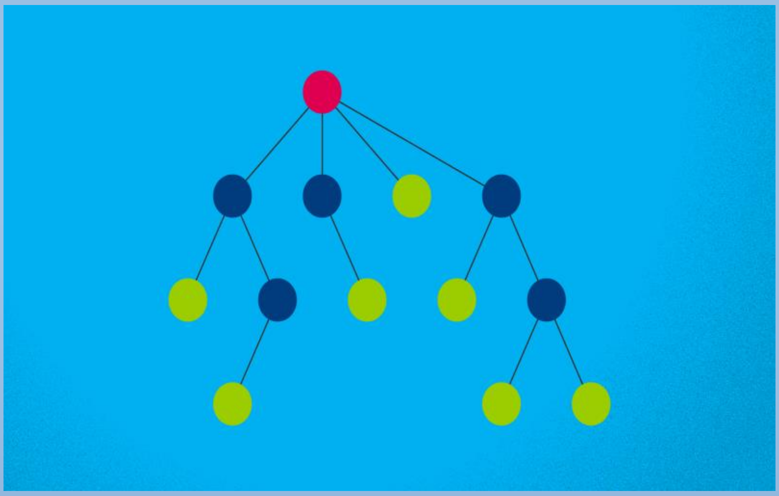
RIPHAH INTERNATIONAL UNIVERSITY

DATA STRUCTURE & ALGORITHMS



LAB # 06

NAME : ASAD ULLAH

SAP ID : 55181

SECTION : SE 3-2

TASK # T-01 : Compare linked list and array performance for insertion and deletion in terms of time complexity.

ANSWER # T-01 :

Linked List:

A **linked list** is a dynamic data structure where elements (nodes) are connected through pointers.

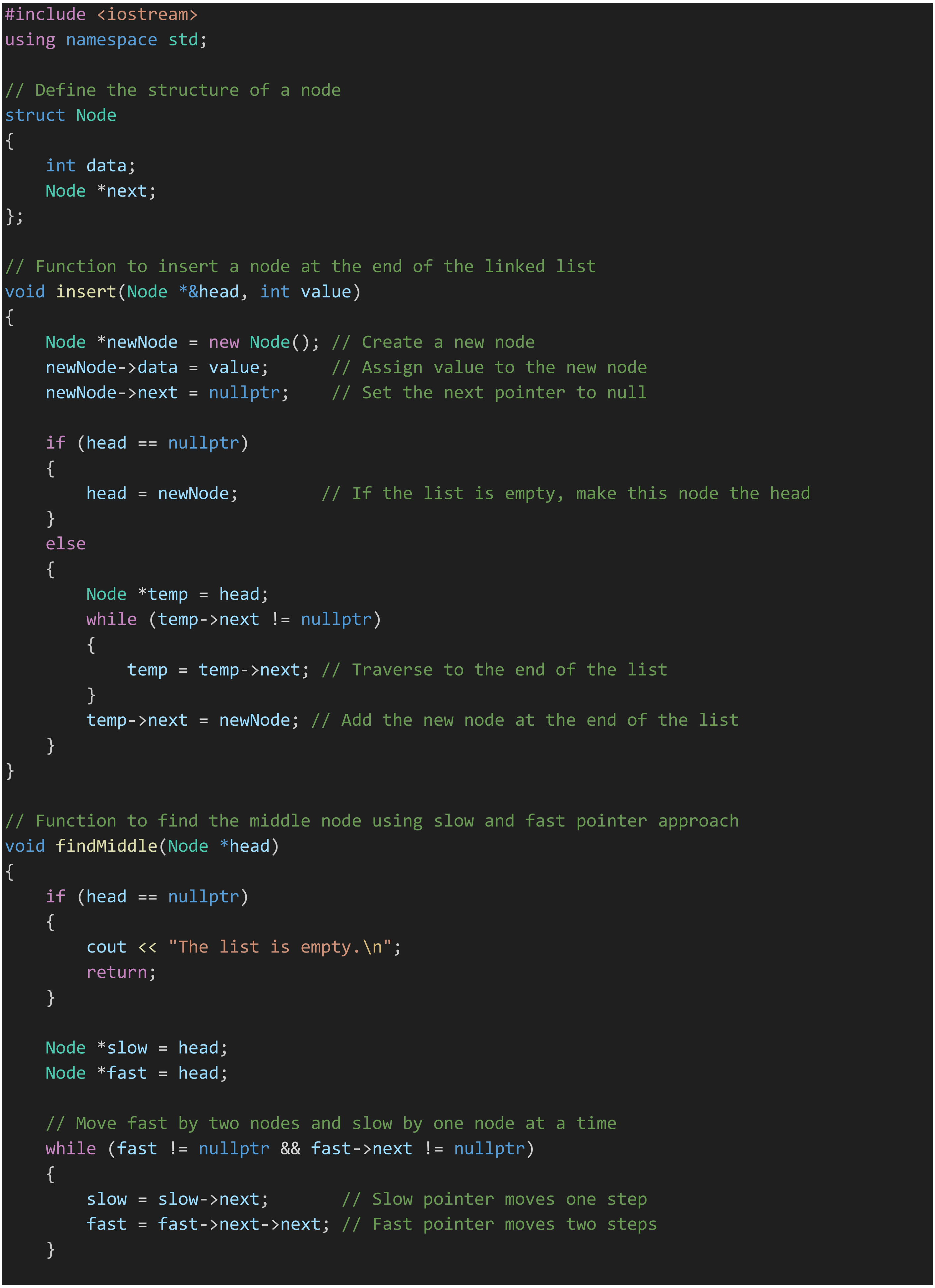
- **Insertion**:
  - **At the beginning**: (Constant time) – Insertions at the head are efficient as you just update the head pointer.
  - **At the end**: (Linear time) – You need to traverse the entire list to reach the end before inserting.
  - **At a specific position**:Requires traversal to the desired position before insertion.
- **Deletion**:
  - **At the beginning**: Simply update the head pointer to the next node.
  - **At the end**: Requires traversal to the last node to remove it.
  - **At a specific position**: Need to traverse to the node before the one being deleted.

