LAB 1 - Linked List

GitHub Profile -> Akash kafle
GitHub link -> github.com/Akash-kafle/DSA_Akash_kafle
***NOTE:**

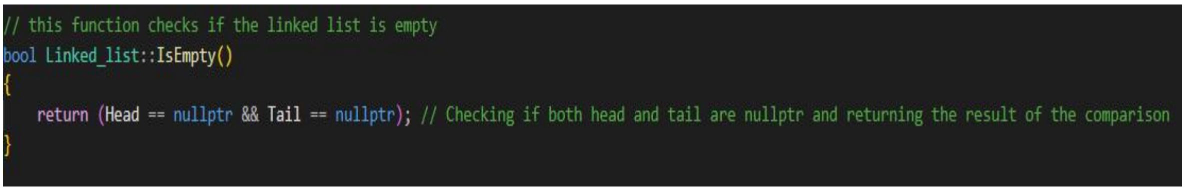
- The repository is private as stated.

Code implementation:

Functions:

(a) isEmpty():

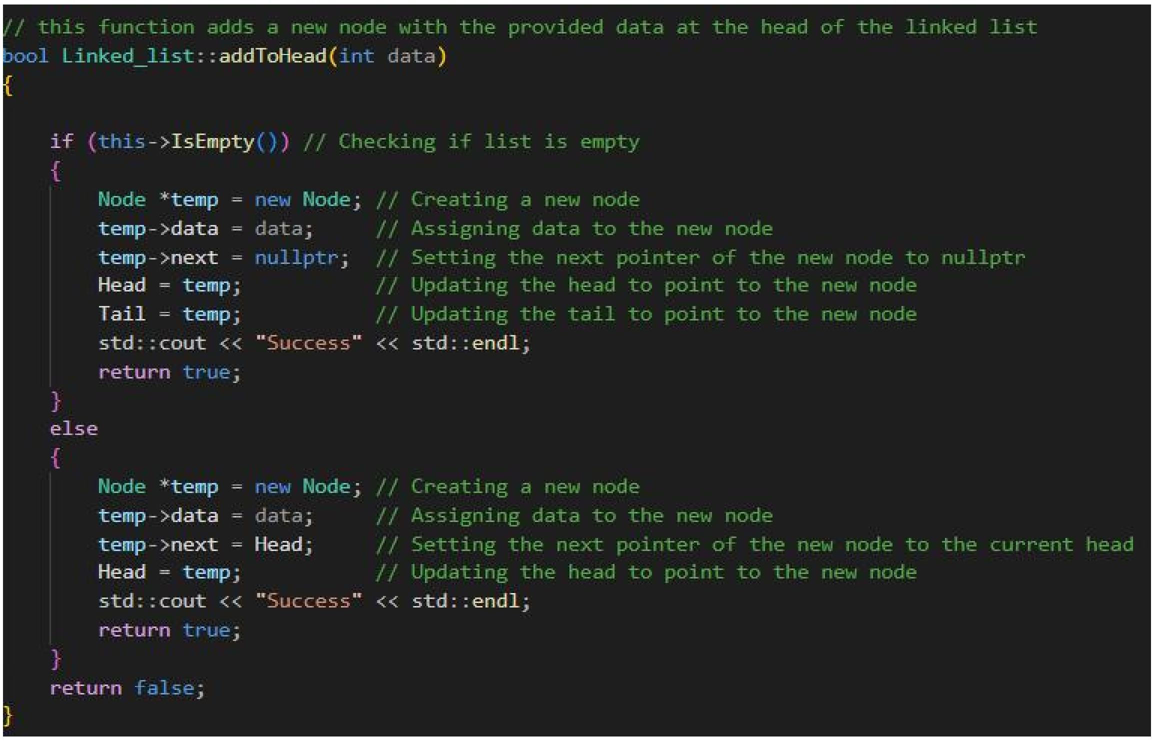
- Returns true if the list is empty, and false otherwise



```
// this function checks if the linked list is empty
bool Linked_list::IsEmpty()
{
    return (Head == nullptr && Tail == nullptr); // Checking if both head and tail are nullptr and returning the result of the comparison
}
```

(b) addToHead(data):

