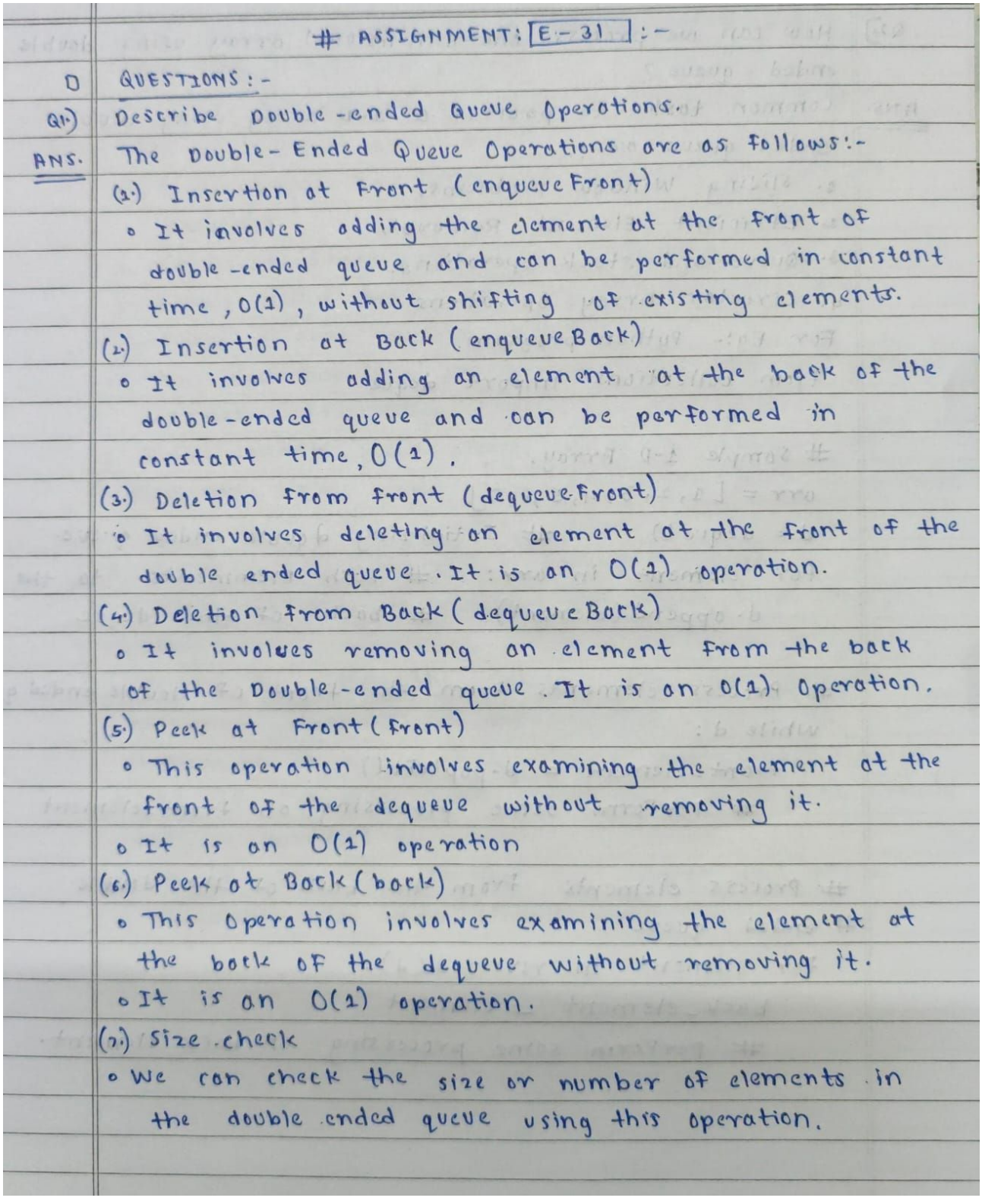
```
{
    int a[10], front, rear, count


    Dequeue()
    {
        front = -1
        rear = -1
        count = 0
    }


    addBegin(item)
    {
        if front == -1
            front++
            rear++
            a[rear] = item
            count++
        else if rear >= SIZE - 1
            print "Insertion is not possible, overflow!!!!"
        else
            for i from count down to 0
                a[i + 1] = a[i]
            a[front] = item
            count++
            rear++
    }


    addEnd(item)
    {
        if front == -1
            front++
```

```
            rear++
            a[rear] = item
            count++
        else if rear >= SIZE - 1
            print "Insertion is not possible, overflow!!!"
        else
            rear++
            a[rear] = item
            count++
    }


    deleteFront()
    {
        if front == -1
            print "Deletion is not possible: Dequeue is empty"
        else
            print "The deleted element is " + a[front]
            if front == rear
                front = -1
                rear = -1
            else
                front++
    }


    deleteEnd()
    {
        if front == -1
            print "Deletion is not possible: Dequeue is empty"
        else
            print "The deleted element is " + a[rear]
            if front == rear
                front = -1
                rear = -1
            else
                rear--
    }


    display()
    {
        for i from front to rear
            print a[i] + " "
    }
}
```

```
function main()
{
   integer c, item
   Dequeue d1


   repeat
   {
      print("\n\n****DEQUEUE OPERATION****")
      print("1-Insert at beginning")
      print("2-Insert at end")
      print("3-Display")
      print("4-Deletion from front")
      print("5-Deletion from rear")
      print("6-Exit")
      print("Enter your choice <1-6>:")
      input(c)


      switch (c)
      {
         case 1:
            print("Enter the element to be inserted:")
            input(item)
            d1.addBegin(item)
            break


         case 2:
            print("Enter the element to be inserted:")
            input(item)
            d1.addEnd(item)
            break


         case 3:
            d1.display()
            break


         case 4:
            d1.deleteFront()
            break


         case 5:
            d1.deleteEnd()
            break


         case 6:
            exit(1)
            break
```

```
            default:
                print("Invalid choice")
                break
        }
    } until (c == 6)
}
```