Array:

An **array** is a contiguous block of memory with a fixed size or resizable (in dynamic arrays like vectors).

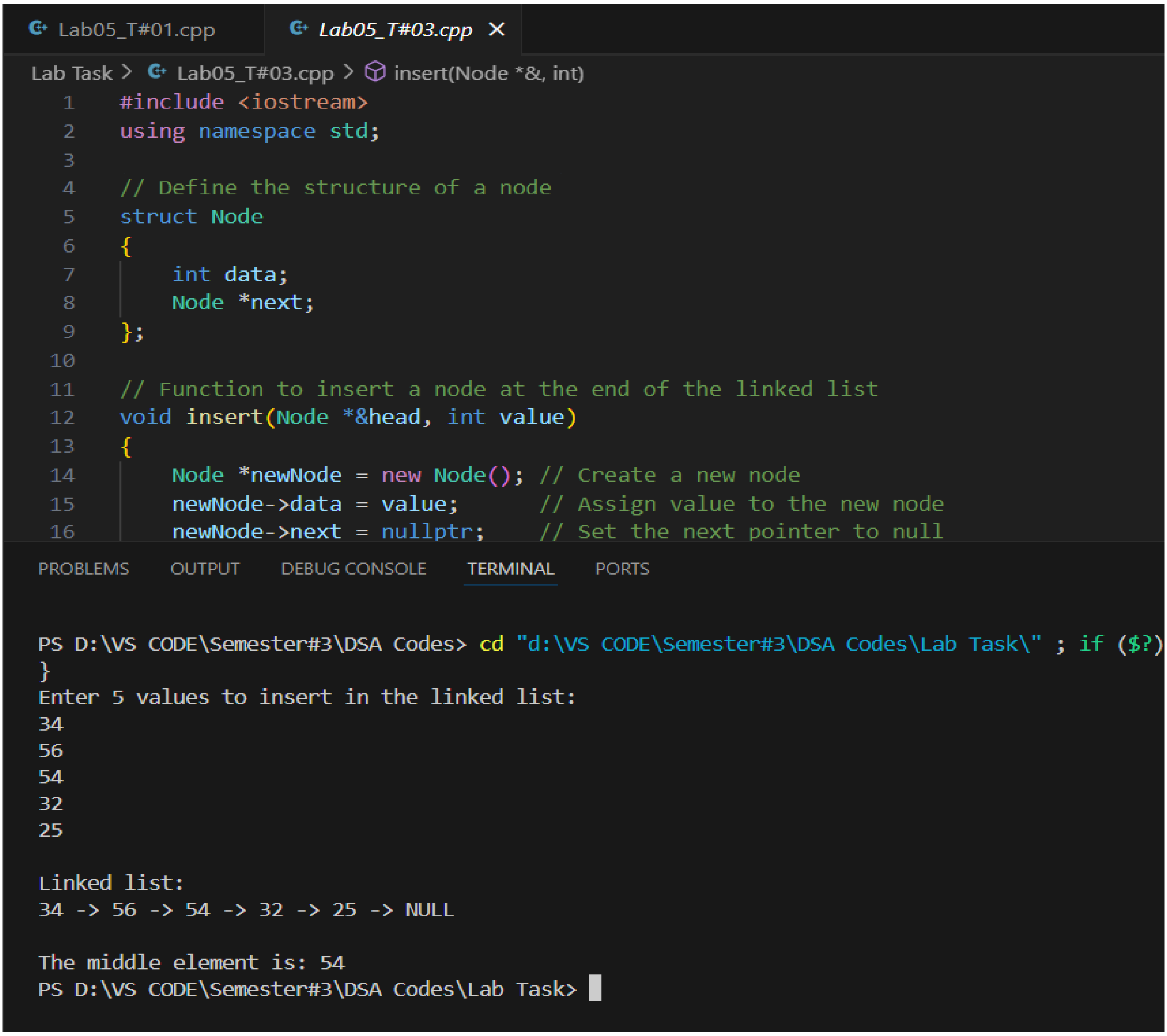
- **Insertion**:
  - **At the beginning**: Requires shifting all elements to the right to make room for the new element.
  - **At the end**: (Amortized time for dynamic arrays) – Insertions at the end are fast unless the array is full and needs resizing.
  - **At a specific position**: Requires shifting elements after the insertion point.
- **Deletion**:
  - **At the beginning**: Requires shifting all elements to the left after removing the first element.
  - **At the end**: Removing the last element is quick.
  - **At a specific position**: Requires shifting elements to maintain the array's continuity.

CODE # T-02 :



```cpp
#include <iostream>
using namespace std;

// Define the structure of a node
struct Node
{
    int data;
    Node *next;
};

// Function to insert a node at the end of the linked list
void insert(Node *&head, int value)
{
    Node *newNode = new Node(); // Create a new node
    newNode->data = value;      // Assign value to the new node
    newNode->next = nullptr;    // Set the next pointer to null

    if (head == nullptr)
    {
        head = newNode;        // If the list is empty, make this node the head
    }
    else
    {
        Node *temp = head;
        while (temp->next != nullptr)
        {
            temp = temp->next; // Traverse to the end of the list
