**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**



**A Lab Report**

**On**

**“Linked List”**

**[Code No.: COMP 202]**

**Submitted by**

**Hridayanshu Raj Acharya**

**UNG CE II-I**

**Roll no:1**

**Submitted to**

**Dr. Rajani Chulyadyo**

**Department of Computer Science and Engineering**

**Submission Date: 2024/05/06**

## Lab Report 1

### Linked List

Linked lists are linear data structures consisting of a sequence of elements called nodes. Unlike arrays, where elements are stored in contiguous memory locations, linked list elements are dynamically allocated and connected via pointers. This report aims to demonstrate the functionality and applications of linked lists through experiments and code demonstrations.

### Advantages of Linked Lists

* Linked lists allows for dynamic memory allocation, meaning that memory can be allocated and deallocated as needed during program execution.
* Efficient Insertion and deletion of nodes in a linked list.
* Linked list don’t suffer from the overhead of fixed size allocation like arrays.
* Linked lists can grow or shrink dynamically without need for resizing operations.
* Linked lists can be used to implement various other data structures and algorithms, like stack, queues etc.

### Operations Performed in a Linked Lists

The operations that are performed in the implementation of linked lists in c++ are as follows:-

1. isEmpty
2. addToHead
3. addToTail
4. addAfter
5. removeFromHead
6. removeFromTail
7. remove
8. search
9. retrieve
10. traverse

### Github Link:

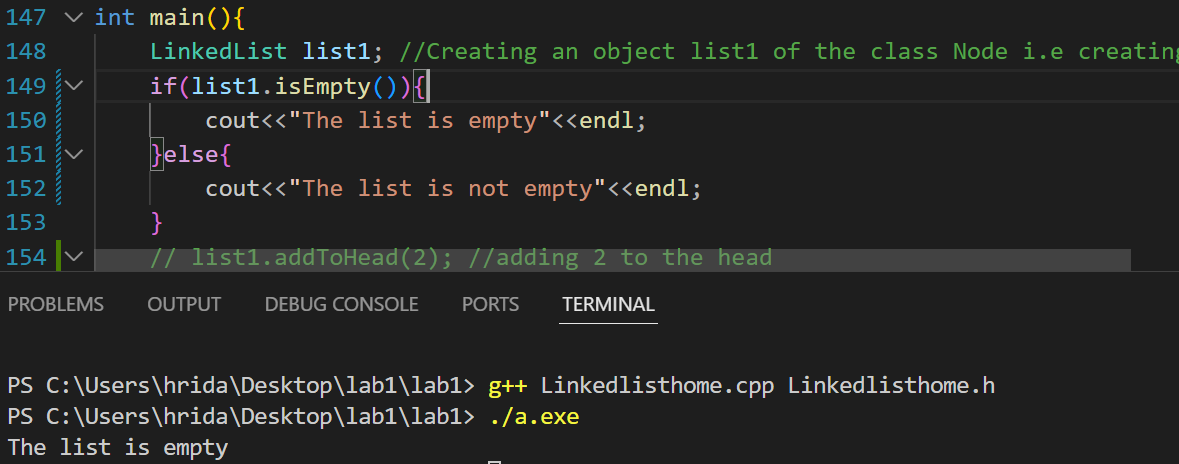
You can clone the following git repository and run the code:

<https://github.com/hridayanshu236/lab1>

### **Outputs**:

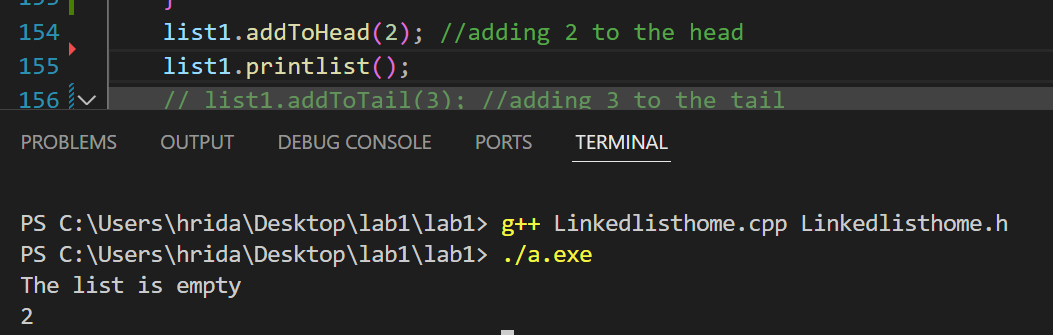
The outputs for each operation performed on the linked list are displayed below.

