Write C++ programs to implement the dequeue (double ended queue) ADT using a doubly linked list.

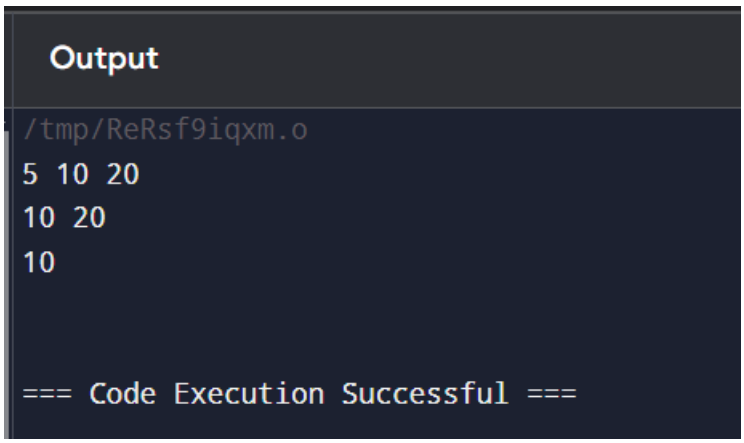
Program:

```
#include <iostream>
using namespace std;

class Node {
public:
   int data;
   Node* next;
   Node* prev;
   Node(int val) : data(val), next(nullptr), prev(nullptr) {}
};

class Deque {
private:
   Node* front;
   Node* rear;

public:
   Deque() : front(nullptr), rear(nullptr) {}

   // Insert at the front
   void insertFront(int val) {
      Node* newNode = new Node(val);
      if (!front) front = rear = newNode;
      else {
         newNode->next = front;
         front->prev = newNode;
```

```
        front = newNode;
    }
}

// Insert at the rear
void insertRear(int val) {
    Node* newNode = new Node(val);
    if (!rear) front = rear = newNode;
    else {
        newNode->prev = rear;
        rear->next = newNode;
        rear = newNode;
    }
}

// Delete from the front
void deleteFront() {
    if (!front) return;
    Node* temp = front;
    if (front == rear) front = rear = nullptr;
    else {
        front = front->next;
        front->prev = nullptr;
    }
    delete temp;
}

// Delete from the rear
void deleteRear() {
    if (!rear) return;
    Node* temp = rear;
```

```
        if (front == rear) front = rear = nullptr;
        else {
            rear = rear->prev;
            rear->next = nullptr;
        }
        delete temp;
    }

    // Display the deque
    void display() {
        Node* temp = front;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Deque dq;
    dq.insertFront(10);
    dq.insertRear(20);
    dq.insertFront(5);
    dq.display();  // Output: 5 10 20
    dq.deleteFront();
    dq.display();  // Output: 10 20
    dq.deleteRear();
    dq.display();  // Output: 10
    return 0;
}
```

Output:



Explanation:

What is a Deque?

A **deque** (double-ended queue) is a data structure that allows insertions and deletions at both the front and rear ends. In this code, the deque is implemented using a doubly linked list, where each node points to both its previous and next node.

Key Parts of the Code

1. Node Class

The Node class represents an individual element (node) in the doubly linked list.

```
class Node {
public:
    int data;
    Node* next;
    Node* prev;
    Node(int val) : data(val), next(nullptr), prev(nullptr) {}
};
```

- **data**: Stores the value of the node.
- **next**: Pointer to the next node in the deque.
- **prev**: Pointer to the previous node in the deque.
- The constructor initializes a node with the given value (val) and sets the next and prev pointers to nullptr.

2. Deque Class

The Deque class is where the operations like insertion, deletion, and display of elements are performed.

Private Members:

private:

```
Node* front;
Node* rear;
```

- **front**: Points to the first node of the deque.
- **rear**: Points to the last node of the deque.