        }
        temp->next = newNode; // Add the new node at the end of the list
    }
}

// Function to find the middle node using slow and fast pointer approach
void findMiddle(Node *head)
{
    if (head == nullptr)
    {
        cout << "The list is empty.\n";
        return;
    }

    Node *slow = head;
    Node *fast = head;

    // Move fast by two nodes and slow by one node at a time
    while (fast != nullptr && fast->next != nullptr)
    {
        slow = slow->next;       // Slow pointer moves one step
        fast = fast->next->next; // Fast pointer moves two steps
    }
    // When fast pointer reaches the end, slow pointer is at the middle
    cout << "\nThe middle element is: " << slow->data << endl;
}

// Function to display the linked list
void display(Node *head)
{
    if (head == nullptr)
    {
        cout << "List is empty.\n";
        return;
    }

    Node *temp = head;
    while (temp != nullptr)
    {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

int main()
{
    Node *head = nullptr;
    int value;

    // Insert some values into the linked list
    cout << "Enter 5 values to insert in the linked list:"<<endl;
    for (int i = 0; i < 5; i++)
    {
        cin >> value;
        insert(head, value);
    }

    cout << "\nLinked list: "<<endl;
    display(head);

    // Find and display the middle of the linked list
    findMiddle(head);

    return 0;
}
```

