# Assignment No 4

Name :- Shreyas Anil Dimbar

PRN :- 245100140

Batch :- S6

* Q1 - Write a Menu Driven program to perform following operations on Doubly linked list.
  + Insert new element at Beginning, End and middle position.
  + Delete element from Beginning, End and middle position.
  + Display Linked List.

Code - #include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node(int data) {

        this->data = data;

        this->next = NULL;

        this->prev = NULL;

    }

};

class DoublyLinkedList {

public:

    Node\* head;

    DoublyLinkedList() {

        head = NULL;

    }

    void insertAtBeginning(int data) {

        Node\* newNode = new Node(data);

        if (head != NULL) {

            head->prev = newNode;

        }

        newNode->next = head;

        head = newNode;

        cout << "Inserted " << data << " at beginning." << endl;

    }

    void insertAtEnd(int data) {

        Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

            cout << "Inserted " << data << " at end." << endl;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

        cout << "Inserted " << data << " at end." << endl;

    }

    void insertAtMiddle(int data, int pos) {

        if (pos <= 1) {

            insertAtBeginning(data);

            return;

        }

        Node\* newNode = new Node(data);

        Node\* temp = head;

        for (int i = 1; i < pos - 1 && temp != NULL; i++) {

            temp = temp->next;

        }

        if (temp == NULL) {

            cout << "Invalid position." << endl;

            return;

        }

        newNode->next = temp->next;

        if (temp->next != NULL) {

            temp->next->prev = newNode;

        }

        temp->next = newNode;

        newNode->prev = temp;

        cout << "Inserted " << data << " at position " << pos << "." << endl;

    }

    void deleteFromBeginning() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        Node\* temp = head;

        head = head->next;

        if (head != NULL) {

            head->prev = NULL;

        }

        cout << "Deleted " << temp->data << " from beginning." << endl;

        delete temp;

    }

    void deleteFromEnd() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        if (head->next == NULL) {

            deleteFromBeginning();

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->prev->next = NULL;

        cout << "Deleted " << temp->data << " from end." << endl;

        delete temp;

    }

    void deleteFromMiddle(int pos) {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        if (pos <= 1) {

            deleteFromBeginning();

            return;

        }

        Node\* temp = head;

        for (int i = 1; i < pos && temp != NULL; i++) {

            temp = temp->next;

        }

        if (temp == NULL) {

            cout << "Invalid position." << endl;

            return;

        }

        temp->prev->next = temp->next;

        if (temp->next != NULL) {

            temp->next->prev = temp->prev;

        }

        cout << "Deleted " << temp->data << " from position " << pos << "." << endl;

        delete temp;

    }

    void display() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        Node\* temp = head;

        cout << "List: NULL <-> ";

        while (temp != NULL) {

            cout << temp->data << " <-> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main() {

    DoublyLinkedList list;

    int choice, value, position;

    while (true) {

        cout << "\n--- Doubly Linked List Menu ---" << endl;

        cout << "1. Insert at Beginning" << endl;

        cout << "2. Insert at End" << endl;

        cout << "3. Insert at Position" << endl;

        cout << "4. Delete from Beginning" << endl;

        cout << "5. Delete from End" << endl;

        cout << "6. Delete from Position" << endl;

        cout << "7. Display List" << endl;

        cout << "8. Exit" << endl;

        cout << "-----------------------------" << endl;

        cout << "Enter choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtBeginning(value);

                break;

            case 2:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtEnd(value);

                break;

            case 3:

                cout << "Enter value: ";

                cin >> value;

                cout << "Enter position: ";

                cin >> position;

                list.insertAtMiddle(value, position);

                break;

            case 4:

                list.deleteFromBeginning();

                break;

            case 5:

                list.deleteFromEnd();

                break;

            case 6:

                cout << "Enter position: ";

                cin >> position;

                list.deleteFromMiddle(position);

                break;

            case 7:

                list.display();

                break;

            case 8:

                cout << "Exiting." << endl;

                return 0;

            default:

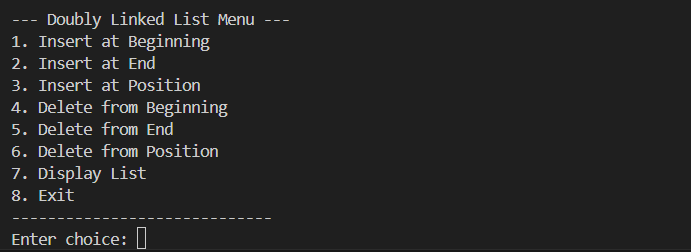
                cout << "Invalid choice." << endl;

        }

    }

    return 0;

}

Output - 

* Q 2 - Write a program to Reverse Doubly Linked List.

