Bahria University,
Karachi Campus

LAB EXPERIMENT NO. _10_
LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Write all programs to create a linked list.
Task 2	Write a program to implement dynamic Stacks using linked list.
Task 3	Write a program to implement dynamic queue by using linked list.
Task 4	N/A
Task 5	**N/A**
Task 6	**N/A**
Task 7	N/A
Task 8	N/A

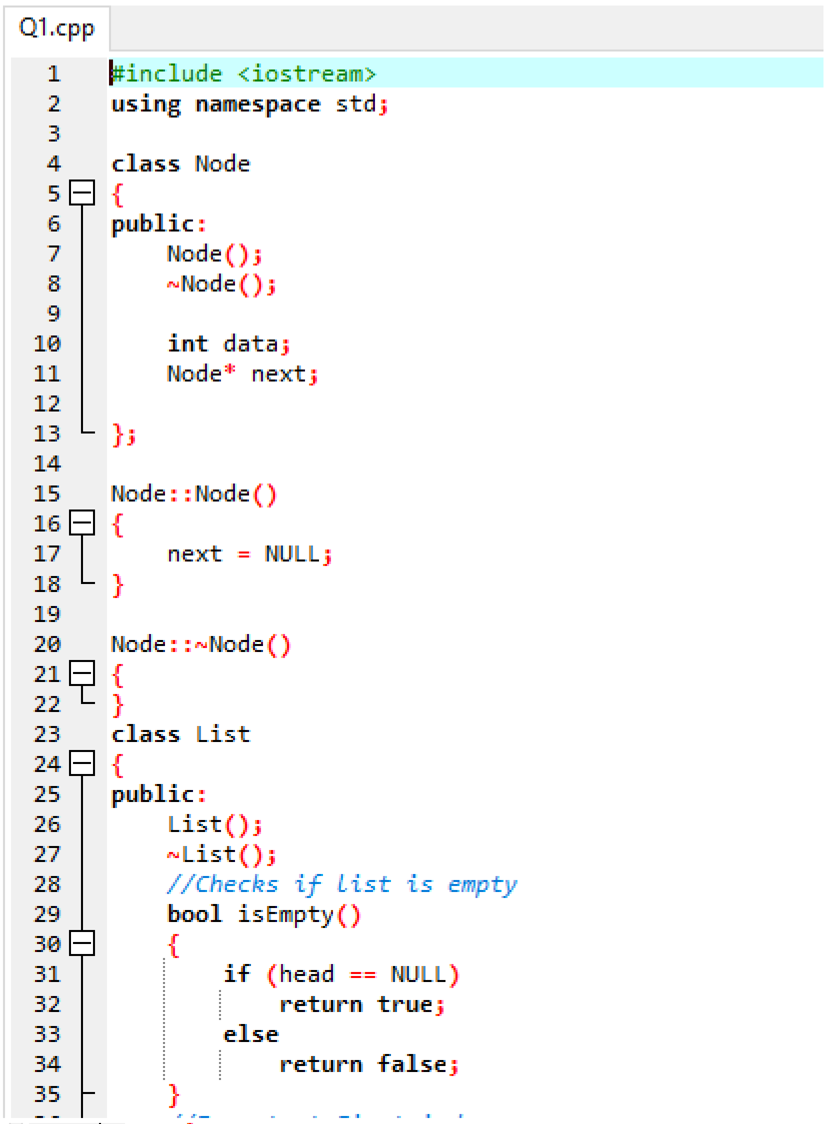
Submitted On:

__04/06/2020__

(Date: DD/MM

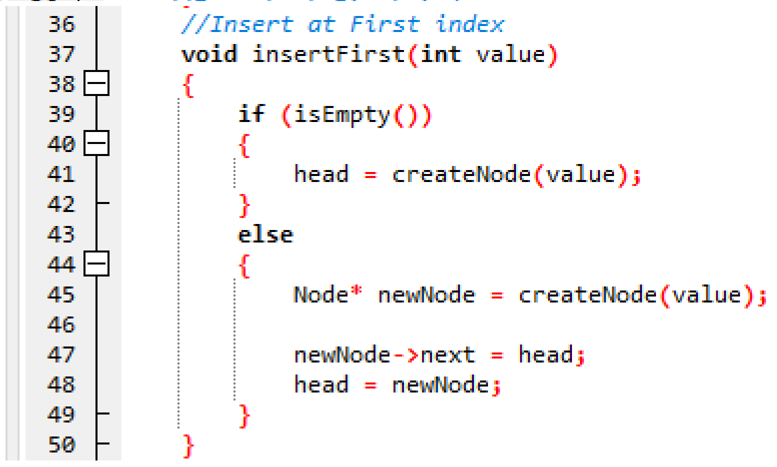
Task No. 1: Write all programs to create a linked list.

Coding:

