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

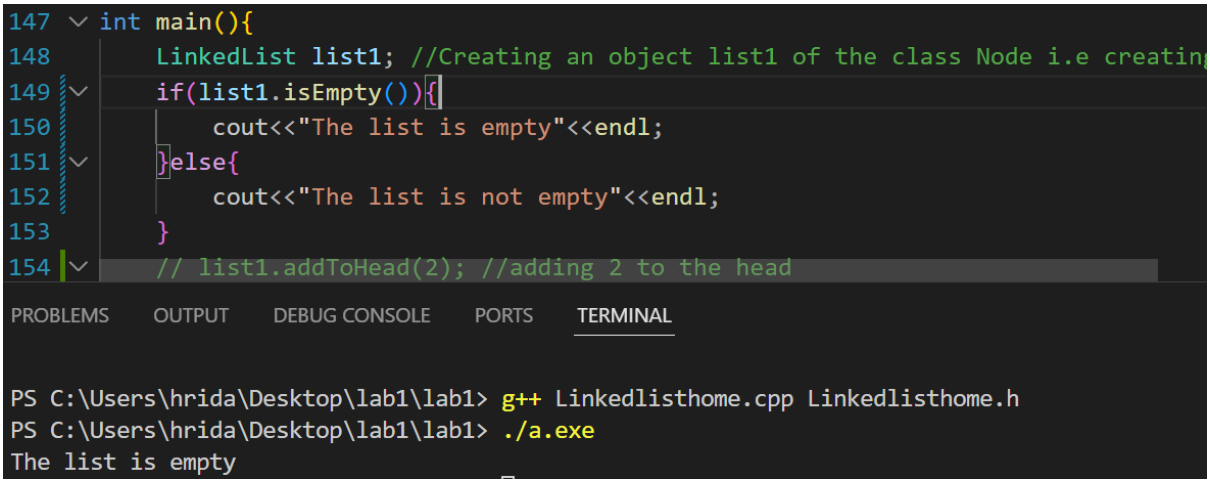
Github Link:

You can clone the following git repository and run the code:
https://github.com/hridayanshu236/lab1

Outputs:

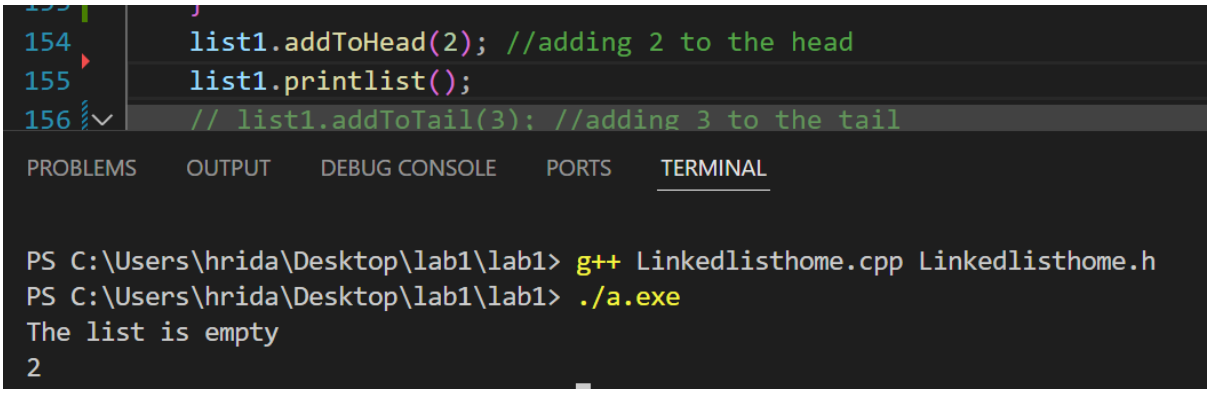
The outputs for each operation performed on the linked list are displayed below.

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.

2. **addToHead(data):**



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

3. **addToTail(data):**

```
156     list1.addToTail(3); //adding 3 to the tail
157     list1.addToTail(4); //adding 4 to the tail
158     list1.printlist();
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
The list is empty
2
3
4
```

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

4. **removeFromHead():**

```
157     list1.addToTail(4); //adding 4 to the tail
158     list1.printlist();
159     list1.removeFromHead();
160     cout<<"After removing from head"<<endl;
161     list1.printlist();
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
The list is empty
2
3
4
After removing from head
3
4
```

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

5. **removeFromTail():**

```
161     list1.removeFromTail();
162     cout<<"After removing from tail"<<endl;
163     list1.printlist();
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
The list is empty
2
3
4
After removing from tail
2
3
```

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

6. **remove(int):**

```
159     list1.remove(3);
160     cout<<"After removing 3"<<endl;
161     list1.printlist();
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
The list is empty
2
3
4
After removing 3
2
4
```

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

7. **search(int):**

```
159     list1.search(5); //searcing for the info 5
160     list1.search(2); //searching for the info 2
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL

PS C:\Users\hrida\Desktop\lab1\lab1> ./a.exe
The list is empty
2
3
4
Info not found
Info found
```

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.

8. **retrieve(int, Node**):**

```
174     Node* outputPtr; // Pointer to store the address of the node con
175     bool found = list1.retrieve(3, &outputPtr); // Call the retrieve
176     // Checking if the data was found
177     if (found) {
178         cout << "Data found: " << outputPtr->info << endl; //printin
179     } else {
180         cout << "Data not found" << endl;
181     }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL

PS C:\Users\hrida\Desktop\lab1\lab1> g++ Linkedlisthome.cpp Linkedlisthome.h
PS C:\Users\hrida\Desktop\lab1\lab1> ./a.exe
The list is empty
2
3
4
Data found: 3
```

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.

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

```
185     list1.addAfter(3,5);
186     cout<<"After adding 5"<<endl;
187     list1.printlist();
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
2
3
4
After adding 5
2
3
5
4
```

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

10. **traverse():**

```
185     list1.traverse();
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
PS C:\Users\hrida\Desktop\lab1\lab1> ./a.exe
The list is empty
2
3
4
```

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.