Code – #include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node(int data) {

        this->data = data;

        this->next = NULL;

        this->prev = NULL;

    }

};

class DoublyLinkedList {

public:

    Node\* head;

    DoublyLinkedList() {

        head = NULL;

    }

    void insertAtEnd(int data) {

        Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

    void reverse() {

        if (head == NULL || head->next == NULL) {

            cout << "List reversed." << endl;

            return;

        }

        Node\* temp = NULL;

        Node\* current = head;

        while (current != NULL) {

            temp = current->prev;

            current->prev = current->next;

            current->next = temp;

            current = current->prev;

        }

        if (temp != NULL) {

            head = temp->prev;

        }

        cout << "List reversed." << endl;

    }

    void display() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        Node\* temp = head;

        cout << "List: NULL <-> ";

        while (temp != NULL) {

            cout << temp->data << " <-> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main() {

    DoublyLinkedList list;

    list.insertAtEnd(10);

    list.insertAtEnd(20);

    list.insertAtEnd(30);

    list.insertAtEnd(40);

    cout << "Original List:" << endl;

    list.display();

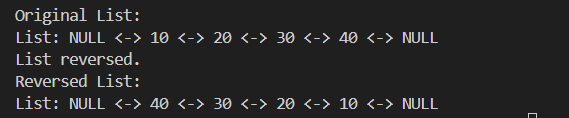
    list.reverse();

    cout << "Reversed List:" << endl;

    list.display();

    return 0;

}

Output - 

* Q3 - Write a program to Sort given Doubly Linked List.

Code - #include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node(int data) {

        this->data = data;

        this->next = NULL;

        this->prev = NULL;

    }

};

class DoublyLinkedList {

public:

    Node\* head;

    DoublyLinkedList() {

        head = NULL;

    }

    void insertAtEnd(int data) {

        Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

    void sort() {

        if (head == NULL) {

            return;

        }

        bool swapped;

        Node\* current;

        Node\* last = NULL;

        do {

            swapped = false;

            current = head;

            while (current->next != last) {

                if (current->data > current->next->data) {

                    int temp = current->data;

                    current->data = current->next->data;

                    current->next->data = temp;

                    swapped = true;

                }

                current = current->next;

            }

            last = current;

        } while (swapped);

        cout << "List sorted." << endl;

    }

    void display() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        Node\* temp = head;

        cout << "List: NULL <-> ";

        while (temp != NULL) {

            cout << temp->data << " <-> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main() {

    DoublyLinkedList list;

    list.insertAtEnd(40);

    list.insertAtEnd(20);

    list.insertAtEnd(50);

    list.insertAtEnd(10);

    list.insertAtEnd(30);

    cout << "Original List:" << endl;

    list.display();

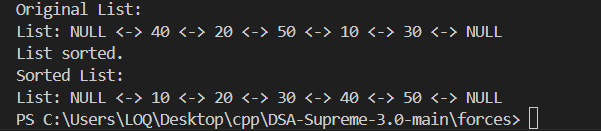
    list.sort();

    cout << "Sorted List:" << endl;

    list.display();

    return 0;

}

Output - 

* Q4 - Write a program to concatenate two Doubly Linked Lists.

Code – #include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node(int data) {

        this->data = data;

        this->next = NULL;

        this->prev = NULL;

    }

};

class DoublyLinkedList {

public:

    Node\* head;

    DoublyLinkedList() {

        head = NULL;

    }

    void insertAtEnd(int data) {

        Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

    void concatenate(DoublyLinkedList &list2) {

        if (head == NULL) {

            head = list2.head;

            list2.head = NULL;

            cout << "Lists concatenated." << endl;

            return;

        }

        if (list2.head == NULL) {

            cout << "Lists concatenated." << endl;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = list2.head;

        list2.head->prev = temp;

        list2.head = NULL;

        cout << "Lists concatenated." << endl;

    }

    void display() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        Node\* temp = head;

        cout << "List: NULL <-> ";

        while (temp != NULL) {

            cout << temp->data << " <-> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main() {

    DoublyLinkedList list1;

    list1.insertAtEnd(10);

    list1.insertAtEnd(20);

    list1.insertAtEnd(30);

    DoublyLinkedList list2;

    list2.insertAtEnd(40);

    list2.insertAtEnd(50);

    cout << "First List:" << endl;

    list1.display();

    cout << "Second List:" << endl;

    list2.display();

    list1.concatenate(list2);

    cout << "Concatenated List:" << endl;

    list1.display();

    return 0;

}

Output - 