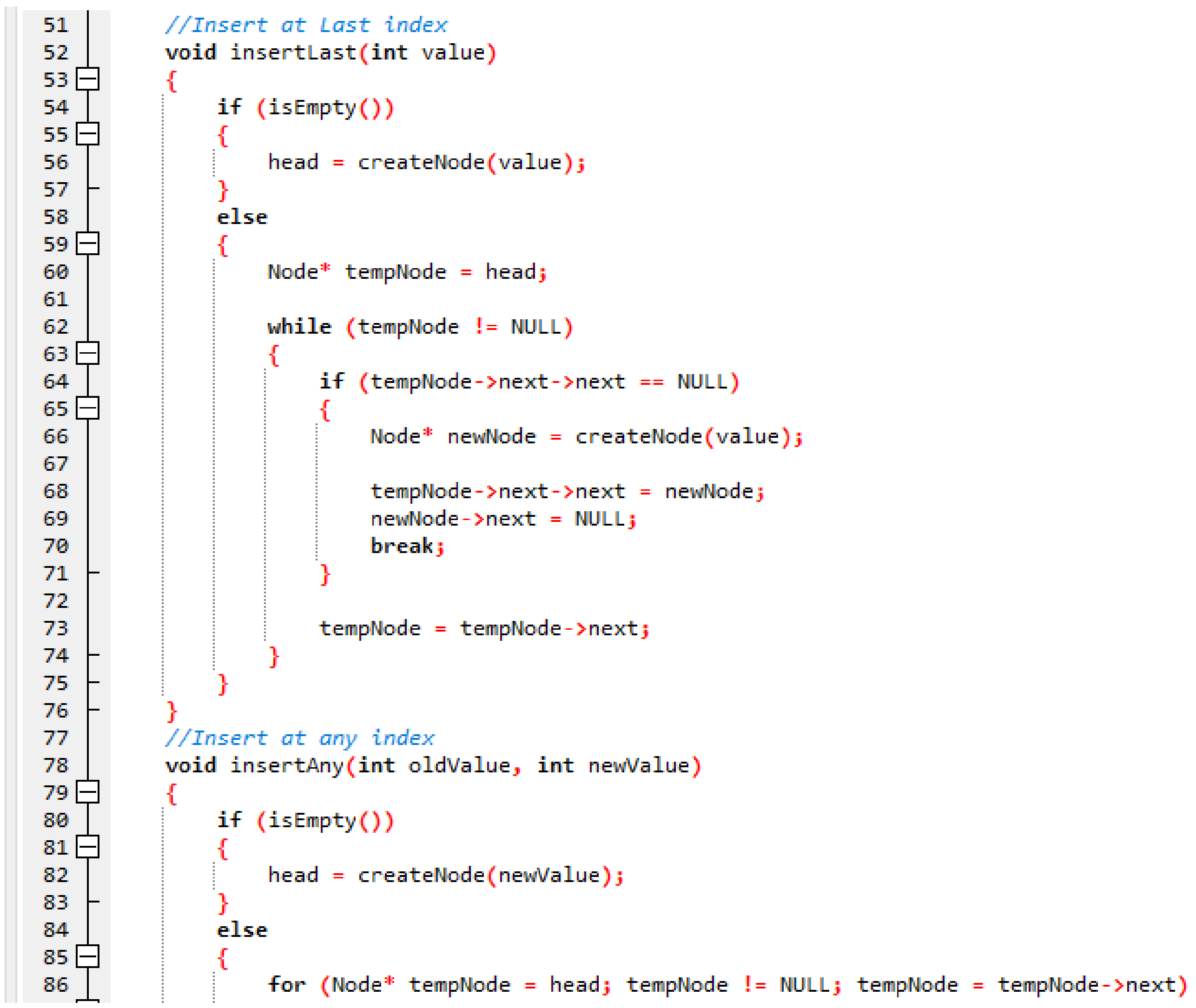


Q1.cpp

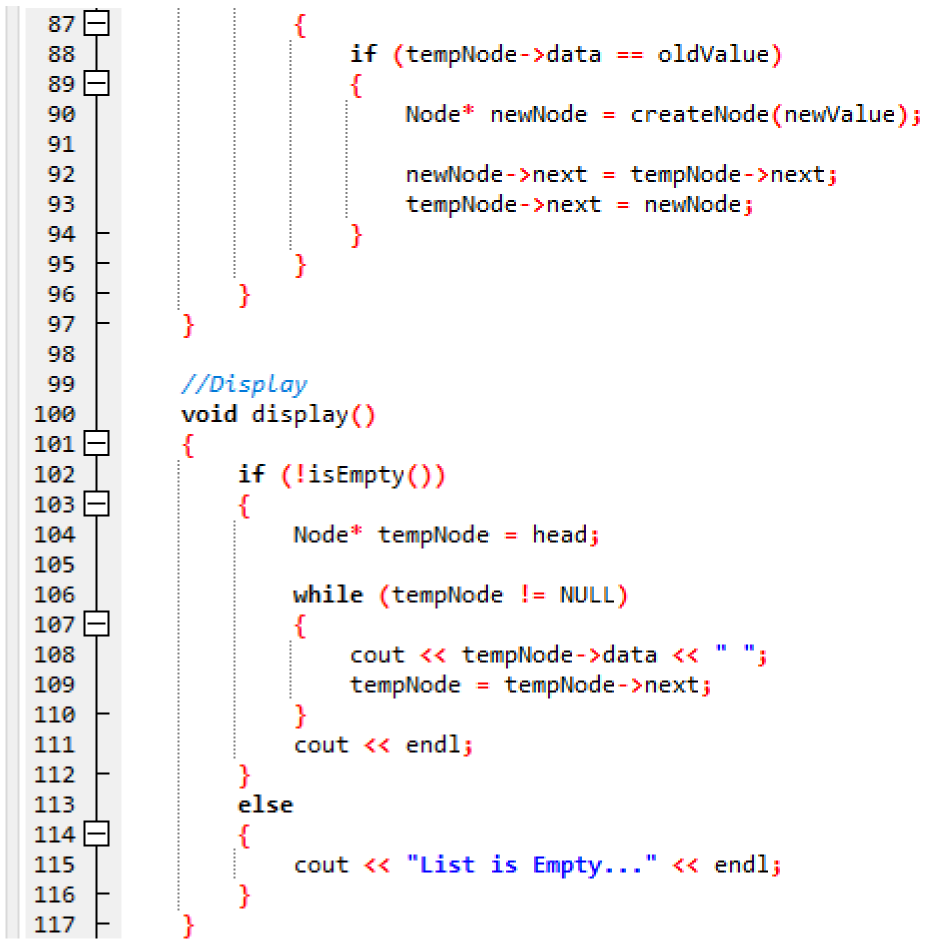
```cpp
#include <iostream>
using namespace std;

class Node
{
public:
    Node();
    ~Node();

    int data;
    Node* next;

};

Node::Node()
{
    next = NULL;
}

Node::~Node()
{
}
class List
{
public:
    List();
    ~List();
    //Checks if list is empty
    bool isEmpty()
    {
        if (head == NULL)
            return true;
        else
            return false;
    }
```



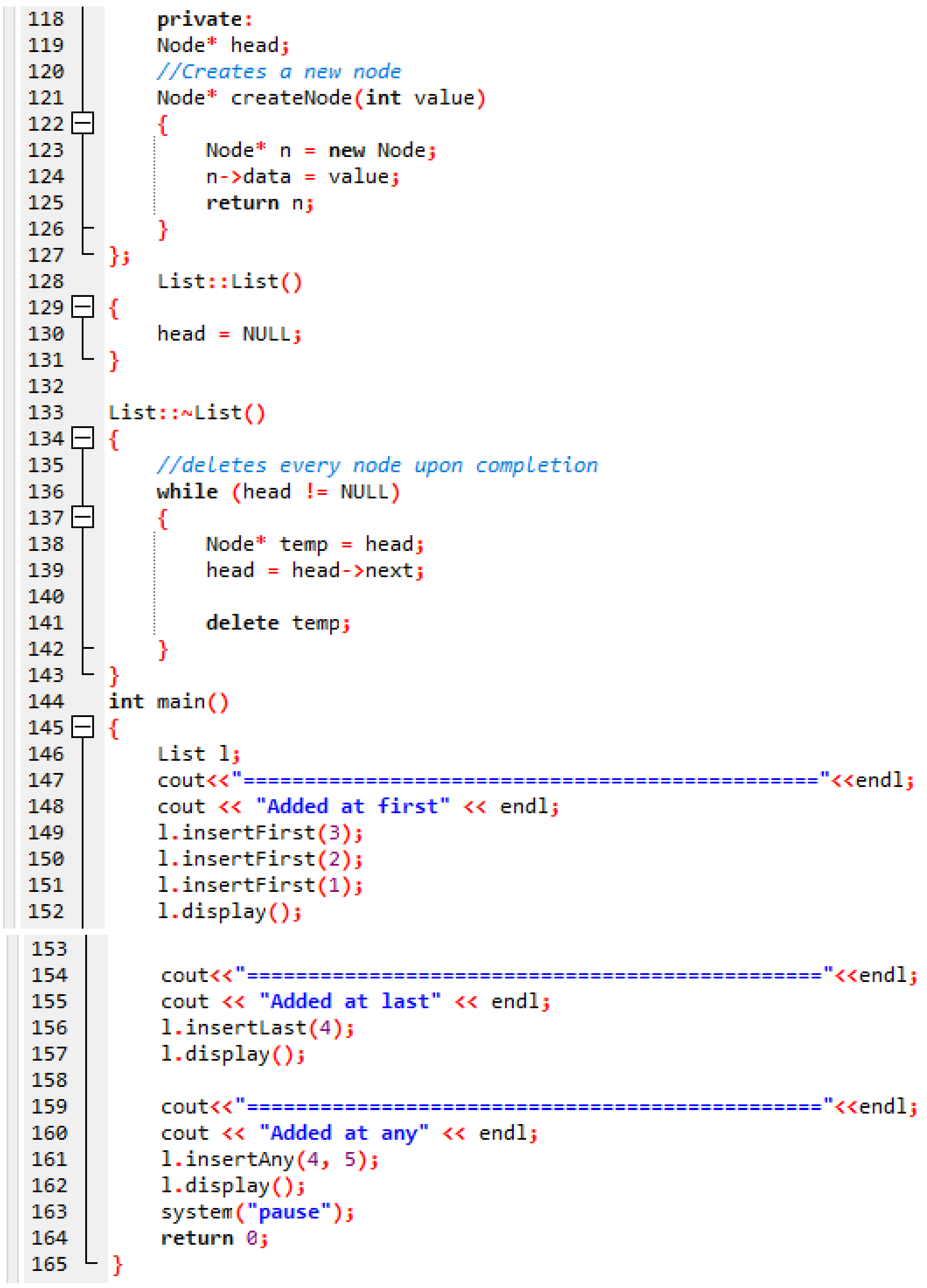
```cpp
    //Insert at First index
    void insertFirst(int value)
    {
        if (isEmpty())
        {
            head = createNode(value);
        }
        else
        {
            Node* newNode = createNode(value);

            newNode->next = head;
            head = newNode;
        }
    }
```



```cpp
//Insert at Last index
void insertLast(int value)
{
    if (isEmpty())
    {
        head = createNode(value);
    }
    else
    {
        Node* tempNode = head;

        while (tempNode != NULL)
        {
            if (tempNode->next->next == NULL)
            {
                Node* newNode = createNode(value);

                tempNode->next->next = newNode;
                newNode->next = NULL;
                break;
            }

            tempNode = tempNode->next;
        }
    }
}
//Insert at any index
void insertAny(int oldValue, int newValue)
{
    if (isEmpty())
    {
        head = createNode(newValue);
    }
    else
    {
        for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
```

