Q DOUBLY LINKED LIST

```
#include <iostream>

using namespace std;

class node {
public:
    int data;
    node* prev;
    node* next;

    node(int val) {
        data = val;
        prev = NULL;
        next = NULL;
    }
};

class dll {
public:
    node* head = NULL;

    void insert_AT_Head(int d) {
        node* n1 = new node(d);
        if (head == NULL) {
            head = n1;
        } else {
            n1->next = head;
            head->prev = n1;
            head = n1;
        }
    }

    void insert_AT_loc(int position, int d) {
        if (position == 1) {
            insert_AT_Head(d);
            return;
        }
        node* temp = head;
        int count = 1;
        while (count < position - 1) {
            temp = temp->next;
            count++;
        }

        if (temp->next == NULL) {
            insert_AT_End(d);
            return;
        }
        node* n1 = new node(d);
        n1->next = temp->next;
        temp->next->prev = n1;
        temp->next = n1;
        n1->prev = temp;
    }

    void insert_AT_End(int d) {
        node* n1 = new node(d);
        node* temp = head;
        if (head == NULL) {
            insert_AT_Head(d);
            return;
        }
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = n1;
        n1->prev = temp;
    }

    void delete_AT_head() {
        if (head == NULL) {
            cout << "List is empty, nothing to delete." << endl;
            return;
        }

        node* temp = head;
        temp->next->prev = NULL;
        head = temp->next;
        temp->next = NULL;
        delete temp;
    }

    void delete_AT_loc(int position) {
        if (head == NULL) {
            cout << "List is empty, nothing to delete." << endl;
            return;
        }

        node* curr = head;
        node* temp = NULL;
        int count = 1;
        while (count < position) {
            temp = curr;
            curr = curr->next;
            count++;
```

```cpp
        }

        if (curr == NULL) {
            cout << "Invalid position, nothing to delete." << endl;
            return;
        }

        curr->prev = NULL;
        temp->next = curr->next;
        if (curr->next != NULL) {
            curr->next->prev = temp;
        }
        curr->next = NULL;
        delete curr;
    }

    void display() {
        node* temp = head;
        while (temp != NULL) {
            cout << temp->data << "->";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};

int main() {
    dll d1;
    int choice, data, position;

    do {
        cout << "\n1. Insert at head";
        cout << "\n2. Insert at tail";
        cout << "\n3. Insert at any position";
        cout << "\n4. Delete at head";
        cout << "\n5. Delete at any position";
        cout << "\n6. Display";
        cout << "\n0. Exit";
        cout << "\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter data: ";
                cin >> data;
                d1.insert_AT_Head(data);
                break;
            case 2:
                cout << "Enter data: ";
                cin >> data;
                d1.insert_AT_End(data);
                break;
            case 3:
                cout << "Enter data: ";
                cin >> data;
                cout << "Enter position: ";
                cin >> position;
                d1.insert_AT_loc(position, data);
                break;
            case 4:
                d1.delete_AT_head();
                break;
            case 5:
                cout << "Enter position: ";
                cin >> position;
                d1.delete_AT_loc(position);
                break;
            case 6:
                cout << "Doubly Linked List: ";
                d1.display();
                break;
            case 0:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please enter a valid option." <<
endl;
        }
    } while (choice != 0);

    return 0;
}
```

OUTPUT ➔

1. Insert at head

2. Insert at tail

3. Insert at any position

4. Delete at head

5. Delete at any position

```
6. Display
0. Exit
Enter your choice: 1
Enter data: 9

1. Insert at head
2. Insert at tail
3. Insert at any position
4. Delete at head
5. Delete at any position
6. Display
0. Exit
Enter your choice: 2
Enter data: 8

1. Insert at head
2. Insert at tail
3. Insert at any position
4. Delete at head
5. Delete at any position
6. Display
0. Exit
Enter your choice: 2
Enter data: 8

1. Insert at head
```

```
2. Insert at tail

3. Insert at any position

4. Delete at head

5. Delete at any position

6. Display

0. Exit

Enter your choice: 2

Enter data: 7


1. Insert at head

2. Insert at tail

3. Insert at any position

4. Delete at head

5. Delete at any position

6. Display

0. Exit

Enter your choice: 3

Enter data: 4

Enter position: 4


1. Insert at head

2. Insert at tail

3. Insert at any position

4. Delete at head

5. Delete at any position

6. Display
```

```
0. Exit

Enter your choice: 6

Doubly Linked List: 9->8->8->4->7->NULL
```