Aim: Implement Linear Queue ADT using Linked List.

Theory:

Queue is a linear data structure that follows the First-In-First-Out (FIFO) order of operations. This means the first element added to the queue will be the first one to be removed. There are different ways using which we can implement a queue data structure in C.

In this experiment, we will learn how to implement a queue using a linked list in C, its basic operations along with their time and space complexity analysis, and the benefits of a linked list queue in C.

Procedure:

we maintain two pointers, front, and rear. The front points to the first item of the queue and rear points to the last item.

enQueue(): This operation adds a new node after the rear and moves the rear to the next node.

deQueue(): This operation removes the front node and moves the front to the next node.

Steps:-

- Create a class QNode with data members integer data and QNode* next
    - A parameterized constructor that takes an integer x value as a parameter and sets data equal to x and next as NULL
- Create a class Queue with data members QNode front and rear
- Enqueue Operation with parameter x:
    - Initialize QNode* temp with data = x
    - If the rear is set to NULL then set the front and rear to temp and return(Base Case)
    - Else set rear next to temp and then move rear to temp
- Dequeue Operation:
    - If the front is set to NULL return(Base Case)
    - Initialize QNode temp with front and set front to its next

- If the front is equal to NULL then set the rear to NULL
- Delete temp from the memory

Code:

```
// C program to implement the queue data structure using
 // linked list
 #include <limits.h>
 #include <stdio.h>
 #include <stdlib.h>
 // Node structure representing a single node in the linked
 // list
 typedef struct Node {
 int data;
 struct Node* next;
 } Node;
 // Function to create a new node
 Node* createNode(int new_data)
 {
 Node* new_node = (Node*)malloc(sizeof(Node));
 new_node->data = new_data;
 new_node->next = NULL;
 return new_node;
 }
 // Structure to implement queue operations using a linked
 // list
 typedef struct Queue {
 // Pointer to the front and the rear of the linked list
```

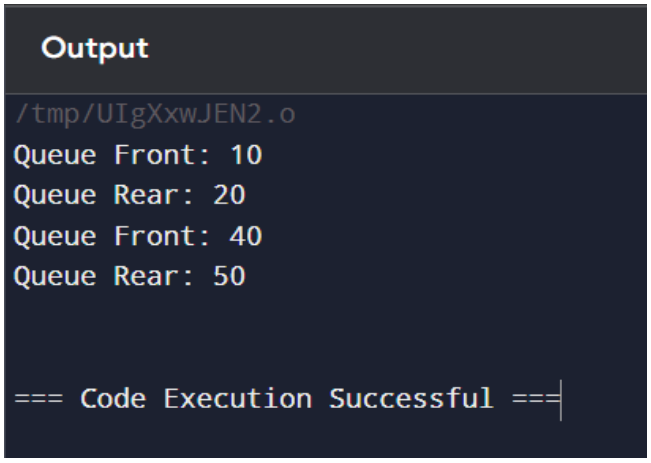
```
Node *front, *rear;
} Queue;
// Function to create a queue
Queue* createQueue()
{
Queue* q = (Queue*)malloc(sizeof(Queue));
q->front = q->rear = NULL;
return q;
}
// Function to check if the queue is empty
int isEmpty(Queue* q)
{
// If the front and rear are null, then the queue is
// empty, otherwise it's not
if (q->front == NULL && q->rear == NULL) {
return 1;
}
return 0;
}
// Function to add an element to the queue
void enqueue(Queue* q, int new_data)
{
// Create a new linked list node
Node* new_node = createNode(new_data);
// If queue is empty, the new node is both the front
// and rear
if (q->rear == NULL) {
q->front = q->rear = new_node;
return;
```

```
}
// Add the new node at the end of the queue and
// change rear
q->rear->next = new_node;
q->rear = new_node;
}
// Function to remove an element from the queue
void dequeue(Queue* q)
{
// If queue is empty, return
if (isEmpty(q)) {
printf("Queue Underflow\n");
return;
}
// Store previous front and move front one node
// ahead
Node* temp = q->front;
q->front = q->front->next;
// If front becomes null, then change rear also
// to null
if (q->front == NULL)
q->rear = NULL;
// Deallocate memory of the old front node
free(temp);
}
// Function to get the front element of the queue
int getFront(Queue* q)
{
// Checking if the queue is empty
```

```
if (isEmpty(q)) {
printf("Queue is empty\n");
return INT_MIN;
}
return q->front->data;
}
// Function to get the rear element of the queue
int getRear(Queue* q)
{
// Checking if the queue is empty
if (isEmpty(q)) {
printf("Queue is empty\n");
return INT_MIN;
}
return q->rear->data;
}
// Driver code
int main()
{
Queue* q = createQueue();
// Enqueue elements into the queue
enqueue(q, 10);
enqueue(q, 20);
printf("Queue Front: %d\n", getFront(q));
printf("Queue Rear: %d\n", getRear(q));
// Dequeue elements from the queue
dequeue(q);
dequeue(q);
// Enqueue more elements into the queue
```

```
enqueue(q, 30);
 enqueue(q, 40);
 enqueue(q, 50);
 // Dequeue an element from the queue
 dequeue(q);
 printf("Queue Front: %d\n", getFront(q));
 printf("Queue Rear: %d\n", getRear(q));
 return 0;
 }
```

Output:



Conclusion: This experiment shows how to implement a Stack ADT using a linked list, an alternative to the array-based implementation. It emphasizes the use of linked lists to create a dynamic stack that can grow as needed, providing flexibility and avoiding fixed-size limitations.