- Inserts an element to the beginning of the list

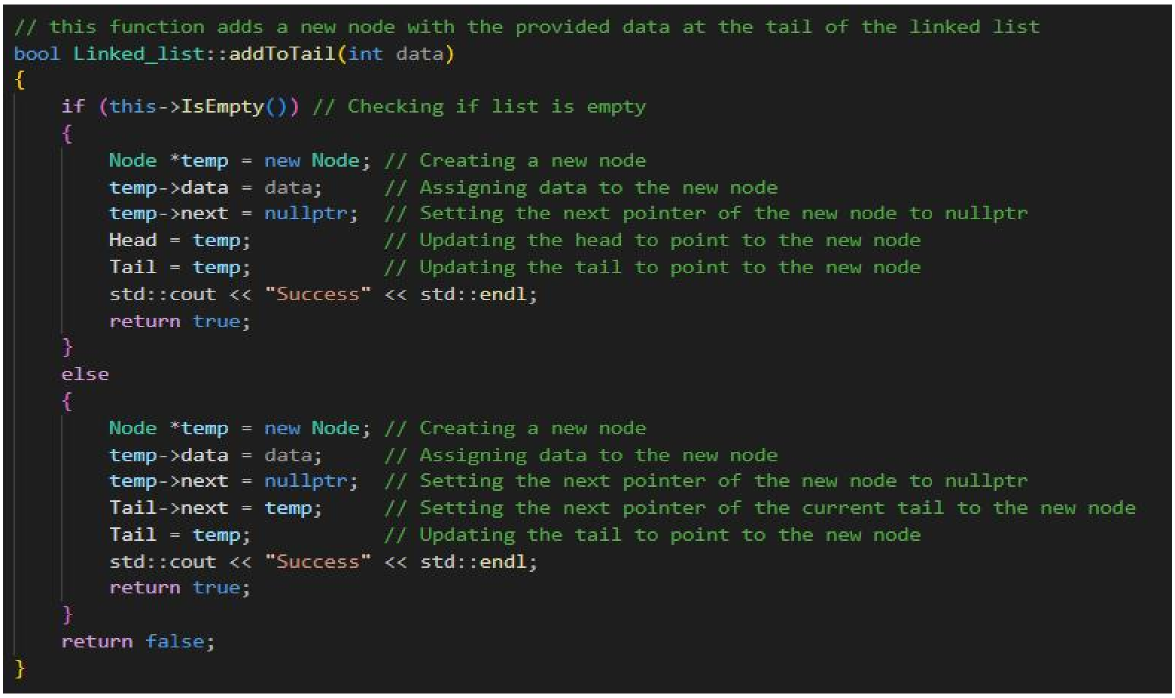


```
// this function adds a new node with the provided data at the head of the linked list
bool Linked_list::addToHead(int data)
{

    if (this->IsEmpty()) // Checking if list is empty
    {
        Node *temp = new Node; // Creating a new node
        temp->data = data;     // Assigning data to the new node
        temp->next = nullptr;  // Setting the next pointer of the new node to nullptr
        Head = temp;           // Updating the head to point to the new node
        Tail = temp;           // Updating the tail to point to the new node
        std::cout << "Success" << std::endl;
        return true;
    }
    else
    {
        Node *temp = new Node; // Creating a new node
        temp->data = data;     // Assigning data to the new node
        temp->next = Head;     // Setting the next pointer of the new node to the current head
        Head = temp;           // Updating the head to point to the new node
        std::cout << "Success" << std::endl;
        return true;
    }
    return false;
}
```

LAB 1 - Linked List

(c) addToTail(data):

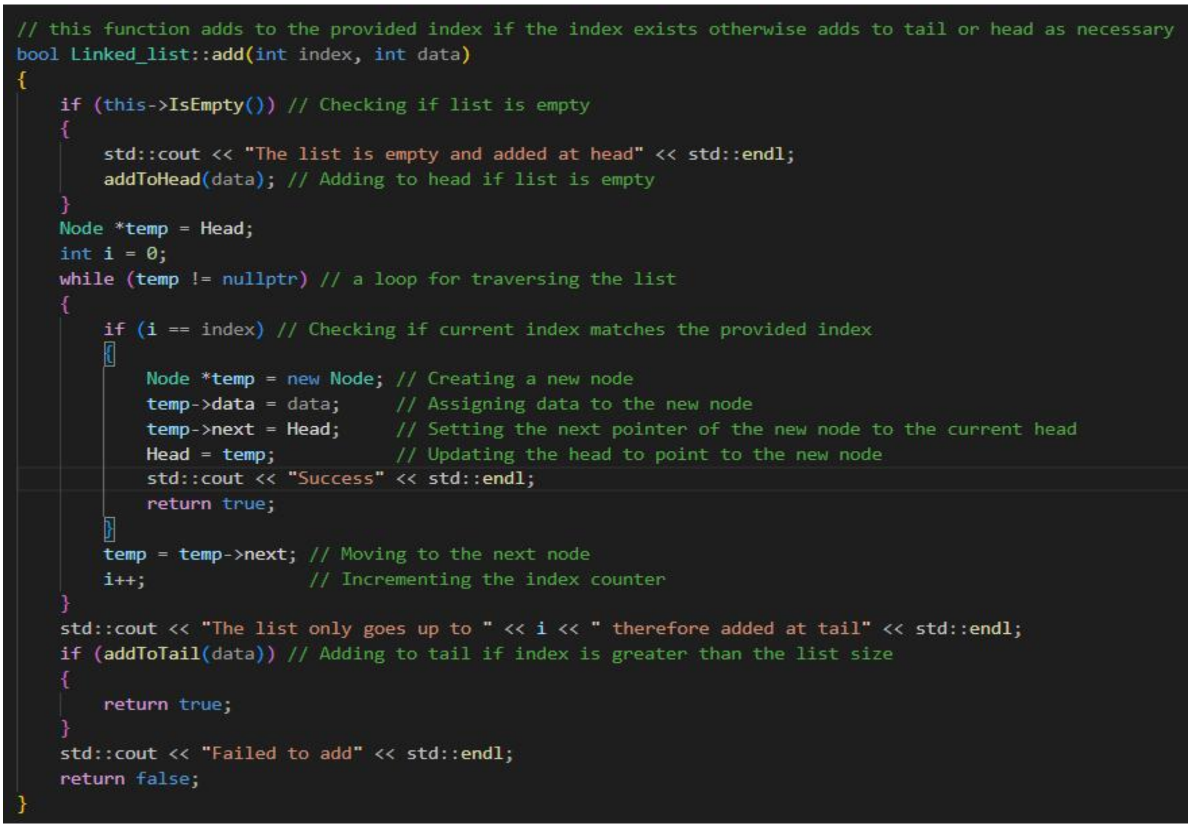
- Inserts an element to the end of the list



```cpp
// this function adds a new node with the provided data at the tail of the linked list
bool Linked_list::addToTail(int data)
{
    if (this->IsEmpty()) // Checking if list is empty
    {
        Node *temp = new Node; // Creating a new node
        temp->data = data;     // Assigning data to the new node
        temp->next = nullptr;  // Setting the next pointer of the new node to nullptr
        Head = temp;           // Updating the head to point to the new node
        Tail = temp;           // Updating the tail to point to the new node
        std::cout << "Success" << std::endl;
        return true;
    }
    else
    {
        Node *temp = new Node; // Creating a new node
        temp->data = data;     // Assigning data to the new node
        temp->next = nullptr;  // Setting the next pointer of the new node to nullptr
        Tail->next = temp;     // Setting the next pointer of the current tail to the new node
        Tail = temp;           // Updating the tail to point to the new node
        std::cout << "Success" << std::endl;
        return true;
    }
    return false;
}
```

