Rajalakshmi Engineering College

Name: KEERTHI PRIYA T
Email: 240701258@rajalakshmi.edu.in
Roll no: 2116240701258
Phone: 7397397221
Branch: REC
Department: I CSE FC
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are tasked with implementing basic operations on a queue data structure using a linked list.

You need to write a program that performs the following operations on a queue:

Enqueue Operation: Implement a function that inserts an integer element at the rear end of the queue.Print Front and Rear: Implement a function that prints the front and rear elements of the queue. Dequeue Operation: Implement a function that removes the front element from the queue.

Input Format

The first line of input consists of an integer N, representing the number of elements to be inserted into the queue.

The second line consists of N space-separated integers, representing the queue elements.

Output Format

The first line prints "Front: X, Rear: Y" where X is the front and Y is the rear elements of the queue.

The second line prints the message indicating that the dequeue operation (front element removed) is performed: "Performing Dequeue Operation:".

The last line prints "Front: M, Rear: N" where M is the front and N is the rear elements after the dequeue operation.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
12 56 87 23 45

Output: Front: 12, Rear: 45
Performing Dequeue Operation:
Front: 56, Rear: 45

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* front = NULL;
struct Node* rear = NULL;
```

```
# You are using Python
class Node:
    def __init__(self, data):
        self.data = data
```

```
        self.next = None

class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

    def enqueue(self, data):
        new_node = Node(data)
        if self.rear is None:  # Queue is empty
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node

    def dequeue(self):
        if self.front is None:  # Queue is empty
            return
        temp = self.front
        self.front = self.front.next
        if self.front is None:  # Queue became empty
            self.rear = None
        del temp

    def get_front(self):
        return self.front.data if self.front else None

    def get_rear(self):
        return self.rear.data if self.rear else None

# Input reading
N = int(input())
elements = list(map(int, input().split()))

q = Queue()

# Enqueue elements
for num in elements:
    q.enqueue(num)

# Print front and rear before dequeue
print(f"Front: {q.get_front()}, Rear: {q.get_rear()}")
```

```
# Perform Dequeue
print("Performing Dequeue Operation:")
q.dequeue()

# Print front and rear after dequeue
print(f"Front: {q.get_front()}, Rear: {q.get_rear()}")
```

```
int main() {
    int n, data;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        enqueue(data);
    }
    printFrontRear();
    printf("Performing Dequeue Operation:\n");
    dequeue();
    printFrontRear();
    return 0;
}
```

Status : Correct **Marks : 10/10**