OUTPUT # T-02 :

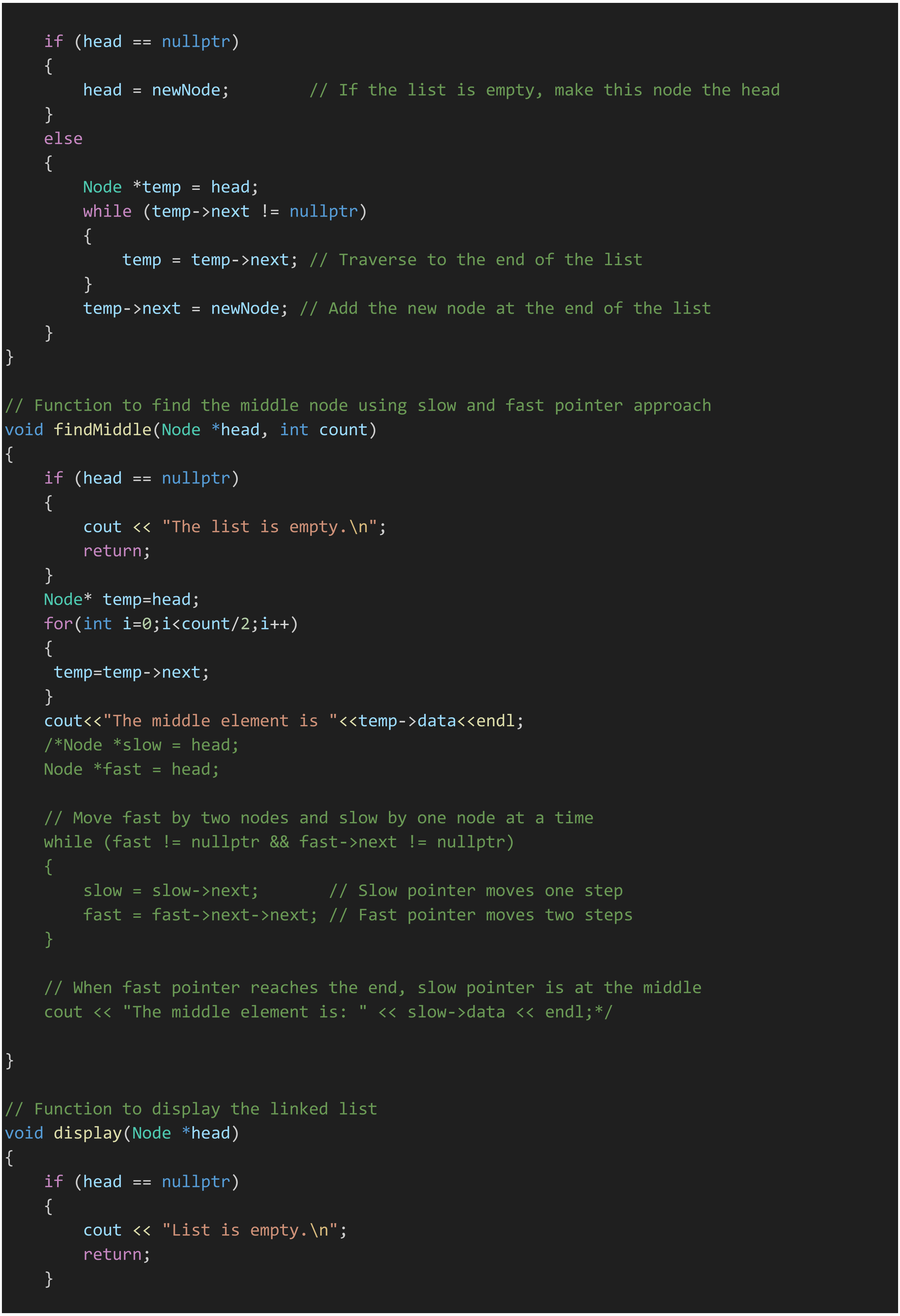


```cpp
#include <iostream>
using namespace std;

// Define the structure of a node
struct Node
{
    int data;
    Node *next;
};

// Function to insert a node at the end of the linked list
void insert(Node *&head, int value)
{
    Node *newNode = new Node(); // Create a new node
    newNode->data = value;      // Assign value to the new node
    newNode->next = nullptr;    // Set the next pointer to null
```