(d) add(index, data):

- Inserts an element after the given index in the node and if the given index is not valid or has limits then it will be added to tail

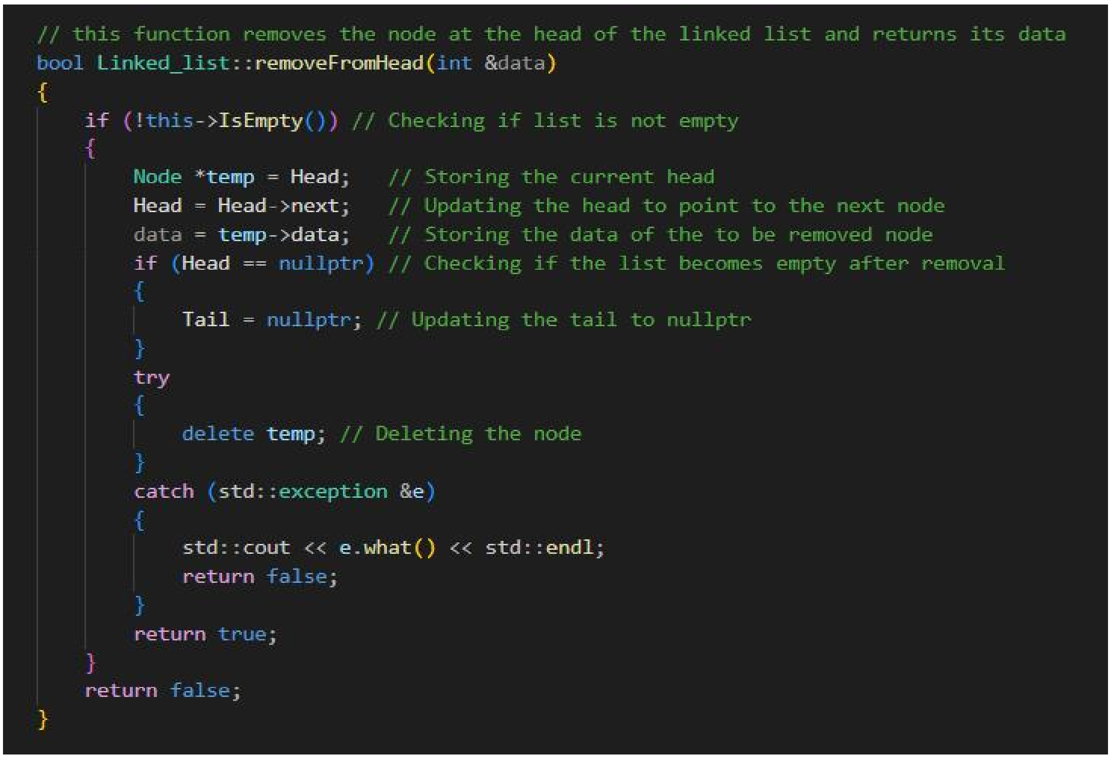


```cpp
// this function adds to the provided index if the index exists otherwise adds to tail or head as necessary
bool Linked_list::add(int index, int data)
{
    if (this->IsEmpty()) // Checking if list is empty
    {
        std::cout << "The list is empty and added at head" << std::endl;
        addToHead(data); // Adding to head if list is empty
    }
    Node *temp = Head;
    int i = 0;
    while (temp != nullptr) // a loop for traversing the list
    {
        if (i == index) // Checking if current index matches the provided index
        {
            Node *temp = new Node; // Creating a new node
            temp->data = data;     // Assigning data to the new node
            temp->next = Head;     // Setting the next pointer of the new node to the current head
            Head = temp;           // Updating the head to point to the new node
            std::cout << "Success" << std::endl;
            return true;
        }
        temp = temp->next; // Moving to the next node
        i++;               // Incrementing the index counter
    }
    std::cout << "The list only goes up to " << i << " therefore added at tail" << std::endl;
    if (addToTail(data)) // Adding to tail if index is greater than the list size
    {
        return true;
    }
    std::cout << "Failed to add" << std::endl;
    return false;
}
```

LAB 1 - Linked List

(e) removeFromHead(&data):