Answers



ASSIGNMENT: E-31 :-

D

QUESTIONS :-

Q1.) Describe Double-ended Queue Operations.

ANS. The Double-Ended Queue Operations are as follows:-

(1.) Insertion at Front (enqueue Front)

- It involves adding the element at the front of double-ended queue and can be performed in constant time, O(1), without shifting of existing elements.

(2.) Insertion at Back (enqueue Back)

- It involves adding an element at the back of the double-ended queue and can be performed in constant time, O(1).

(3.) Deletion from front (dequeue Front)

- It involves deleting an element at the front of the double ended queue. It is an O(1) operation.

(4.) Deletion from Back (dequeue Back)

- It involves removing an element from the back of the Double-ended queue. It is an O(1) Operation.

(5.) Peek at Front (front)

- This operation involves examining the element at the front of the dequeue without removing it.
- It is an O(1) operation

(6.) Peek at Back (back)

- This Operation involves examining the element at the back of the dequeue without removing it.
- It is an O(1) operation.

(7.) Size check

- We can check the size or number of elements in the double ended queue using this operation.

2] How can we process one-dimensional array using double ended queue?

Ans. Common tasks to process a 1-D array using double ended queue are :-

1. Sliding Window problems.
2. Efficient Element Removal.
3. Queue and stack operations.
4. Circular Array Operations.

For Eg:- Python program.

```
from collections import deque

# Sample 1-D Array.
arr = [1, 2, 3, 4, 5, 6, 7]
d = deque()  # Intializing double ended queue.
for element in arr:  # Push Elemments to the
    d.append(element)  # back of the deque.

# Process elements from the front of double ended queue
while d:
    front_element = d.popleft()
    # Perform some processing on front_element

# Process elements from the back of the double
# ended queue
for element in reversed(d):
    back_element = element
    # Perform some processing on back element.
```

Q3. What are the Advantages of Double-Ended Queue over single/simple Queue?

ANS. The Key Advantages of using a Double ended queue over a simple queue are:-

1. Bidirectional Insertion and Deletion (Removal) of elements.
2. Efficient Stack and Queue Operations
3. Double ended Queues are particularly used for solving sliding window problems.
4. They can simulate circular array operations, such as rotation efficiently.
5. Double-ended queue allows efficient removal of elements from both front and back ends in constont time
6. Useful in implementing Algorithms like Breadth-First search (BFS) and Depth-First search (DFS)
7. Useful to efficiently maintain the minimum or maximum elements in the window, which is challenging to do with a simple queue.
8. It is Dynamic Data structure allowing to handle varying workloads.
9. It provides simpler i.e. cleaner and more readable code than simple queue.