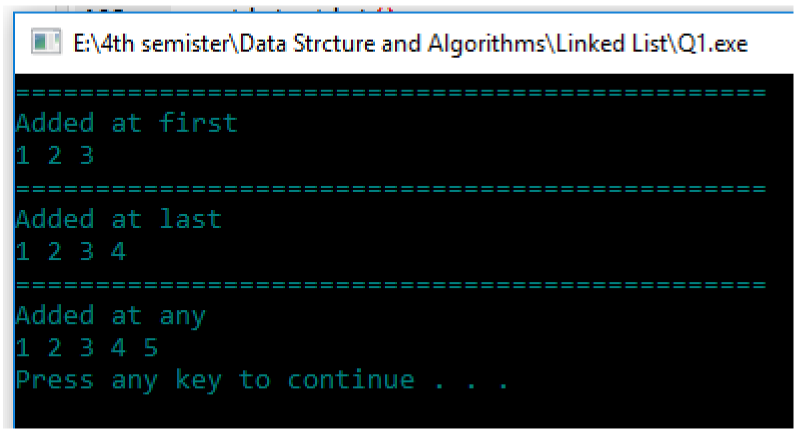


```cpp
        {
            if (tempNode->data == oldValue)
            {
                Node* newNode = createNode(newValue);

                newNode->next = tempNode->next;
                tempNode->next = newNode;
            }
        }
    }
}

//Display
void display()
{
    if (!isEmpty())
    {
        Node* tempNode = head;

        while (tempNode != NULL)
        {
            cout << tempNode->data << " ";
            tempNode = tempNode->next;
        }
        cout << endl;
    }
    else
    {
        cout << "List is Empty..." << endl;
    }
}
```



```cpp
    private:
    Node* head;
    //Creates a new node
    Node* createNode(int value)
    {
        Node* n = new Node;
        n->data = value;
        return n;
    }
};
    List::List()
{
    head = NULL;
}

List::~List()
{
    //deletes every node upon completion
    while (head != NULL)
    {
        Node* temp = head;
        head = head->next;

        delete temp;
    }
}
int main()
{
    List l;
    cout<<"================================================"<<endl;
    cout << "Added at first" << endl;
    l.insertFirst(3);
    l.insertFirst(2);
    l.insertFirst(1);
    l.display();

    cout<<"================================================"<<endl;
    cout << "Added at last" << endl;
    l.insertLast(4);
    l.display();

    cout<<"================================================"<<endl;
    cout << "Added at any" << endl;
    l.insertAny(4, 5);
    l.display();
    system("pause");
    return 0;
}
```

Output:

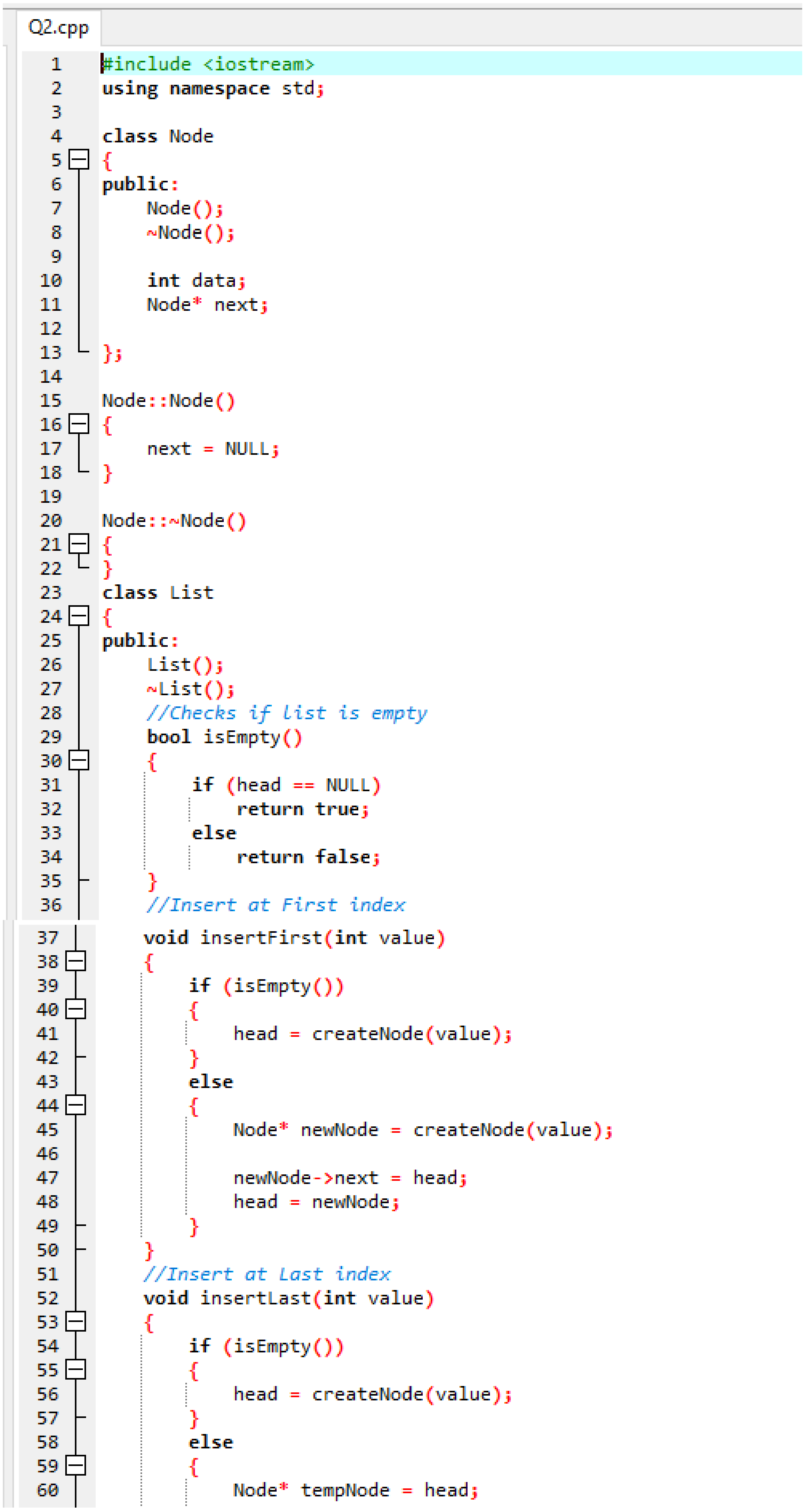


E:\4th semister\Data Strcture and Algorithms\Linked List\Q1.exe

```
===============================================
Added at first
1 2 3
===============================================
Added at last
1 2 3 4
===============================================
Added at any
1 2 3 4 5
Press any key to continue . . .
```

