Data Structure and Algorithms

Lab Report

Name:	Muhammad Arslan Ishaq
Registration #:	CSU-F16-112
Lab Report #:	06
Dated:	21-05-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 6
Introduction to Doubly Linked list

Objective
The objectives of this lab session are to understand the basics of doubly
linked list.
Software Tool
1. Dev C++

1 Theory

A doubly-linked list is a linked data structure that consists of a set of se-
quentially linked records called nodes. Each node contains two fields, called
links, that are references to the previous and to the next node in the se-
quence of nodes. Doubly linked list program in C++. Each Node will have
a reference pointer to its next as well as previous node.

Advantages: :-
1. We can traverse in both directions i.e. from starting to end and as well
as from end to starting. 2. It is easy to reverse the linked list. 3. If we are
at a node, then we can go to any node. But in linear linked list, it is not
possible to reach the previous node. .

Disadvantages:-
1. It requires more space per space per node because one extra field is re-
quired for pointer to previous node. 2. Insertion and deletion take more
time than linear linked list because more pointer operations are required
than linear linked list.

Inserting and Deleting: -
To insert a node before another, we change the link that pointed to the old
node, using the prev link; then set the new node's next link to point to the
old node, and change that node's prev link accordingly..

As in doubly linked lists, "removeAfter" and "removeBefore" can be imple-
mented with "remove(list, node.prev)" and "remove(list, node.next)".

2 Lab Task

Write a C++ code using functions for the following operations. 1. Create a
Doubly Link List. 2. Traversing a Linked List

2.1 Program

```cpp
#include<iostream>
#include<stdlib.h>
#include<conio.h>
using namespace std;
struct node{
        int data;
        node* pre;
        node* next;
};

node* head = NULL;

node* getNode(int data){
        node* newNode = (node*)malloc(sizeof(node));
        (*newNode).data = data;
        (*newNode).pre = NULL;
        (*newNode).next = NULL;
        return newNode;
}

void insert(node* newNode){
        node* last_node = (node*)malloc(sizeof(node));
        last_node = head;
        head = newNode;
        newNode -> pre = NULL;
```

```cpp
        newNode -> next = last_node;
        cout<<"\n\nData inserted successfully.\n\n";
        cout<<"\n\nPress any key to continue ..............";
        getch();
        return;
}

void show(){
        node* newNode = (node*)malloc(sizeof(node));
        newNode = head;
        cout<<"\n\nData in the list \n\n";
        while(newNode != NULL){
                cout<<newNode -> data<<" ";
                newNode = newNode -> next;
        }
        cout<<"\n\nPress any key to continue ..";
        getch();
        return;
}

int main(){
        int choice, data;
        node* newNode;
        up:
        system("cls");
        cout<<"\n\n\t Enter your Choice\n";
        cout<<"\t1. Insert Data\n";
        cout<<"\t2. Show Data\n";
        cout<<"\t3. Exit\n\n\t";
        cin>>choice;
        if(choice == 1){
                cout<<"\n\nEnter data to insert: ";
                cin>>data;
                newNode = getNode(data);
                insert(newNode);
                goto up;
        }
        else if(choice == 2){
                show();
                goto up;
```
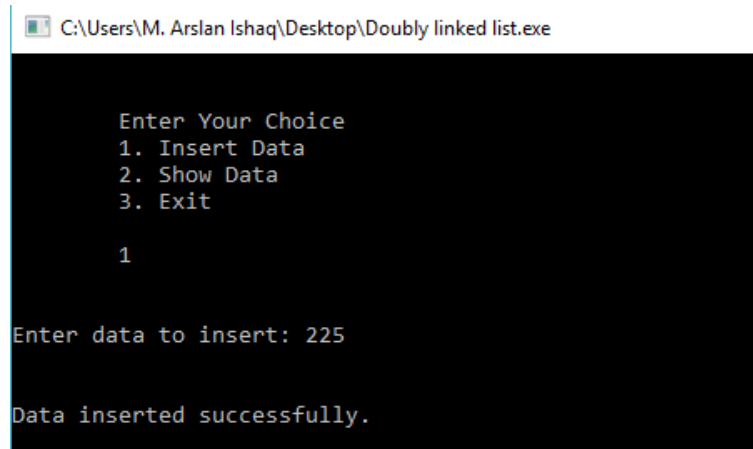
```
        }
        else  if(choice  ==  3){
                exit(0);
        }
        else{
                cout<<"\n\nWrong_Input_Choice!";
                cout<<"\n\nPress_any_key_to_continue........";
                getch();
                goto up;
        }
        return  0;
}
```


Figure : 1 Inserting Data

4
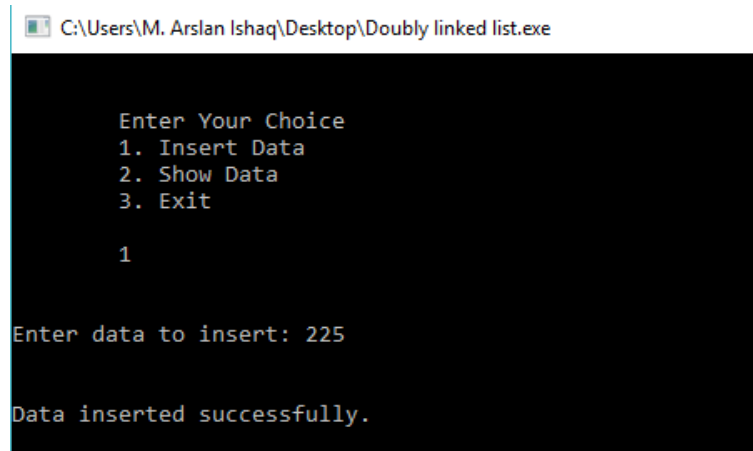
Figure : 2 Displaying Data

3 Conclusion

The insertion and remove operation works in the doubly linked list, adding extra features such as add/removing items by index, smart traversal, etc. should be easier.It is important that you have a solid conceptual understand of these basic data structures before attempting to move onto some of the more complex data structures.I recommend playing around with the doubly linked list by adding the aforementioned features such as insertion/removal by index. In the source code, I have added these features for your reference.