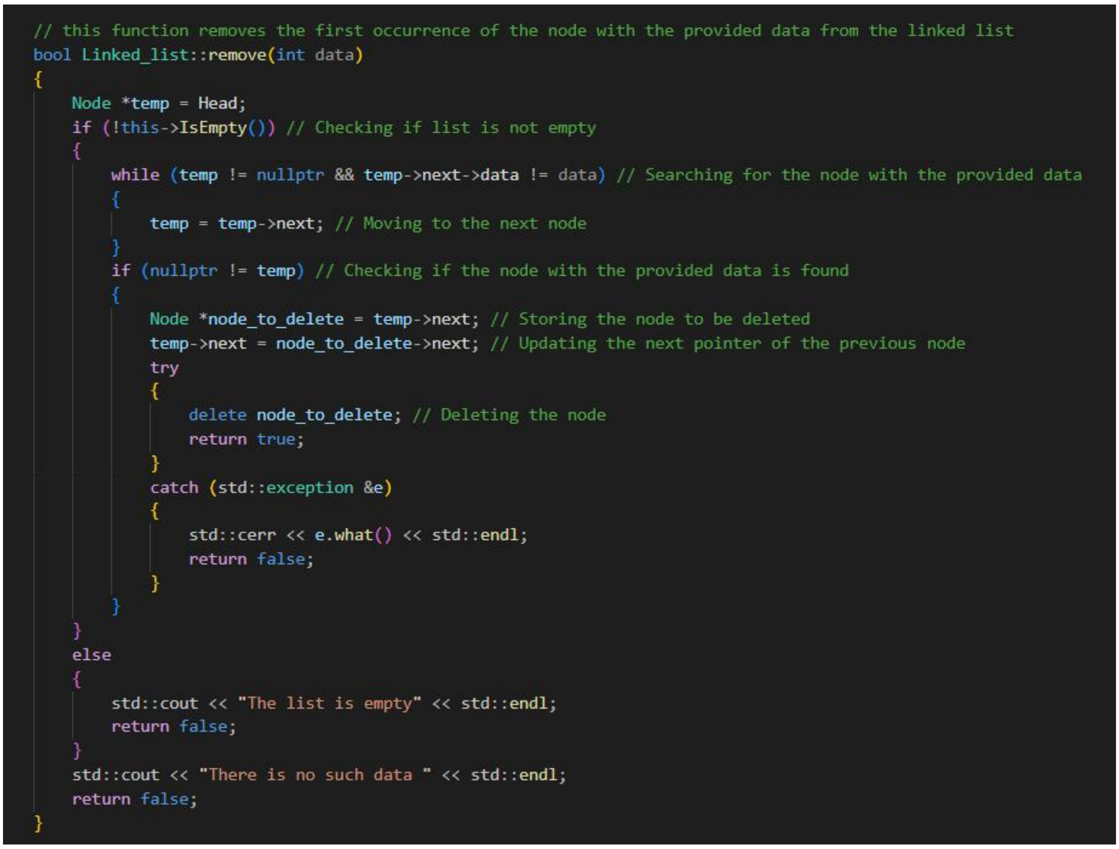
- Removes the first node in the list



```
// this function removes the node at the head of the linked list and returns its data
bool Linked_list::removeFromHead(int &data)
{
    if (!this->IsEmpty()) // Checking if list is not empty
    {
        Node *temp = Head;    // Storing the current head
        Head = Head->next;    // Updating the head to point to the next node
        data = temp->data;    // Storing the data of the to be removed node
        if (Head == nullptr) // Checking if the list becomes empty after removal
        {
            Tail = nullptr; // Updating the tail to nullptr
        }
        try
        {
            delete temp; // Deleting the node
        }
        catch (std::exception &e)
        {
            std::cout << e.what() << std::endl;
            return false;
        }
        return true;
    }
    return false;
}
```

(f) remove(data):

- Removes the node with the given data

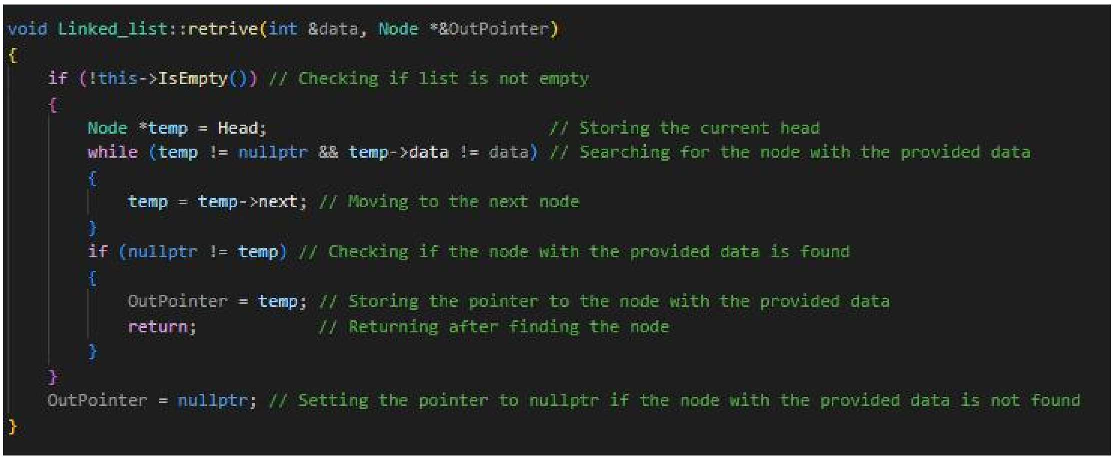


```
// this function removes the first occurrence of the node with the provided data from the linked list
bool Linked_list::remove(int data)
{
    Node *temp = Head;
    if (!this->IsEmpty()) // Checking if list is not empty
    {
        while (temp != nullptr && temp->next->data != data) // Searching for the node with the provided data
        {
            temp = temp->next; // Moving to the next node
        }
        if (nullptr != temp) // Checking if the node with the provided data is found
        {
            Node *node_to_delete = temp->next; // Storing the node to be deleted
            temp->next = node_to_delete->next; // Updating the next pointer of the previous node
            try
            {
                delete node_to_delete; // Deleting the node
                return true;
            }
            catch (std::exception &e)
            {
                std::cerr << e.what() << std::endl;
                return false;
            }
        }
    }
    else
    {
        std::cout << "The list is empty" << std::endl;
        return false;
    }
    std::cout << "There is no such data " << std::endl;
    return false;
}
```

