

|  |
| --- |
| **LAB 5** 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 insert\_at\_last(int val) {

Node\* newnode = new Node(val);

if (head == nullptr) {

head = newnode;

return;

}

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newnode;

}

void display() {

if (head == nullptr) {

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

return;

}

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

void display\_first\_node() {

if (head == nullptr) {

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

}

else {

cout << "First Node: " << head->value << endl;

}

}

void display\_last\_node() {

if (head == nullptr) {

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

return;

}

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

cout << "Last Node: " << temp->value << endl;

}

void display\_nth\_node(int n) {

if (head == nullptr) {

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

return;

}

Node\* temp = head;

int count = 1;

while (temp != nullptr) {

if (count == n) {

cout << "Nth Node (" << n << "): " << temp->value << endl;

return;

}

temp = temp->next;

count++;

}

cout << "Position " << n << " is out of range!" << endl;

}

void display\_center\_node() {

if (head == nullptr) {

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

return;

}

Node\* turtle = head;

Node\* tiger = head;

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

turtle = turtle->next;

tiger = tiger->next->next;

}

cout << "Center Node: " << turtle->value << endl;

}

};

int main() {

LinkedList obj;

obj.insert\_at\_last(1);

obj.insert\_at\_last(2);

obj.insert\_at\_last(3);

obj.insert\_at\_last(4);

obj.insert\_at\_last(5);

obj.display();

obj.display\_first\_node();

obj.display\_last\_node();

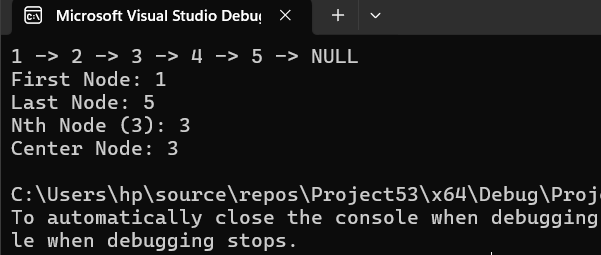
obj.display\_nth\_node(3);

obj.display\_center\_node();

return 0;

}

**OUTPUT**



Explanation

**Display\_first\_node()**

* Prints the first node’s value (head->value).

**Display\_last\_node()**

* Traverses the list until the last node (temp->next == nullptr) and prints its value.

**Display\_nth\_node(int n)**

* Iterates through the list, stopping at the Nth node.
* If n is greater than the list size, prints "Position n is out of range!".

**Display\_center\_node()**

Uses the turtle & tiger pointer approach:

turtle moves one step at a time.

tiger moves two steps at a time.

When tiger reaches the end, turtle is at the center.