1. **isEmpty():**

****

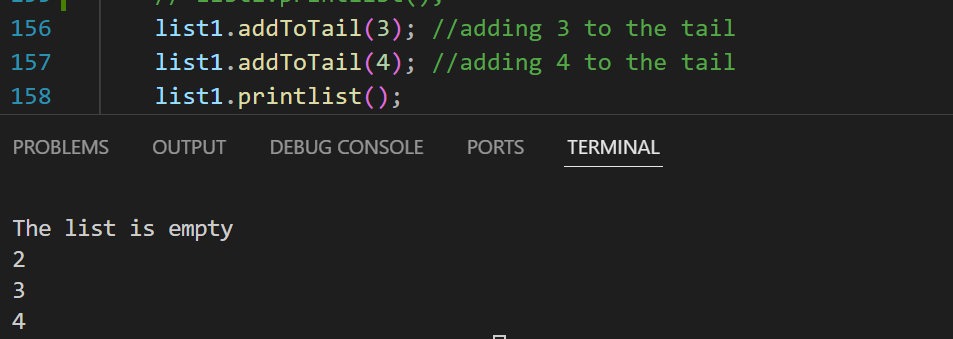
The isEmpty() function checks if the created linked list is empty or not. The function returns true value if the list is empty and returns false if it is not empty.

1. **addToHead(data):**



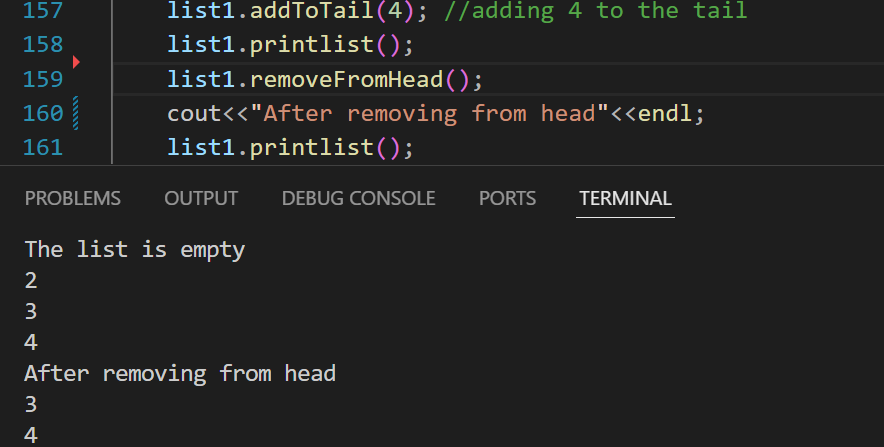
The addToHead(data) takes any input data from the user and adds the data in a new node in head.

1. **addToTail(data):**



The addToTail(data) takes any input data from the user and adds the data in a new node in Tail.

1. **removeFromHead():**



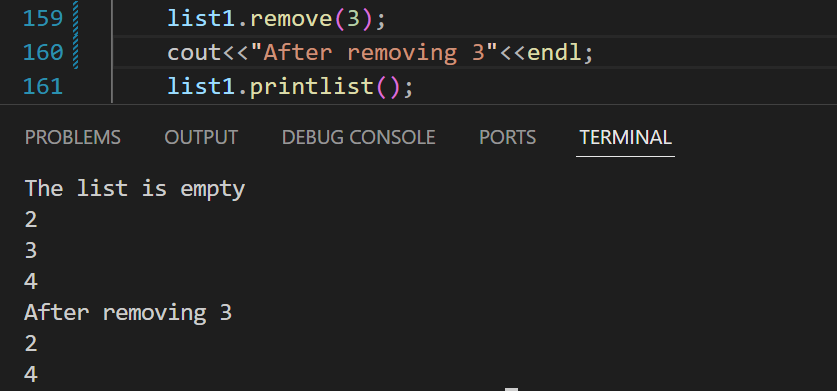
The removeFromHead() function lets user to remove the node at the beginning i.e Head.

1. **removeFromTail():**



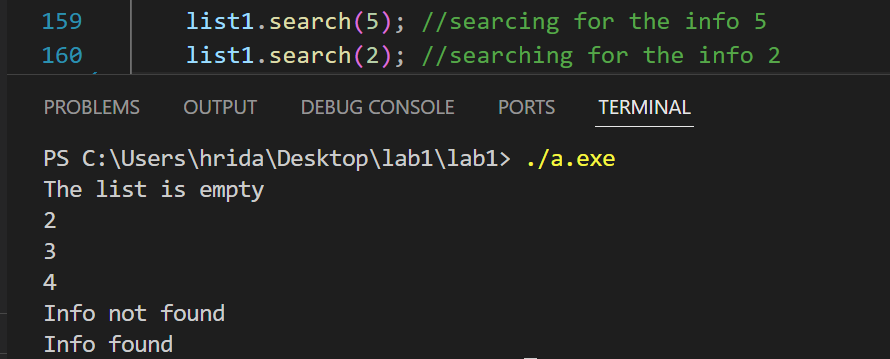
The removeFromTail() function lets user to remove the node at the last i.e Tail.