LAB 1 - Linked List

(g) retrieve(data, outputNodePointer):

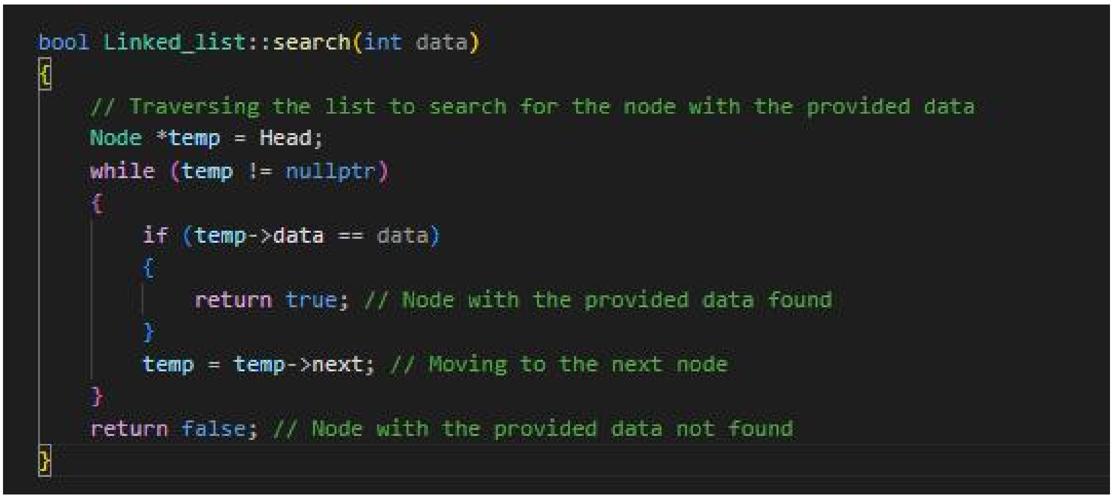
- Returns the pointer to the node with the requested data



```
void Linked_list::retrive(int &data, Node *&OutPointer)
{
    if (!this->IsEmpty()) // Checking if list is not empty
    {
        Node *temp = Head;                            // Storing the current head
        while (temp != nullptr && temp->data != data) // Searching for the node with the provided data
        {
            temp = temp->next; // Moving to the next node
        }
        if (nullptr != temp) // Checking if the node with the provided data is found
        {
            OutPointer = temp; // Storing the pointer to the node with the provided data
            return;            // Returning after finding the node
        }
    }
    OutPointer = nullptr; // Setting the pointer to nullptr if the node with the provided data is not found
}
```

(h) search(data):

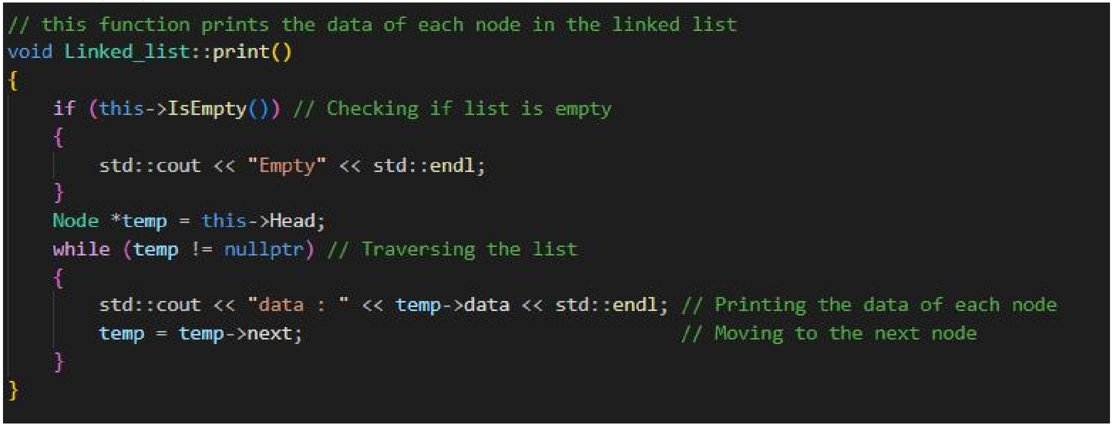
- Returns true if the data exists in the list, and false otherwise



```
bool Linked_list::search(int data)
{
    // Traversing the list to search for the node with the provided data
    Node *temp = Head;
    while (temp != nullptr)
    {
        if (temp->data == data)
        {
            return true; // Node with the provided data found
        }
        temp = temp->next; // Moving to the next node
    }
    return false; // Node with the provided data not found
}
```

(i) traverse():