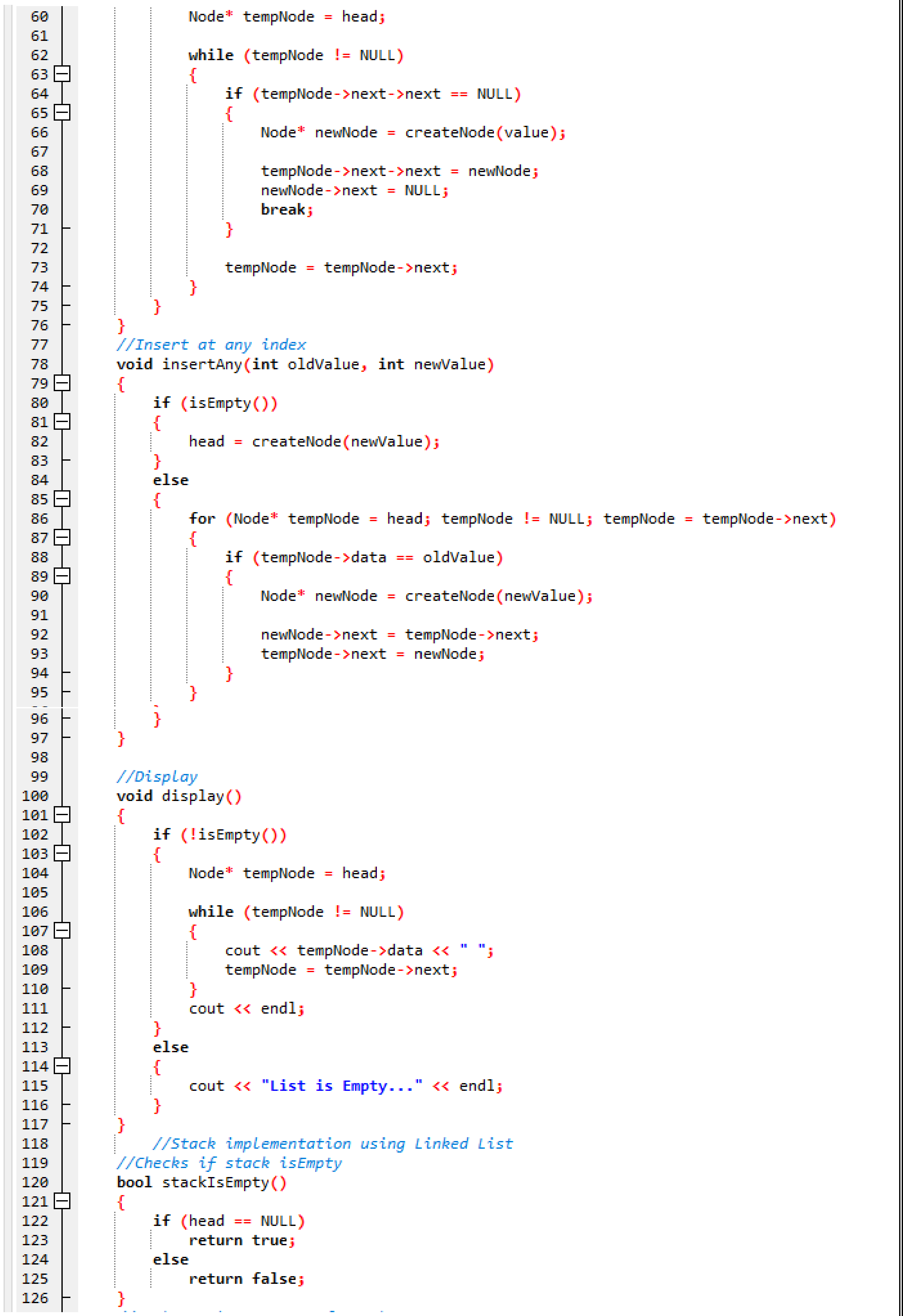
Task No. 2: Write a program to implement dynamic Stacks using linked list.

Coding:

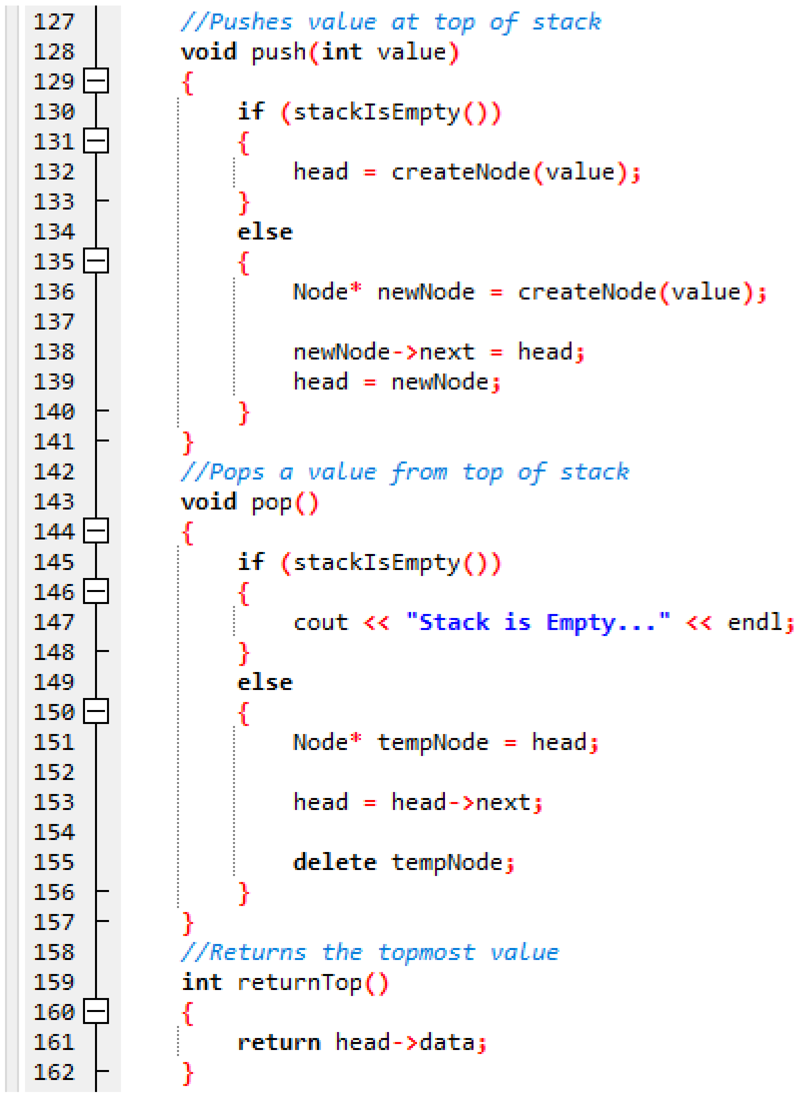


Q2.cpp

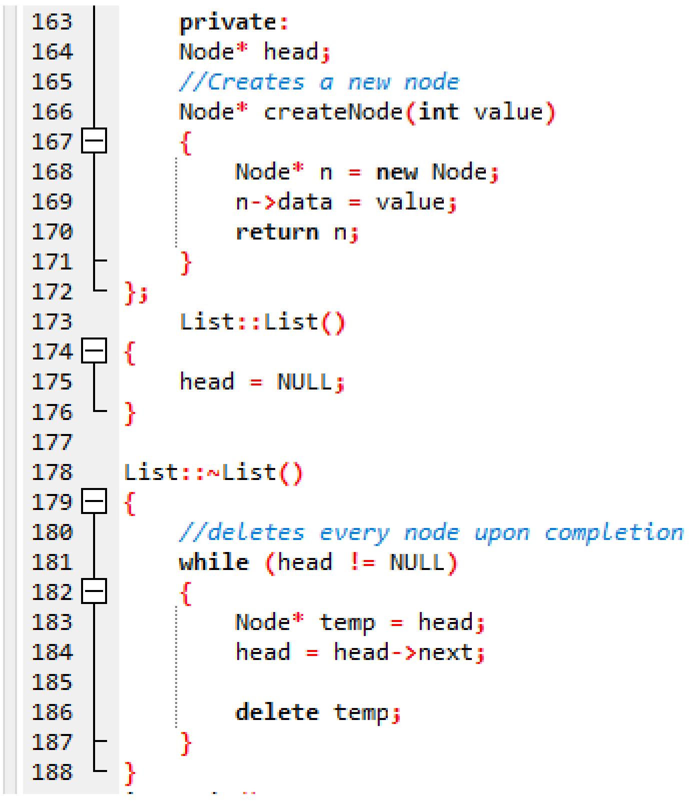
```
#include <iostream>
using namespace std;

class Node
{
public:
    Node();
    ~Node();

    int data;
    Node* next;

};

Node::Node()
{
    next = NULL;
}

Node::~Node()
{
}
class List
{
public:
    List();
    ~List();
    //Checks if list is empty
    bool isEmpty()
    {
        if (head == NULL)
            return true;
        else
            return false;
    }
    //Insert at First index
    void insertFirst(int value)
    {
        if (isEmpty())
        {
            head = createNode(value);
        }
        else
        {
            Node* newNode = createNode(value);

            newNode->next = head;
            head = newNode;
        }
    }
    //Insert at Last index
    void insertLast(int value)
    {
        if (isEmpty())
        {
            head = createNode(value);
        }
        else
        {
            Node* tempNode = head;
```



