

|  |
| --- |
| **LAB 8** of DSA LAB |

**Submitted by:**

**Name:** Muhammad shuraim

**Roll no:** SU92-S24-BSSEM-103

**Section:** 3A

**Submitted to**

Sir Rasikh Ali

**Question 1**

**OUTPUT**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int d) {

data = d;

next = nullptr;

}

};

class LinkedList {

public:

Node\* head;

LinkedList() {

head = nullptr;

}

void insert\_at\_end(int data) {

Node\* newNode = new Node(data);

if (head == nullptr) {

head = newNode;

}

else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

};

Node\* merge(Node\* head1, Node\* head2) {

if (!head1) return head2;

if (!head2) return head1;

if (head1->data < head2->data) {

head1->next = merge(head1->next, head2);

return head1;

}

else {

head2->next = merge(head1, head2->next);

return head2;

}

}

int main() {

LinkedList list1, list2;

list1.insert\_at\_end(6);

list1.insert\_at\_end(5);

list1.insert\_at\_end(4);

list2.insert\_at\_end(3);

list2.insert\_at\_end(2);

list2.insert\_at\_end(1);

cout << "First Linked List: ";

list1.display();

cout << "Second Linked List: ";

list2.display();

LinkedList mergedList;

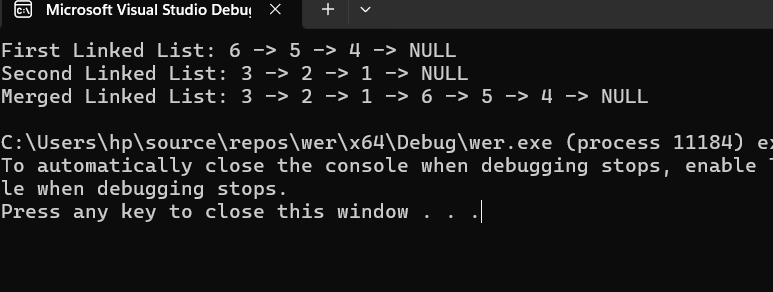
mergedList.head = merge(list1.head, list2.head);

cout << "Merged Linked List: ";

mergedList.display();

return 0;

}



Explanation

This program merges two \*\*singly linked lists\*\* in sorted order using \*\*recursion\*\*.

**1. Node Structure**

- The `Node` class represents each node with `data` and `next` pointer.

2. LinkedList Class

- insert\_at\_end(int data)`\*\* → Inserts a node at the end of the list.

- display()`\*\* → Prints the linked list.

3. Merge Function

- merge(Node\* head1, Node\* head2)`

- Recursively merges two sorted linked lists.

- Compares `head1->data` and `head2->data`, then links the smaller node to the merged result.

. Main Function

1. Creates \*\*two linked lists\*\*:

- List 1: \*\*6 → 5 → 4\*\*

- List 2: \*\*3 → 2 → 1\*\*

2. Merges them using `merge()` function.

3. Displays the merged linked list.

**Question 2**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int d) {

data = d;

next = nullptr;

prev = nullptr;

}

};

class DoublyLinkedList {

public:

Node\* head;

DoublyLinkedList() {

head = nullptr;

}

void insert\_at\_end(int data) {

Node\* newNode = new Node(data);

if (head == nullptr) {

head = newNode;

}

else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

};

Node\* merge(Node\* head1, Node\* head2) {

if (!head1) return head2;

if (!head2) return head1;

Node\* mergedHead = nullptr;

if (head1->data < head2->data) {

mergedHead = head1;

head1 = head1->next;

}

else {

mergedHead = head2;

head2 = head2->next;

}

Node\* temp = mergedHead;

while (head1 && head2) {

if (head1->data < head2->data) {

temp->next = head1;

head1->prev = temp;

head1 = head1->next;

}

else {

temp->next = head2;

head2->prev = temp;

head2 = head2->next;

}

temp = temp->next;

}

if (head1) temp->next = head1, head1->prev = temp;

if (head2) temp->next = head2, head2->prev = temp;

return mergedHead;

}

int main() {

DoublyLinkedList list1, list2;

list1.insert\_at\_end(1);

list1.insert\_at\_end(3);

list1.insert\_at\_end(5);

list2.insert\_at\_end(2);

list2.insert\_at\_end(4);

list2.insert\_at\_end(6);

cout << "First Doubly Linked List: ";

list1.display();

cout << "Second Doubly Linked List: ";

list2.display();

DoublyLinkedList mergedList;

mergedList.head = merge(list1.head, list2.head);

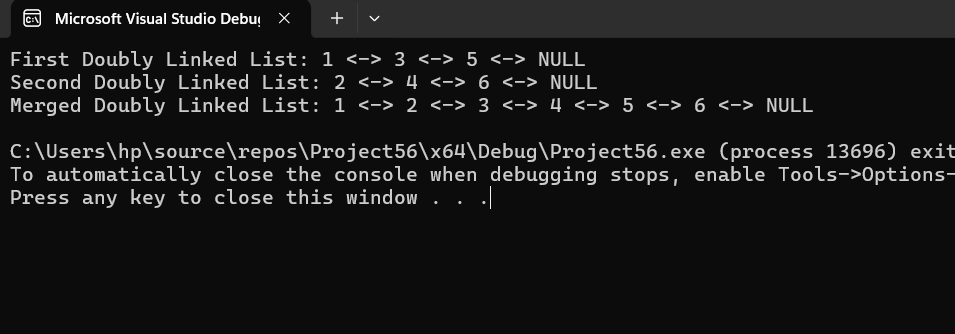
cout << "Merged Doubly Linked List: ";

mergedList.display();

return 0;

}

**OUTPUT**



**EXPLANATION**

This program merges two doubly linked lists in sorted order.

1. Node Structure:

- The `Node` class represents a node with `data`, `next` (forward link), and `prev` (backward link).

2. DoublyLinkedList Class:

`insert\_at\_end(int data)→ Inserts a node at the end of the list.

`display()`→ Prints the list in forward order.

3. Merge Function (`merge(Node\* head1, Node\* head2)`)

- Uses an iterative approach to merge two sorted doubly linked lists.

- Compares nodes from both lists and links them in sorted order.

- Updates both `next` and `prev` pointers to maintain doubly linked structure

4. Main Function

1. Creates two sorted doubly linked lists

- List 1: \*\*1 ↔ 3 ↔ 5\*\*

- List 2: \*\*2 ↔ 4 ↔ 6\*\*

2. Merges them using `merge()` function.

3. Displays the merged \*\*doubly linked list\*\*.