- Displays the contents of the list



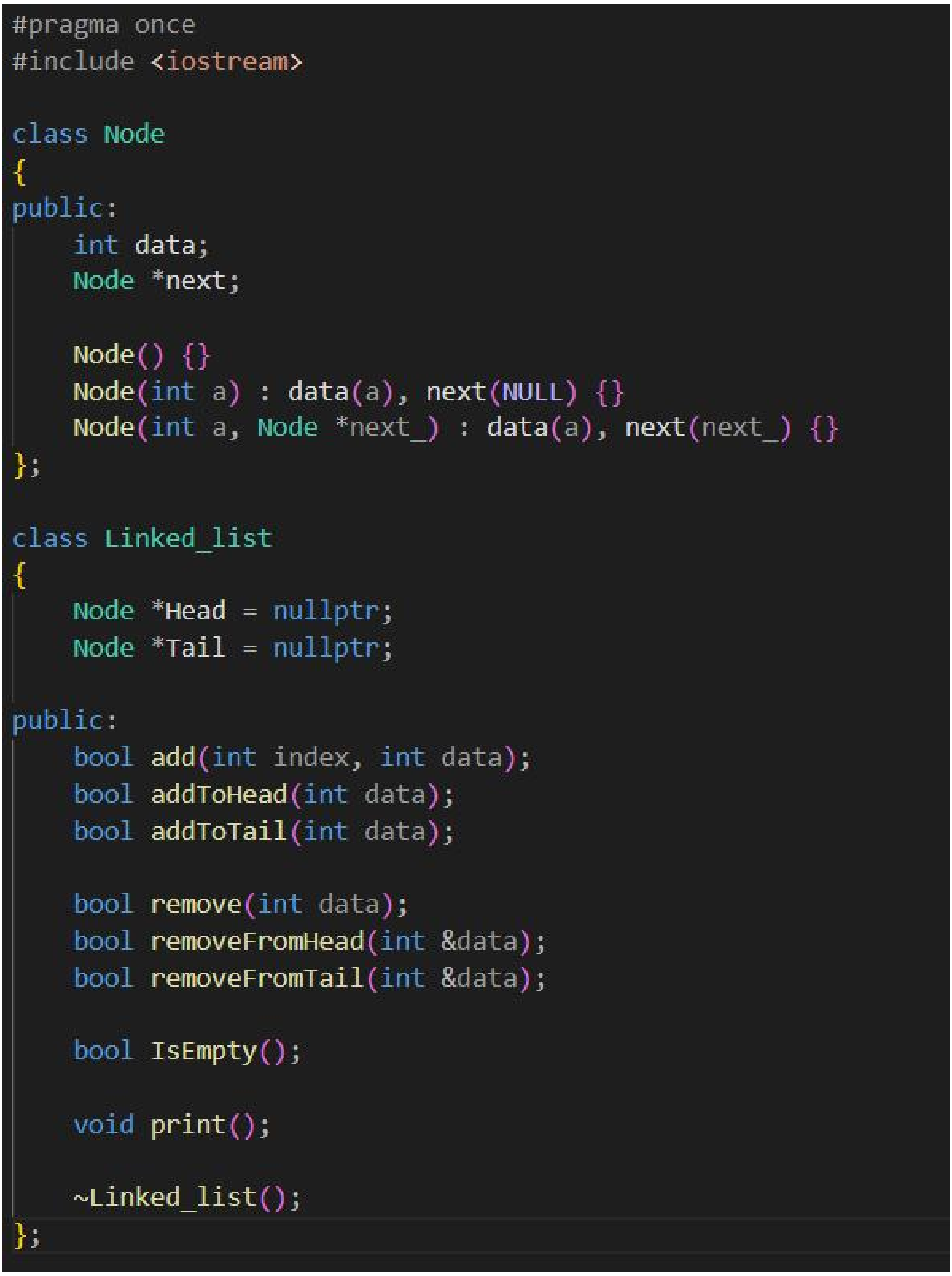
```
// this function prints the data of each node in the linked list
void Linked_list::print()
{
    if (this->IsEmpty()) // Checking if list is empty
    {
        std::cout << "Empty" << std::endl;
    }
    Node *temp = this->Head;
    while (temp != nullptr) // Traversing the list
    {
        std::cout << "data : " << temp->data << std::endl; // Printing the data of each node
        temp = temp->next;                                 // Moving to the next node
    }
}
```

LAB 1 - Linked List

# Main Function and it’s output:

Header File:

- I have used the following class for the above mentioned linked list functions.They are separated in a separate file called ‘function.cpp’ which can be found in the GitHub.