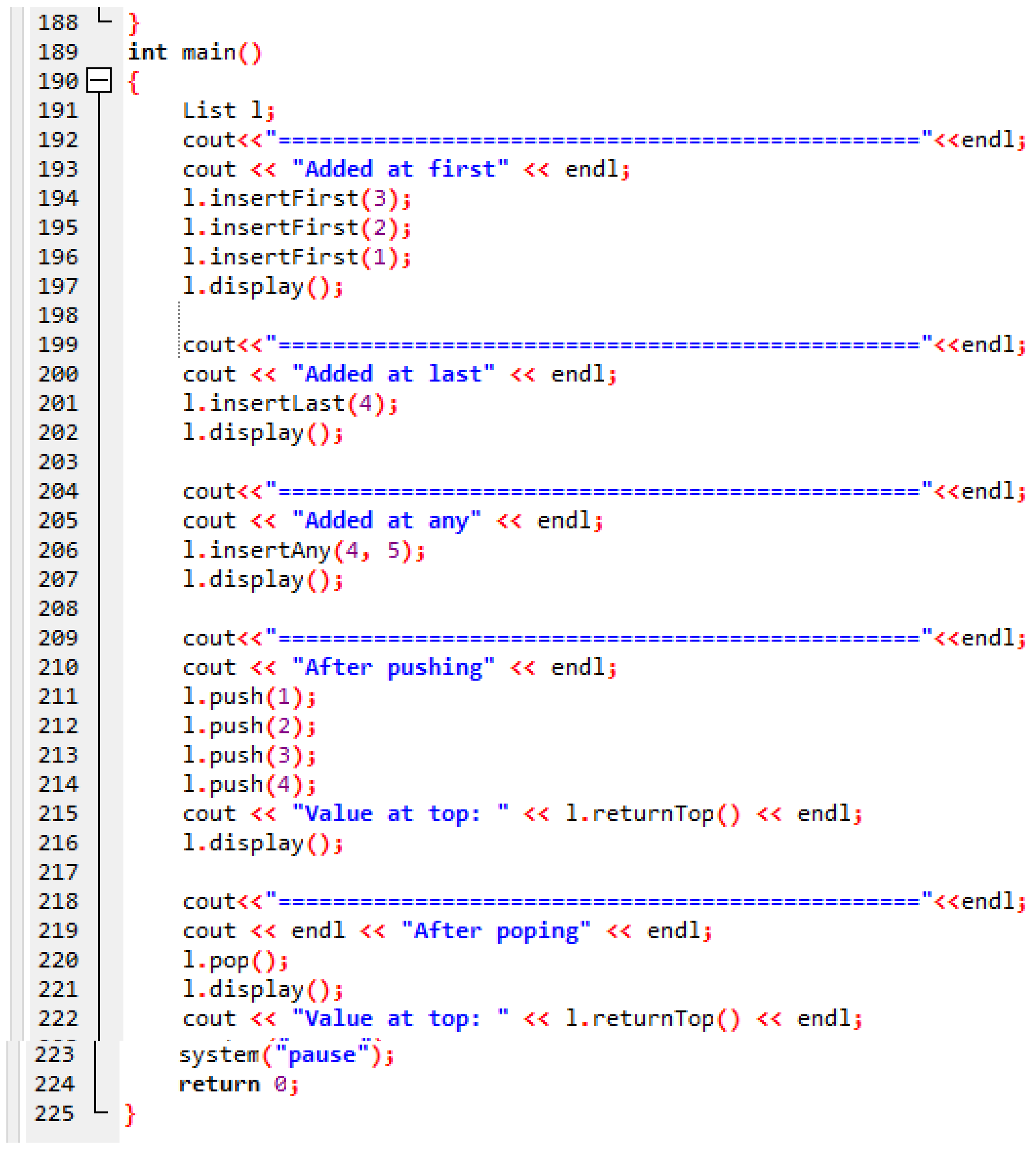
```cpp
        Node* tempNode = head;

        while (tempNode != NULL)
        {
            if (tempNode->next->next == NULL)
            {
                Node* newNode = createNode(value);

                tempNode->next->next = newNode;
                newNode->next = NULL;
                break;
            }

            tempNode = tempNode->next;
        }
    }
}
//Insert at any index
void insertAny(int oldValue, int newValue)
{
    if (isEmpty())
    {
        head = createNode(newValue);
    }
    else
    {
        for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
        {
            if (tempNode->data == oldValue)
            {
                Node* newNode = createNode(newValue);

                newNode->next = tempNode->next;
                tempNode->next = newNode;
            }
        }
    }
}

//Display
void display()
{
    if (!isEmpty())
    {
        Node* tempNode = head;

        while (tempNode != NULL)
        {
            cout << tempNode->data << " ";
            tempNode = tempNode->next;
        }
        cout << endl;
    }
    else
    {
        cout << "List is Empty..." << endl;
    }
}
    //Stack implementation using Linked List
//Checks if stack isEmpty
bool stackIsEmpty()
{
    if (head == NULL)
        return true;
    else
        return false;
}
```



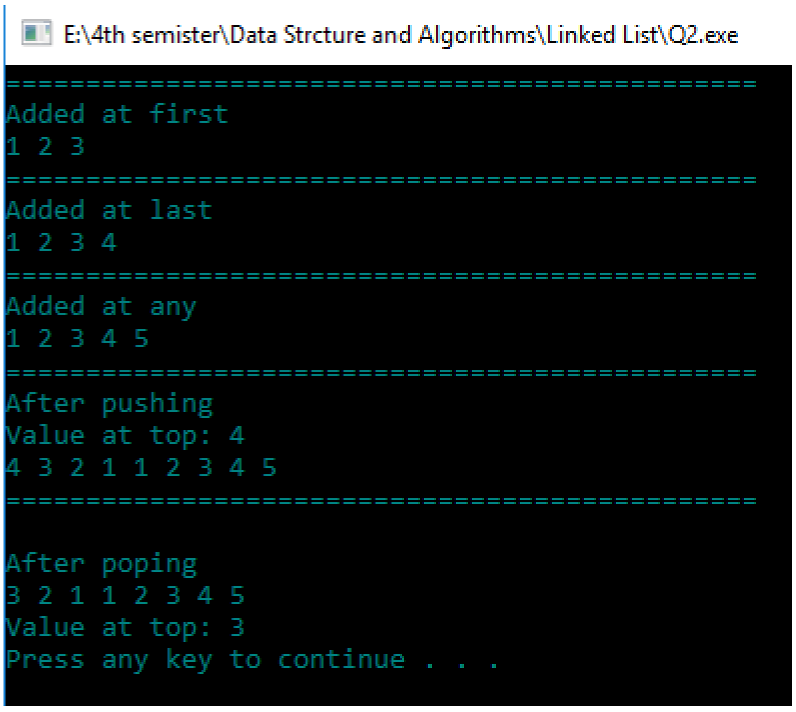
```cpp
    //Pushes value at top of stack
    void push(int value)
    {
        if (stackIsEmpty())
        {
            head = createNode(value);
        }
        else
        {
            Node* newNode = createNode(value);

            newNode->next = head;
            head = newNode;
        }
    }
    //Pops a value from top of stack
    void pop()
    {
        if (stackIsEmpty())
        {
            cout << "Stack is Empty..." << endl;
        }
        else
        {
            Node* tempNode = head;

            head = head->next;

            delete tempNode;
        }
    }
    //Returns the topmost value
    int returnTop()
    {
        return head->data;
    }
```



```cpp
    private:
    Node* head;
    //Creates a new node
    Node* createNode(int value)
    {
        Node* n = new Node;
        n->data = value;
        return n;
    }
};
    List::List()
{
    head = NULL;
}

List::~List()
{
    //deletes every node upon completion
    while (head != NULL)
    {
        Node* temp = head;
        head = head->next;

        delete temp;
    }
}
```


```cpp
}
int main()
{
    List l;
    cout<<"================================================"<<endl;
    cout << "Added at first" << endl;
    l.insertFirst(3);
    l.insertFirst(2);
    l.insertFirst(1);
    l.display();

    cout<<"================================================"<<endl;
    cout << "Added at last" << endl;
    l.insertLast(4);
    l.display();

    cout<<"================================================"<<endl;
    cout << "Added at any" << endl;
    l.insertAny(4, 5);
    l.display();

    cout<<"================================================"<<endl;
    cout << "After pushing" << endl;
    l.push(1);
    l.push(2);
    l.push(3);
    l.push(4);
    cout << "Value at top: " << l.returnTop() << endl;
    l.display();

    cout<<"================================================"<<endl;
    cout << endl << "After poping" << endl;
    l.pop();
    l.display();
    cout << "Value at top: " << l.returnTop() << endl;
    system("pause");
    return 0;
}
```

