Experiments\queue_linkedlist.c

```c
// 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;
```

```c
}

// 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:PS C:\Users\gagan\Documents\Data_Structure_And_Algorithm> cd "c:\Users\gagan\Documents\Data_Structure_And_Algorithm\Experiments\" ; if ($?) { gcc stack_linkedlist.c -o stack_linkedlist } ; if ($?) { .\stack_linkedlist }
Queue Front: 10
Queue Rear: 20
Queue Front: 40
Queue Rear: 50
*/
```