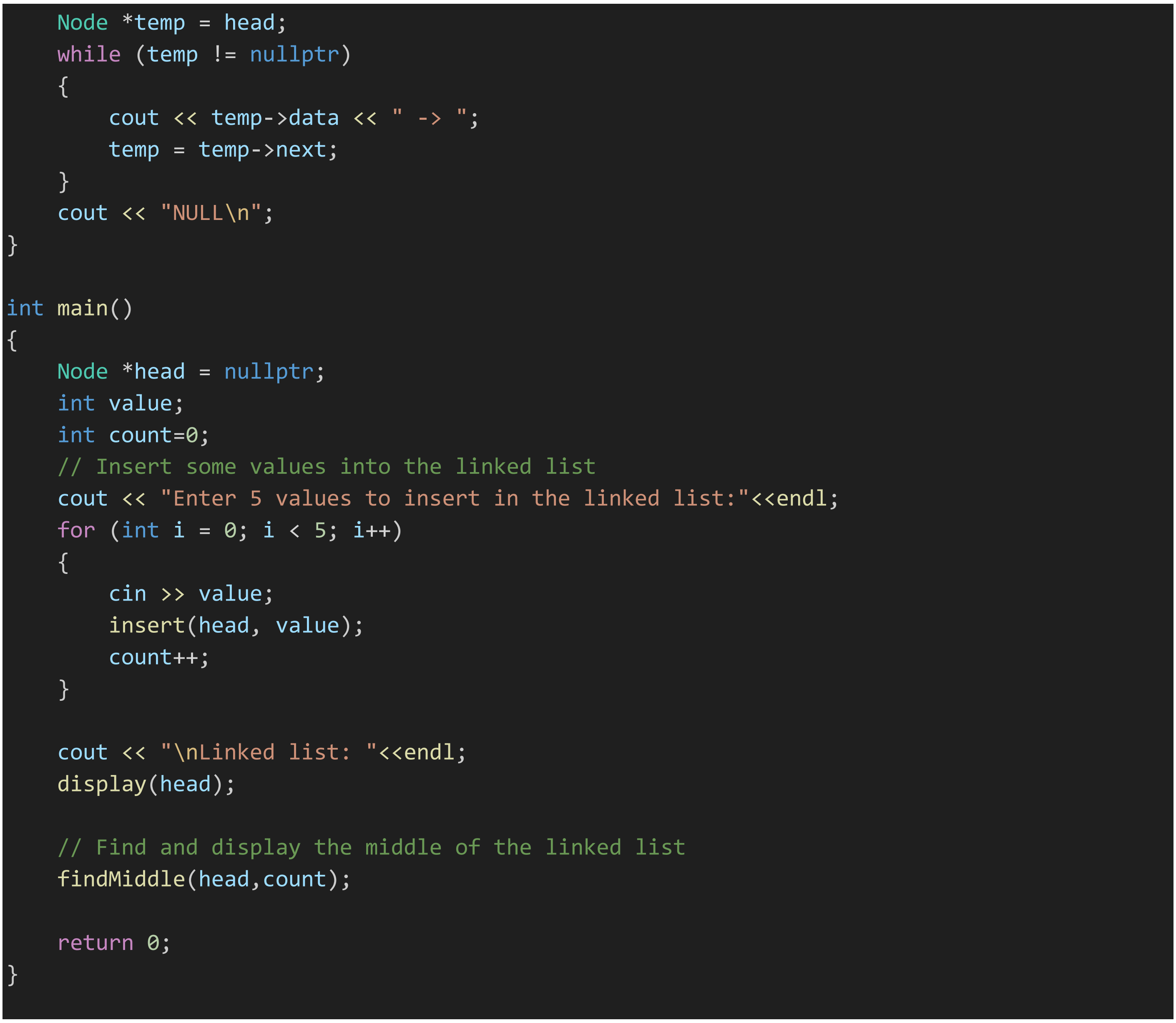
```
PS D:\VS CODE\Semester#3\DSA Codes> cd "d:\VS CODE\Semester#3\DSA Codes\Lab Task\" ; if ($?)
}
Enter 5 values to insert in the linked list:
34
56
54
32
25

Linked list:
34 -> 56 -> 54 -> 32 -> 25 -> NULL

The middle element is: 54
PS D:\VS CODE\Semester#3\DSA Codes\Lab Task>
```