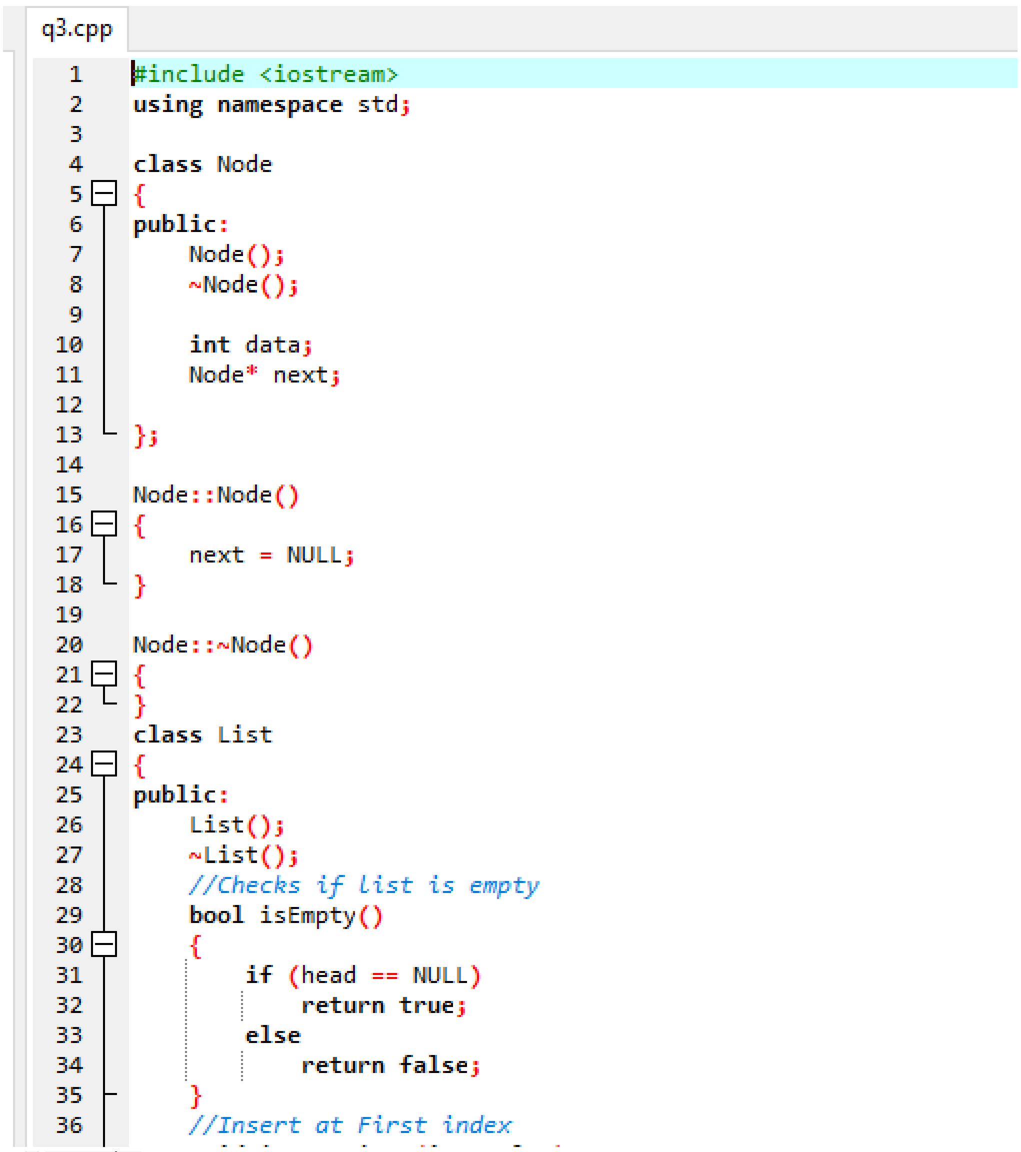
Output:


E:\4th semister\Data Strcture and Algorithms\Linked List\Q2.exe

```
================================================
Added at first
1 2 3
================================================
Added at last
1 2 3 4
================================================
Added at any
1 2 3 4 5
================================================
After pushing
Value at top: 4
4 3 2 1 1 2 3 4 5
================================================

After poping
3 2 1 1 2 3 4 5
Value at top: 3
Press any key to continue . . .
```

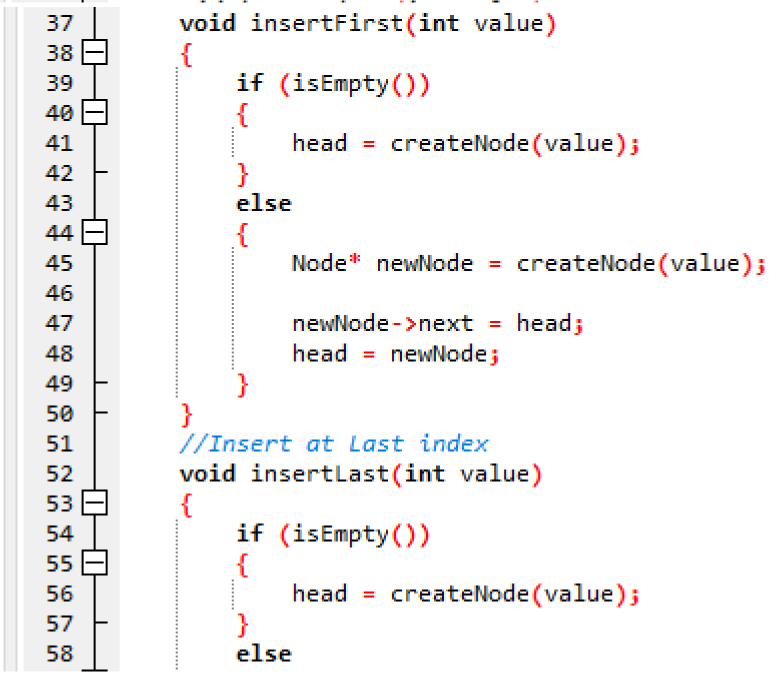
Task No. 3: Write a program to implement dynamic queue by using linked list.

Coding:

