## **LEC-4 Doubly LinkList**

using namespace std;

#include <iostream>

class doublylinklist; // Forward declaration

class Node {

    friend class doublylinklist;

    int data;

    Node\* next;

    Node\* prev;

};

class doublylinklist {

private:

    Node\* head;

public:

    doublylinklist() {

        head = nullptr;

    }

    ~doublylinklist() {

        // TODO: Implement destructor to free memory used by the list nodes

    }

    bool insertatstart(int val) {

        Node\* newNode = new Node();

        newNode->data = val;

        newNode->next = head;

        newNode->prev = nullptr;

        if (head != nullptr) {

            head->prev = newNode;

        }

        head = newNode;

        return true;

    }

    bool insertatend(int val) {

        Node\* newNode = new Node();

        newNode->data = val;

        newNode->next = nullptr;

        if (head == nullptr) {

            newNode->prev = nullptr;

            head = newNode;

            return true;

        }

        Node\* current = head;

        while (current->next != nullptr) {

            current = current->next;

        }

        current->next = newNode;

        newNode->prev = current;

        return true;

    }

    bool sortedinsert(int val) {

        // TODO: Implement sorted insert

        // Insert the given value in a sorted manner in the linked list

        return false;

    }

    bool deletefromstart() {

        if (head == nullptr) {

            return false;

        }

        Node\* temp = head;

        head = head->next;

        if (head != nullptr) {

            head->prev = nullptr;

        }

        delete temp;

        return true;

    }

    bool deletefromend() {

        if (head == nullptr) {

            return false;

        }

        if (head->next == nullptr) {

            delete head;

            head = nullptr;

            return true;

        }

        Node\* current = head;

        while (current->next != nullptr) {

            current = current->next;

        }

        current->prev->next = nullptr;

        delete current;

        return true;

    }

    Node\* search(int val) {

        Node\* current = head;

        while (current != nullptr) {

            if (current->data == val) {

                return current;

            }

            current = current->next;

        }

        return nullptr;

    }

    bool deletespecificvalue(int val) {

        Node\* nodeToDelete = search(val);

        if (nodeToDelete == nullptr) {

            return false;

        }

        if (nodeToDelete == head) {

            head = head->next;

            if (head != nullptr) {

                head->prev = nullptr;

            }

        } else {

            nodeToDelete->prev->next = nodeToDelete->next;

            if (nodeToDelete->next != nullptr) {

                nodeToDelete->next->prev = nodeToDelete->prev;

            }

        }

        delete nodeToDelete;

        return true;

    }

    void display() {

        Node\* current = head;

        while (current != nullptr) {

            cout << current->data << " ";

            current = current->next;

        }

        cout << endl;

    }

};

int main() {

    doublylinklist list;

    list.insertatstart(3);

    list.insertatstart(2);

    list.insertatstart(1);

    list.insertatend(4);

    list.insertatend(5);

    cout << "Original List: ";

    list.display();

    list.deletefromstart();

    list.deletefromend();

    list.deletespecificvalue(3);

    cout << "Modified List: ";

    list.display();

    return 0;

}