DOUBLY LINKED LIST CODE IMPLEMENTATION

```
#include <iostream>

using namespace std;

// Node class definition
class Node {
public:
    int data;
    Node* next;
    Node* prev;

    Node(int data) : data(data), next(nullptr), prev(nullptr) {}
};

// DoublyLinkedList class definition
class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() : head(nullptr) {}

    // Function to create a new node
    Node* createNode(int data) {
        return new Node(data);
```

```
}

// Function to insert a node at the start
void insertNodeAtStart(int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

// Function to insert a node at the end
void insertNodeAtEnd(int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
```

```
}

// Function to insert a node at any position
void insertNodeAtAny(int data, int position) {
    if (position == 0) {
        insertNodeAtStart(data);
        return;
    }

    Node* newNode = createNode(data);
    Node* temp = head;
    for (int i = 0; i < position - 1 && temp != nullptr; ++i) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Position out of bounds" << endl;
        delete newNode;
        return;
    }

    newNode->next = temp->next;
    if (temp->next != nullptr) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    newNode->prev = temp;
```

```
}

// Function to delete a node at the start
void deleteNodeAtStart() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    }
    delete temp;
}

// Function to delete a node at the end
void deleteNodeAtEnd() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

    Node* temp = head;
    if (temp->next == nullptr) {
        head = nullptr;
```

```
        delete temp;
        return;
    }

    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->prev->next = nullptr;
    delete temp;
}

// Function to delete a node at any position
void deleteNodeAtAny(int position) {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

    if (position == 0) {
        deleteNodeAtStart();
        return;
    }

    Node* temp = head;
    for (int i = 0; i < position && temp != nullptr; ++i) {
        temp = temp->next;
    }
```

```
    if (temp == nullptr) {
        cout << "Position out of bounds" << endl;
        return;
    }

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

// Function to traverse and print the list
void traversal() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Function to search for a node with a specific value
bool search(int data) {
```

```
        Node* temp = head;
        while (temp != nullptr) {
            if (temp->data == data) {
                return true;
            }
            temp = temp->next;
        }
        return false;
    }
};

// Main function to test the doubly linked list
int main() {
    DoublyLinkedList dll;

    dll.insertNodeAtStart(10);
    dll.insertNodeAtEnd(20);
    dll.insertNodeAtEnd(30);
    dll.insertNodeAtAny(25, 2);

    cout << "List after insertion: ";
    dll.traversal();

    dll.deleteNodeAtStart();
    cout << "List after deleting node at start: ";
    dll.traversal();
```

```
    dll.deleteNodeAtEnd();
    cout << "List after deleting node at end: ";
    dll.traversal();

    dll.deleteNodeAtAny(1);
    cout << "List after deleting node at position 1: ";
    dll.traversal();

    if (dll.search(20)) {
        cout << "20 is in the list" << endl;
    } else {
        cout << "20 is not in the list" << endl;
    }

    return 0;
}
```

OUTPUT