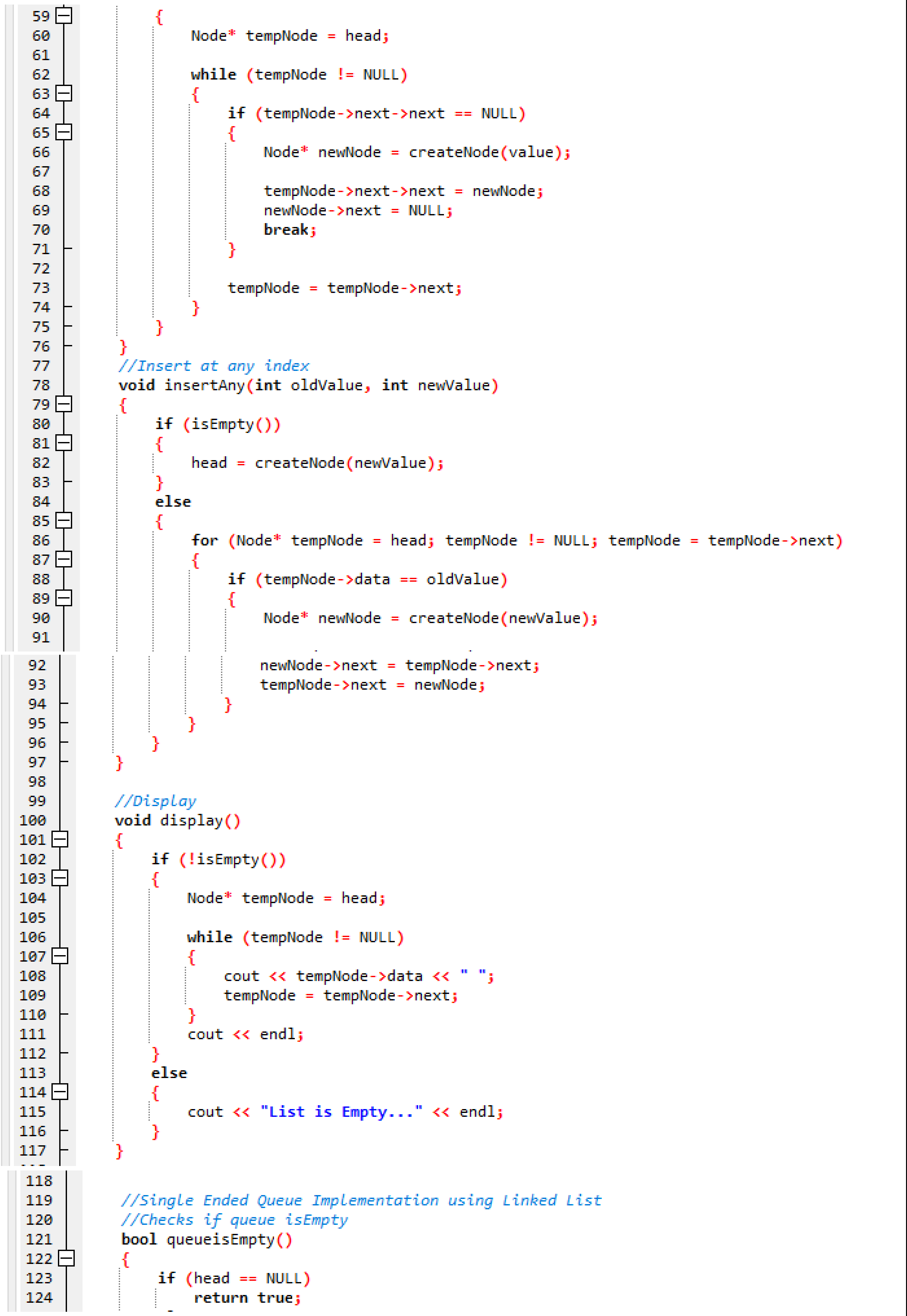


q3.cpp

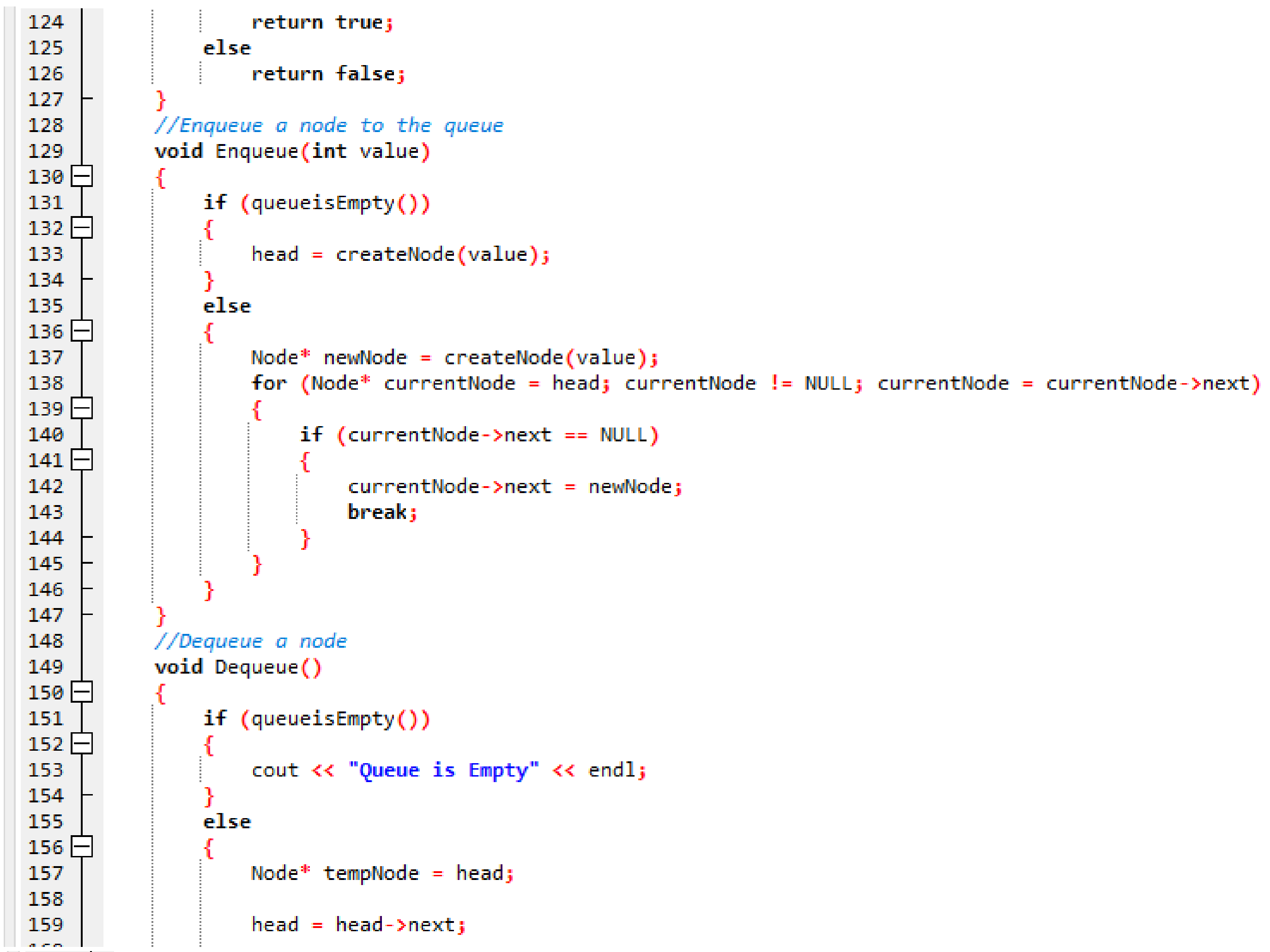
```cpp
#include <iostream>
using namespace std;

class Node
{
public:
    Node();
    ~Node();

    int data;
    Node* next;

};

Node::Node()
{
    next = NULL;
}

Node::~Node()
{
}
class List
{
public:
    List();
    ~List();
    //Checks if List is empty
    bool isEmpty()
    {
        if (head == NULL)
            return true;
        else
            return false;
    }
    //Insert at First index
```



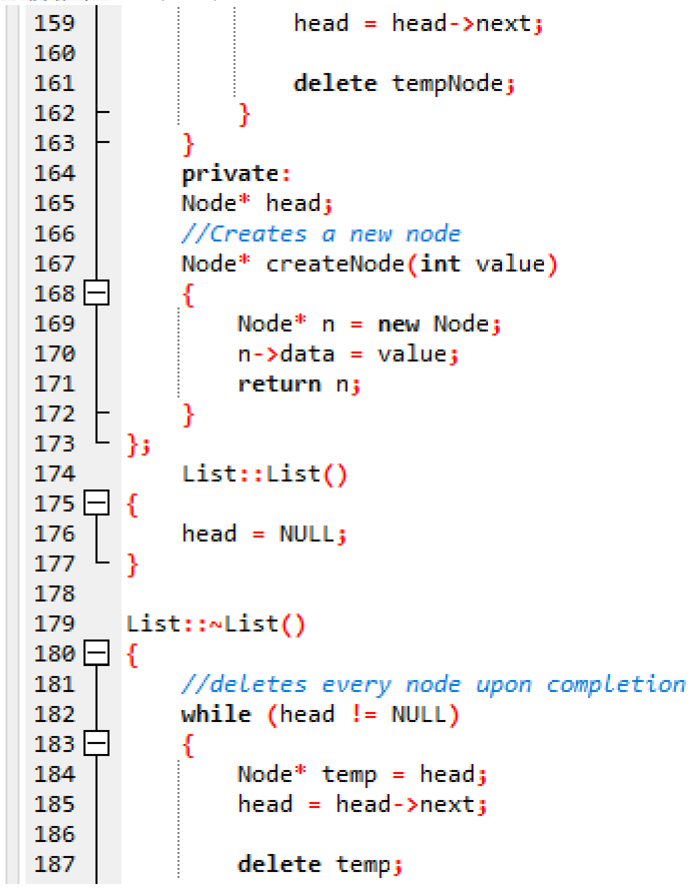
```cpp
    void insertFirst(int value)
    {
        if (isEmpty())
        {
            head = createNode(value);
        }
        else
        {
            Node* newNode = createNode(value);

            newNode->next = head;
            head = newNode;
        }
    }
    //Insert at Last index
    void insertLast(int value)
    {
        if (isEmpty())
        {
            head = createNode(value);
        }
        else
```