CODE # T-03 :

```cpp
#include <iostream>
using namespace std;

// Define the structure of a node
struct Node
{
    int data;
    Node *next;
};

// Function to insert a node at the end of the linked list
void insert(Node *&head, int value)
{
    Node *newNode = new Node(); // Create a new node
    newNode->data = value;      // Assign value to the new node
    newNode->next = nullptr;    // Set the next pointer to null
```

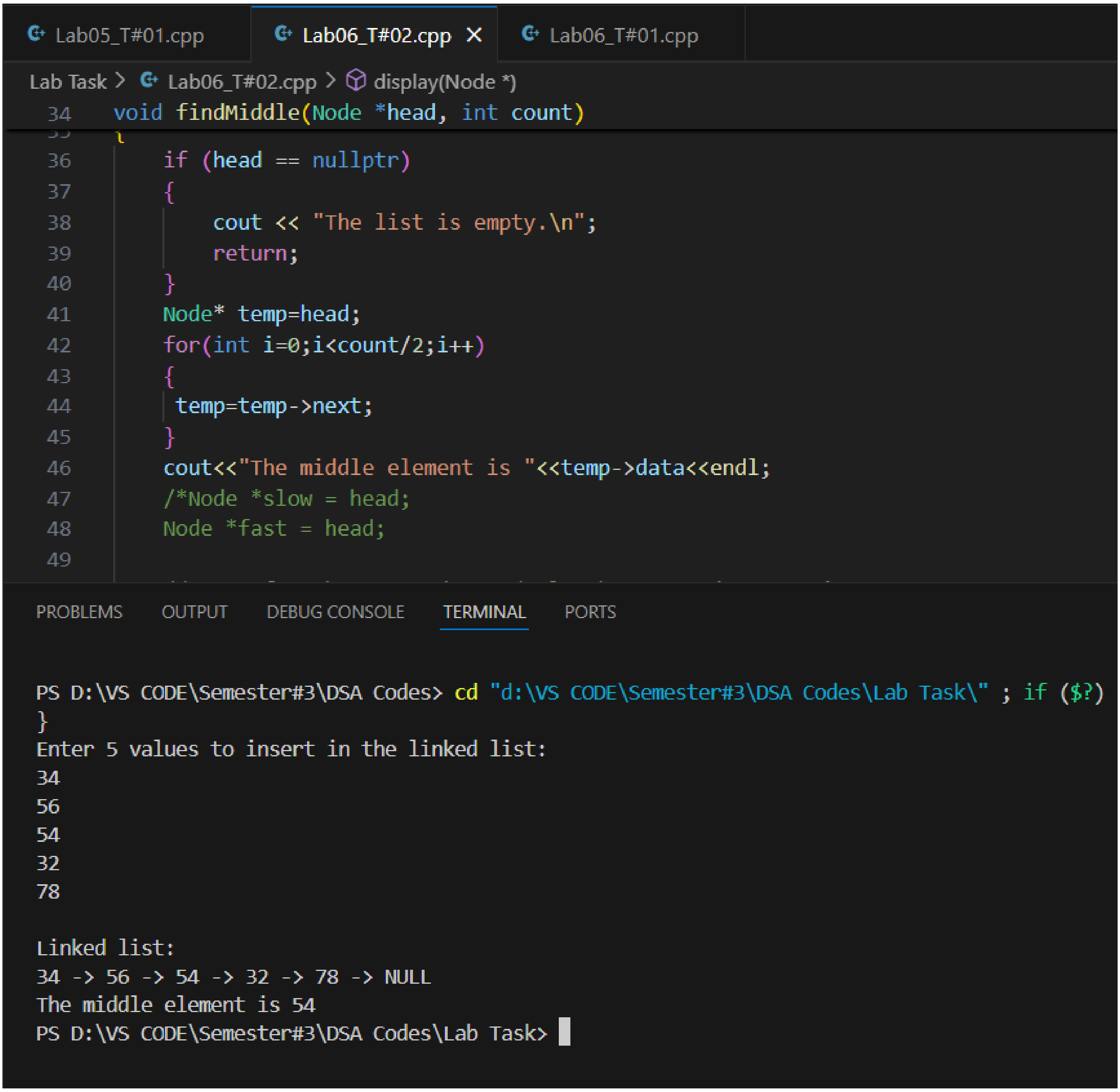


```
    if (head == nullptr)
    {
        head = newNode;        // If the list is empty, make this node the head
    }
    else
    {
        Node *temp = head;
        while (temp->next != nullptr)
        {
            temp = temp->next; // Traverse to the end of the list
        }
        temp->next = newNode; // Add the new node at the end of the list
    }
}

// Function to find the middle node using slow and fast pointer approach
void findMiddle(Node *head, int count)
{
    if (head == nullptr)
    {
        cout << "The list is empty.\n";
        return;
    }
    Node* temp=head;
    for(int i=0;i<count/2;i++)
    {
     temp=temp->next;
    }
    cout<<"The middle element is "<<temp->data<<endl;
    /*Node *slow = head;
    Node *fast = head;

    // Move fast by two nodes and slow by one node at a time
    while (fast != nullptr && fast->next != nullptr)
    {
        slow = slow->next;       // Slow pointer moves one step
        fast = fast->next->next; // Fast pointer moves two steps
    }