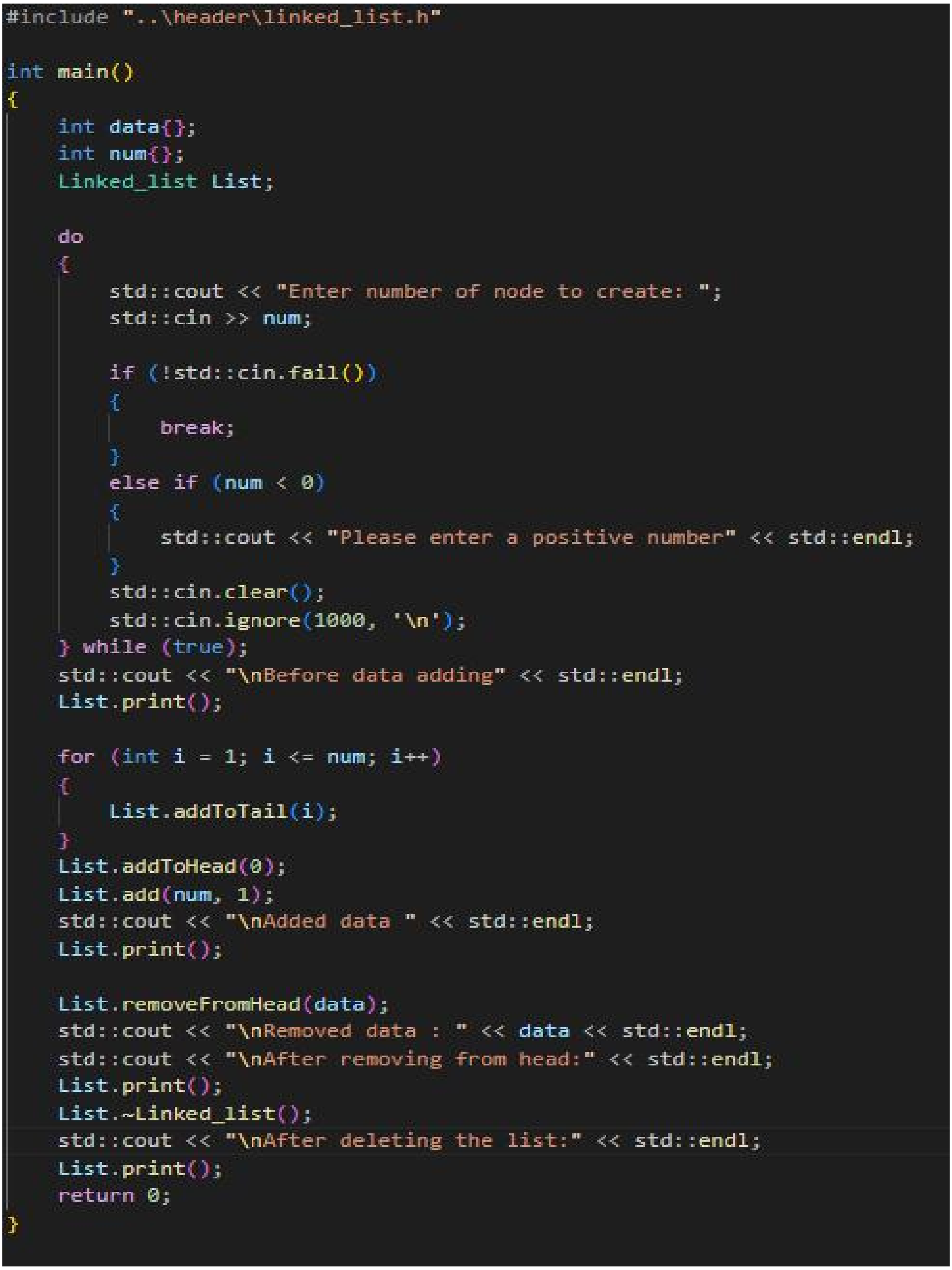


```
#pragma once
#include <iostream>

class Node
{
public:
    int data;
    Node *next;

    Node() {}
    Node(int a) : data(a), next(NULL) {}
    Node(int a, Node *next_) : data(a), next(next_) {}
};

class Linked_list
{
    Node *Head = nullptr;
    Node *Tail = nullptr;

public:
    bool add(int index, int data);
    bool addToHead(int data);
    bool addToTail(int data);

    bool remove(int data);
    bool removeFromHead(int &data);
    bool removeFromTail(int &data);

    bool IsEmpty();

    void print();

    ~Linked_list();
};
```

LAB 1 - Linked List

Main Function:

- Here, I have made sure to implement as much possible and made sure the program doesn't crashes if the user mistakenly enters 'char' or 'string' literals in place of integers.
- I have also made it free to just add and remove as the lecturer wish and check for any loop holes in the code.

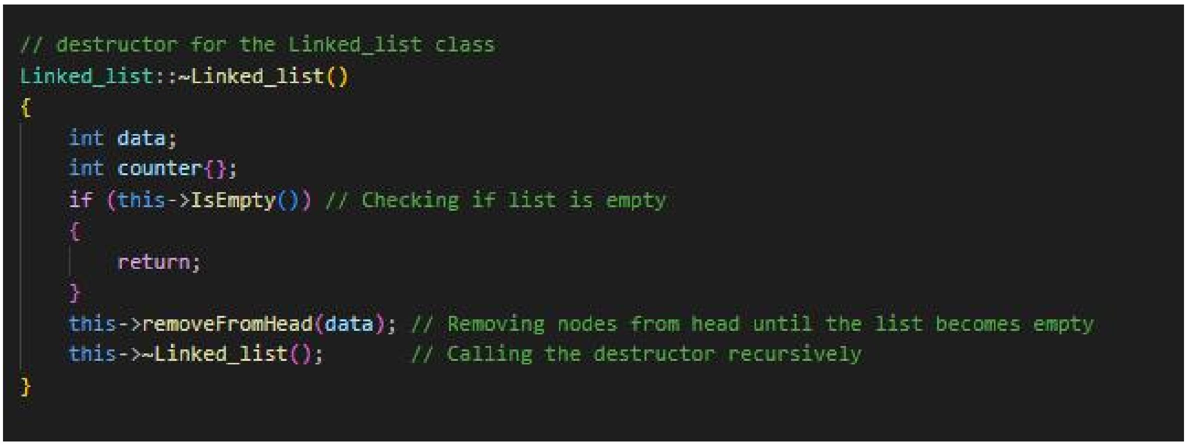


```cpp
#include "..\header\linked_list.h"

int main()
{
    int data{};
    int num{};
    Linked_list List;

    do
    {
        std::cout << "Enter number of node to create: ";
        std::cin >> num;

        if (!std::cin.fail())
        {
            break;
        }
        else if (num < 0)
        {
            std::cout << "Please enter a positive number" << std::endl;
        }
        std::cin.clear();
        std::cin.ignore(1000, '\n');
    } while (true);
    std::cout << "\nBefore data adding" << std::endl;
    List.print();

    for (int i = 1; i <= num; i++)
    {
        List.addToTail(i);
    }
    List.addToHead(0);
    List.add(num, 1);
    std::cout << "\nAdded data " << std::endl;
    List.print();

    List.removeFromHead(data);
    std::cout << "\nRemoved data : " << data << std::endl;
    std::cout << "\nAfter removing from head:" << std::endl;
    List.print();
    List.~Linked_list();
    std::cout << "\nAfter deleting the list:" << std::endl;
    List.print();
    return 0;
}
```

