

|  |
| --- |
| **LAB 6** of DSA LAB |

**Submitted by:**

**Name:** Muhammad shuraim

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

**Section:** 3A

**Submitted to**

Sir Rasikh Ali

**Question 1**

#include <iostream>

using namespace std;

class Node {

public:

int value;

Node\* next;

Node(int val) {

value = val;

next = nullptr;

}

};

class LinkedList {

public:

Node\* head;

LinkedList() {

head = nullptr;

}

void insert\_at\_start(int val) {

Node\* newnode = new Node(val);

newnode->next = head;

head = newnode;

}

void delete\_at\_start() {

if (head == nullptr) {

cout << "List is already empty.\n";

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

void delete\_at\_end() {

if (head == nullptr) {

cout << "List is already empty.\n";

return;

}

if (head->next == nullptr) {

delete head;

head = nullptr;

return;

}

Node\* temp = head;

while (temp->next->next != nullptr) {

temp = temp->next;

}

delete temp->next;

temp->next = nullptr;

}

void delete\_nth\_node(int n) {

if (head == nullptr) {

cout << "List is empty.\n";

return;

}

if (n == 1) {

delete\_at\_start();

return;

}

Node\* temp = head;

for (int i = 1; temp != nullptr && i < n - 1; i++) {

temp = temp->next;

}

if (temp == nullptr || temp->next == nullptr) {

cout << "Invalid position!\n";

return;

}

Node\* node\_to\_delete = temp->next;

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

delete node\_to\_delete;

}

void delete\_middle\_node() {

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

delete\_at\_start();

return;

}

Node\* turtle = head;

Node\* tiger= head;

Node\* prev = nullptr;

while (tiger != nullptr && tiger->next != nullptr) {

prev = turtle;

turtle = turtle->next;

tiger = tiger->next->next;

}

prev->next = turtle->next;

delete turtle;

}

void display() {

if (head == nullptr) {

cout << "LIST IS EMPTY\n";

return;

}

Node\* temp = head;

while (temp != nullptr) {

cout << temp->value << " -> ";

temp = temp->next;

}

cout << "NULL\n";

}

};

int main() {

LinkedList list;

list.insert\_at\_start(10);

list.insert\_at\_start(20);

list.insert\_at\_start(30);

list.insert\_at\_start(40);

list.insert\_at\_start(50);

cout << "Original List: ";

list.display();

list.delete\_at\_start();

cout << "After deleting first node: ";

list.display();

list.delete\_at\_end();

cout << "After deleting last node: ";

list.display();

list.delete\_nth\_node(2);

cout << "After deleting 2nd node: ";

list.display();

list.delete\_middle\_node();

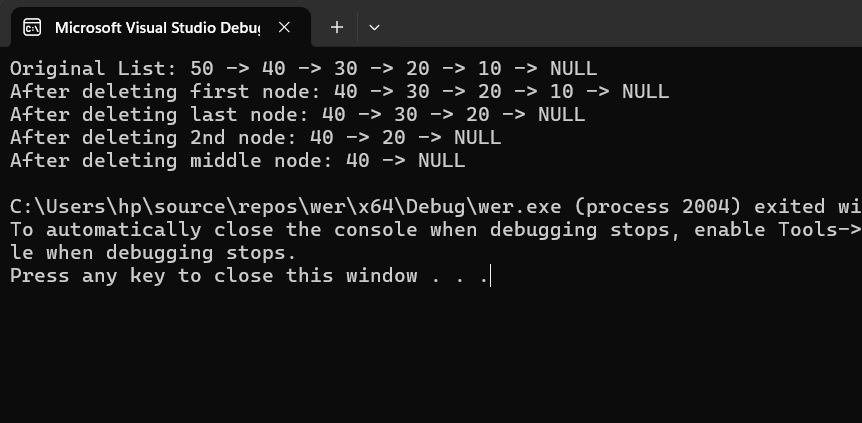
cout << "After deleting middle node: ";

list.display();

return 0;

}

**OUTPUT**



Explanation

* The `Node` class represents each node with a `value` and a pointer `next` to the next node.

**LinkedList Class:**

- \*\*`insert\_at\_start(int val)`\*\* → Adds a node at the beginning.

- \*\*`delete\_at\_start()`\*\* → Deletes the first node.

- \*\*`delete\_at\_end()`\*\* → Deletes the last node.

- \*\*`delete\_nth\_node(int n)`\*\* → Deletes the node at a given position `n`.

- \*\*`delete\_middle\_node()`\*\* → Deletes the middle node using the \*\*tortoise and tiger\*\* approach.

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

**Main Function:**

1. Inserts values \*\*50 → 40 → 30 → 20 → 10\*\* into the list.

2. Performs deletions:

- Deletes the \*\*first node\*\*.

- Deletes the \*\*last node\*\*.

- Deletes the \*\*2nd node\*\*.

- Deletes the \*\*middle node\*\*.