    // When fast pointer reaches the end, slow pointer is at the middle
    cout << "The middle element is: " << slow->data << endl;*/

}

// Function to display the linked list
void display(Node *head)
{
    if (head == nullptr)
    {
        cout << "List is empty.\n";
        return;
    }
```



```cpp
    Node *temp = head;
    while (temp != nullptr)
    {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

int main()
{
    Node *head = nullptr;
    int value;
    int count=0;
    // Insert some values into the linked list
    cout << "Enter 5 values to insert in the linked list:"<<endl;
    for (int i = 0; i < 5; i++)
    {
        cin >> value;
        insert(head, value);
        count++;
    }

    cout << "\nLinked list: "<<endl;
    display(head);

    // Find and display the middle of the linked list
    findMiddle(head,count);

    return 0;
}
```

OUTPUT # T-03 :



```
void findMiddle(Node *head, int count)
{
    if (head == nullptr)
    {
        cout << "The list is empty.\n";
        return;
    }
    Node* temp=head;
    for(int i=0;i<count/2;i++)
    {
     temp=temp->next;
    }
    cout<<"The middle element is "<<temp->data<<endl;
    /*Node *slow = head;
    Node *fast = head;

```

```
PS D:\VS CODE\Semester#3\DSA Codes> cd "d:\VS CODE\Semester#3\DSA Codes\Lab Task\" ; if ($?)
}
Enter 5 values to insert in the linked list:
34
56
54
32
78

Linked list:
34 -> 56 -> 54 -> 32 -> 78 -> NULL
The middle element is 54
PS D:\VS CODE\Semester#3\DSA Codes\Lab Task>
```