1. **remove(int):**

****

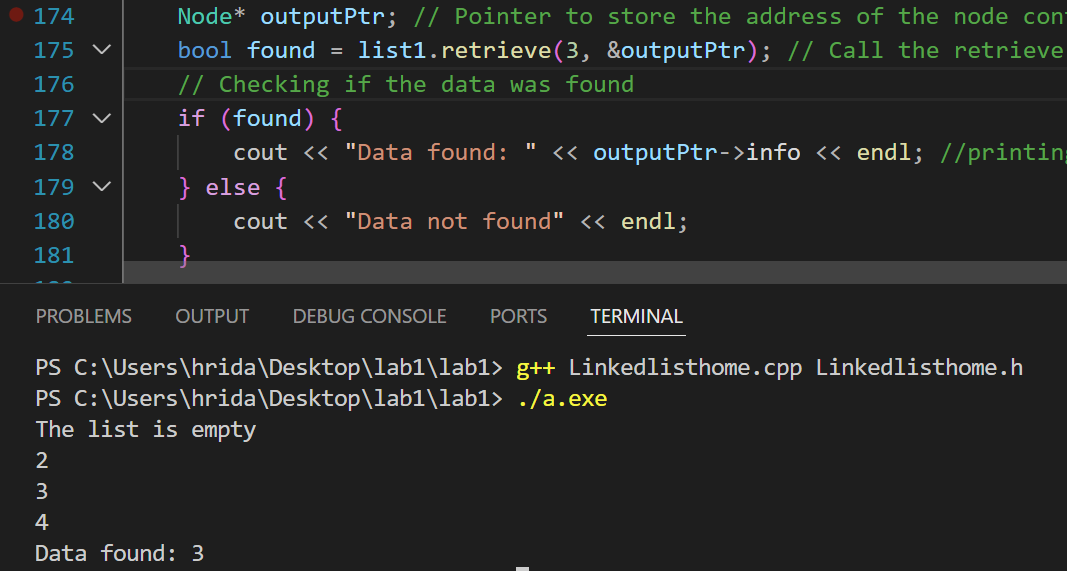
The remove(int) function takes an integer which is to be removed and remove it from the list.

1. **search(int):**

****

The search(int) checks if the required data is in the list or not and returns the message found if it exists else not found.

1. **retrieve(int, Node\*\*):**

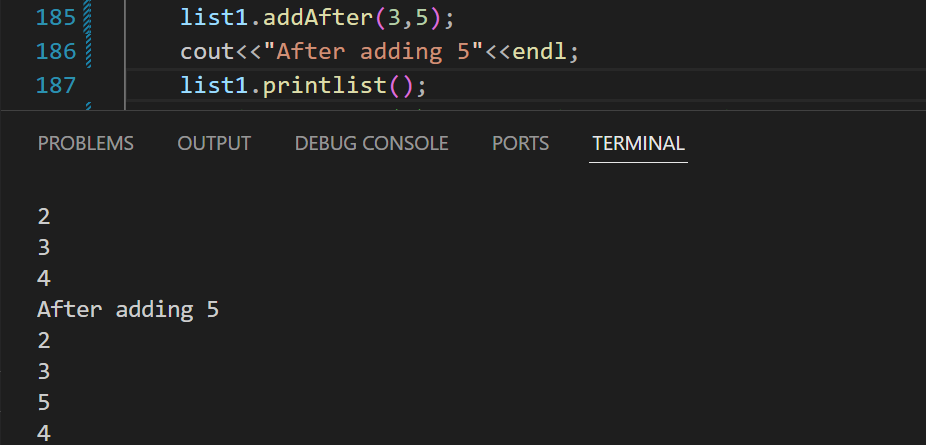


Here, the retrieve(data, outputPtr) function helps us to retrieve the pointer of the node

that the concerned data points to. When the data is retrieved, it displays the retrieved

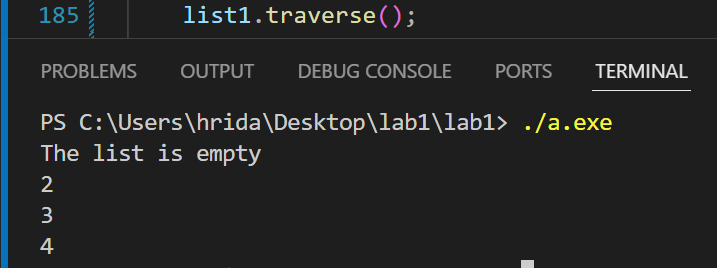
data with the message found or not found.

1. **addAfter(int, int):**

****

The addAfter(int, int) takes a predecessor node’s info and adds another node after it.

1. **traverse():**



The traverse() function is used to visit every node of the linked list and print the info stored in the node.

### Conclusion

Overall, linked lists are valuable data structures due to their flexibility, efficiency in insertion and deletion, and suitability for scenarios where dynamic memory allocation and variable-size data structures are required.