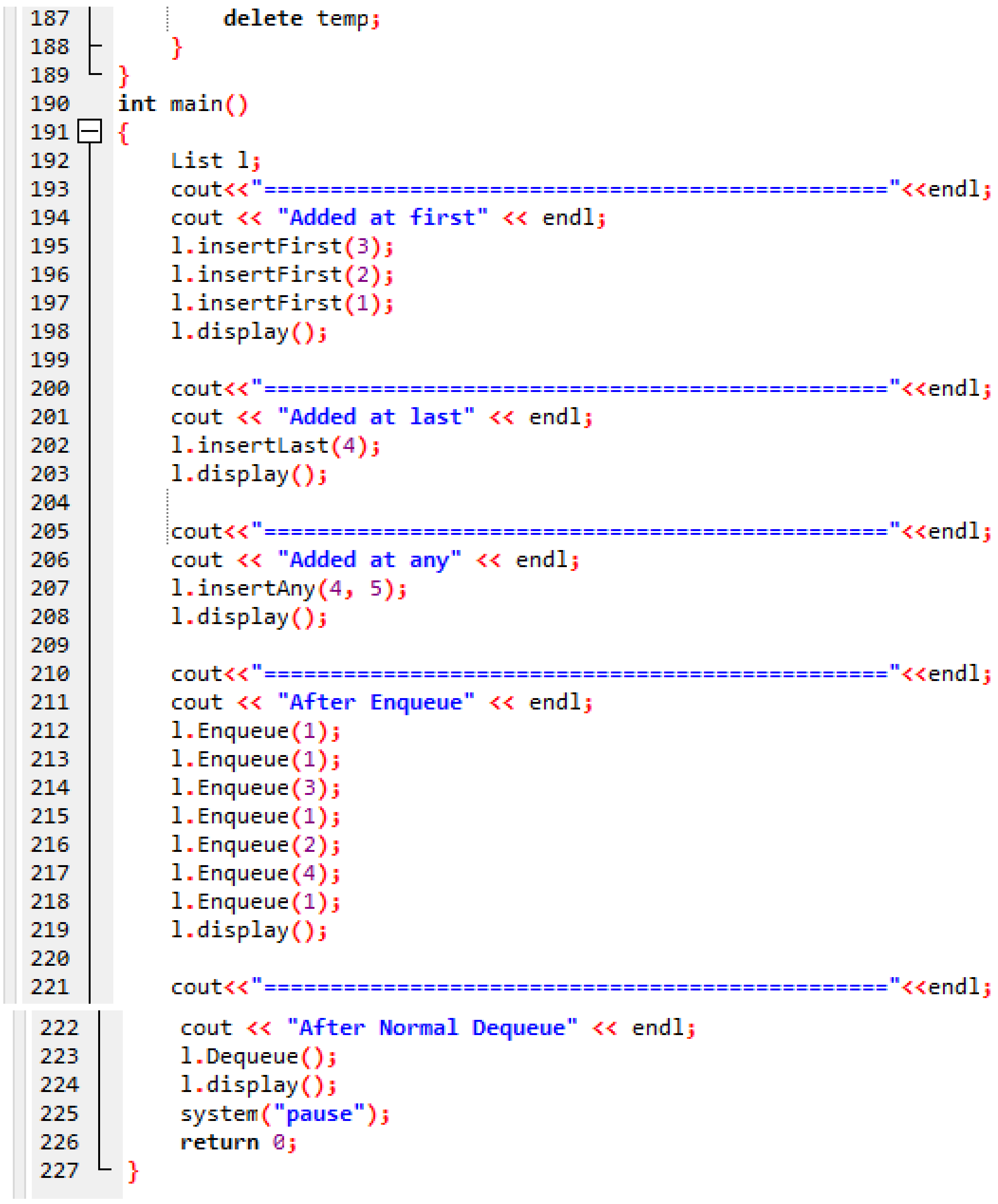
```cpp
        {
            Node* tempNode = head;

            while (tempNode != NULL)
            {
                if (tempNode->next->next == NULL)
                {
                    Node* newNode = createNode(value);

                    tempNode->next->next = newNode;
                    newNode->next = NULL;
                    break;
                }

                tempNode = tempNode->next;
            }
        }
    }
    //Insert at any index
    void insertAny(int oldValue, int newValue)
    {
        if (isEmpty())
        {
            head = createNode(newValue);
        }
        else
        {
            for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
            {
                if (tempNode->data == oldValue)
                {
                    Node* newNode = createNode(newValue);

                    newNode->next = tempNode->next;
                    tempNode->next = newNode;
                }
            }
        }
    }

    //Display
    void display()
    {
        if (!isEmpty())
        {
            Node* tempNode = head;

            while (tempNode != NULL)
            {
                cout << tempNode->data << " ";
                tempNode = tempNode->next;
            }
            cout << endl;
        }
        else
        {
            cout << "List is Empty..." << endl;
        }
    }


    //Single Ended Queue Implementation using Linked List
    //Checks if queue isEmpty
    bool queueisEmpty()
    {
        if (head == NULL)
            return true;
```



```cpp
            return true;
        else
            return false;
    }
    //Enqueue a node to the queue
    void Enqueue(int value)
    {
        if (queueisEmpty())
        {
            head = createNode(value);
        }
        else
        {
            Node* newNode = createNode(value);
            for (Node* currentNode = head; currentNode != NULL; currentNode = currentNode->next)
            {
                if (currentNode->next == NULL)
                {
                    currentNode->next = newNode;
                    break;
                }
            }
        }
    }
    //Dequeue a node
    void Dequeue()
    {
        if (queueisEmpty())
        {
            cout << "Queue is Empty" << endl;
        }
        else
        {
            Node* tempNode = head;

            head = head->next;
```

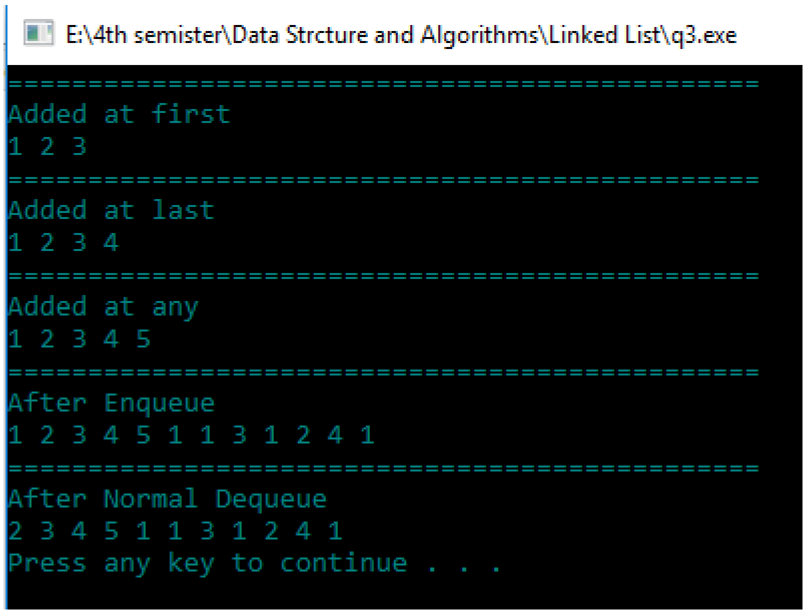


```cpp
            head = head->next;

            delete tempNode;
        }
    }
    private:
    Node* head;
    //Creates a new node
    Node* createNode(int value)
    {
        Node* n = new Node;
        n->data = value;
        return n;
    }
};
    List::List()
{
    head = NULL;
}

List::~List()
{
    //deletes every node upon completion
    while (head != NULL)
    {
        Node* temp = head;
        head = head->next;

        delete temp;
```



```cpp
        delete temp;
    }
}
int main()
{
    List l;
    cout<<"==============================================="<<endl;
    cout << "Added at first" << endl;
    l.insertFirst(3);
    l.insertFirst(2);
    l.insertFirst(1);
    l.display();

    cout<<"==============================================="<<endl;
    cout << "Added at last" << endl;
    l.insertLast(4);
    l.display();

    cout<<"==============================================="<<endl;
    cout << "Added at any" << endl;
    l.insertAny(4, 5);
    l.display();

    cout<<"==============================================="<<endl;
    cout << "After Enqueue" << endl;
    l.Enqueue(1);
    l.Enqueue(1);
    l.Enqueue(3);
    l.Enqueue(1);
    l.Enqueue(2);
    l.Enqueue(4);
    l.Enqueue(1);
    l.display();

    cout<<"==============================================="<<endl;
    cout << "After Normal Dequeue" << endl;
    l.Dequeue();
    l.display();
    system("pause");
    return 0;
}
```

Output:



E:\4th semister\Data Strcture and Algorithms\Linked List\q3.exe

```
===============================================
Added at first
1 2 3
===============================================
Added at last
1 2 3 4
===============================================
Added at any
1 2 3 4 5
===============================================
After Enqueue
1 2 3 4 5 1 1 3 1 2 4 1
===============================================
After Normal Dequeue
2 3 4 5 1 1 3 1 2 4 1
Press any key to continue . . .
```