Public Methods:

public:

```
Deque() : front(nullptr), rear(nullptr) {}
```

- **Constructor**: Initializes the deque with both front and rear pointers set to nullptr, meaning the deque is empty at first.

Operations in Deque

1. Insert at the Front

This function inserts a new node at the front of the deque.

```
void insertFront(int val) {
    Node* newNode = new Node(val);
    if (!front) front = rear = newNode;
    else {
        newNode->next = front;
        front->prev = newNode;
        front = newNode;
    }
}
```

- A new node (newNode) is created with the value val.
- If the deque is empty (!front), both front and rear point to the new node.

- Otherwise, the next pointer of the new node points to the current front, and the prev pointer of the current front points to the new node. Finally, the front pointer is updated to point to the new node.

2. Insert at the Rear

This function inserts a new node at the rear of the deque.

```
void insertRear(int val) {
    Node* newNode = new Node(val);
    if (!rear) front = rear = newNode;
    else {
        newNode->prev = rear;
        rear->next = newNode;
        rear = newNode;
    }
}
```

- A new node (newNode) is created with the value val.
- If the deque is empty (!rear), both front and rear point to the new node.
- Otherwise, the prev pointer of the new node points to the current rear, and the next pointer of the current rear points to the new node. Finally, the rear pointer is updated to point to the new node.

3. Delete from the Front

This function deletes a node from the front of the deque.

```
void deleteFront() {
    if (!front) return;  // If deque is empty, do nothing
    Node* temp = front;
    if (front == rear) front = rear = nullptr;  // If only one element
    else {
        front = front->next;
        front->prev = nullptr;
    }
    delete temp;
}
```

- If the deque is empty (!front), nothing happens.
- Otherwise, a temporary pointer (temp) is used to store the current front node.
- If the deque has only one node (front == rear), both front and rear are set to nullptr.
- If the deque has more than one node, the front pointer is moved to the next node, and the prev pointer of the new front node is set to nullptr. The old front node is then deleted.

4. Delete from the Rear

This function deletes a node from the rear of the deque.

```
void deleteRear() {
   if (!rear) return;  // If deque is empty, do nothing
   Node* temp = rear;
   if (front == rear) front = rear = nullptr;  // If only one element
   else {
      rear = rear->prev;
      rear->next = nullptr;
   }
   delete temp;
}
```

- If the deque is empty (!rear), nothing happens.
- Otherwise, a temporary pointer (temp) is used to store the current rear node.
- If the deque has only one node (front == rear), both front and rear are set to nullptr.
- If the deque has more than one node, the rear pointer is moved to the previous node, and the next pointer of the new rear node is set to nullptr. The old rear node is then deleted.

5. Display the Deque

This function prints the elements of the deque from the front to the rear.

```
void display() {
   Node* temp = front;
   while (temp) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}
```

- A temporary pointer (temp) starts at the front node and traverses through the deque, printing the data of each node until it reaches the end (temp == nullptr).

Main Function

```
int main() {
   Deque dq;
   dq.insertFront(10);
   dq.insertRear(20);
   dq.insertFront(5);
   dq.display();  // Output: 5 10 20
   dq.deleteFront();
   dq.display();  // Output: 10 20
   dq.deleteRear();
   dq.display();  // Output: 10
   return 0;
}
```

1. **Deque dq;**: Creates a deque object.
2. **dq.insertFront(10);**: Inserts 10 at the front. Deque becomes: 10.
3. **dq.insertRear(20);**: Inserts 20 at the rear. Deque becomes: 10 20.
4. **dq.insertFront(5);**: Inserts 5 at the front. Deque becomes: 5 10 20.
5. **dq.display();**: Displays the deque, output: 5 10 20.
6. **dq.deleteFront();**: Deletes the front element (5). Deque becomes: 10 20.
7. **dq.display();**: Displays the deque, output: 10 20.
8. **dq.deleteRear();**: Deletes the rear element (20). Deque becomes: 10.
9. **dq.display();**: Displays the deque, output: 10.