LAB 1 - Linked List

Destructor function:

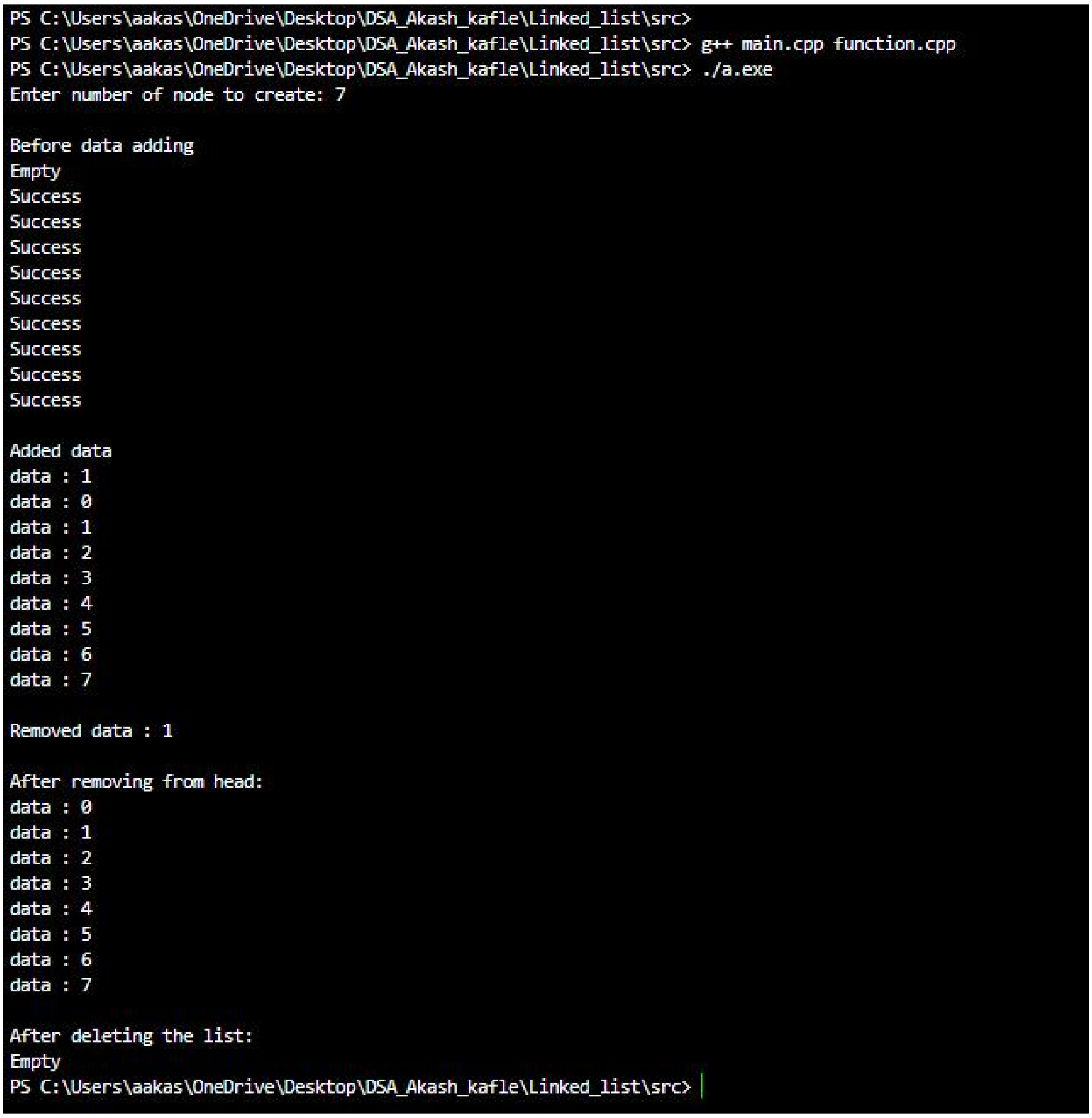
- I have used this destructor to delete the existing pointers and other values. As to not have any memory leaks in the code.



```
// destructor for the Linked_list class
Linked_list::~Linked_list()
{
    int data;
    int counter{};
    if (this->IsEmpty()) // Checking if list is empty
    {
        return;
    }
    this->removeFromHead(data); // Removing nodes from head until the list becomes empty
    this->~Linked_list();       // Calling the destructor recursively
}
```

Output:

This is the output of the code presented above.



```
PS C:\Users\aakas\OneDrive\Desktop\DSA_Akash_kafle\Linked_list\src>
PS C:\Users\aakas\OneDrive\Desktop\DSA_Akash_kafle\Linked_list\src> g++ main.cpp function.cpp
PS C:\Users\aakas\OneDrive\Desktop\DSA_Akash_kafle\Linked_list\src> ./a.exe
Enter number of node to create: 7

Before data adding
Empty
Success
Success
Success
Success
Success
Success
Success
Success
Success

Added data
data : 1
data : 0
data : 1
data : 2
data : 3
data : 4
data : 5
data : 6
data : 7

Removed data : 1

After removing from head:
data : 0
data : 1
data : 2
data : 3
data : 4
data : 5
data : 6
data : 7

After deleting the list:
Empty
PS C:\Users\aakas\OneDrive\Desktop\DSA_Akash_kafle\Linked_list\src>
```

LAB 1 - Linked List

# Final words:

- Any improvements and review of the code for better is highly appreciated.
- If I have missed few things please mentions that in our upcoming classes.I will do my best to improve those missing points in the coming future labs.
- I have provided all the information of my GitHub profile above, at the top of the file.