CS-101 Programming Fundamentals

Lab Manual

Habib University

First printing, July 2018

Contents

1. Double-linked list

1.1 Objective

In this lab, we'll learn about a data structure, called Double linked list. We'll learn about its implementation, operations and uses.

1.2 Description

Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence. It has the same operations as the single-linked list, but the operations takes less time as there are shorter traversals using links to both next and previous element. We'll see it in the examples below.

1.2.1 Real-life examples of Double linked list

There are many real life examples of stack. Consider the simple example of a music player. The music player allows skipping to both next and previous song in the playlist. Moreover, it allows to traverse through the list to find a song using both next and previous buttons.

1.2.2 Time Complexities of Double linked list

Append(data) - O(1) time
Add(i, data) - O(min(i, n - i))
Remove(i) - O(min(i, n - i))
Where i is the index and n is the size of the list.

For add and remove, we have to find the node at the index where a node is getting added or removed. Finding the node with a particular index in a DLList is easy; we can either start at the head of the list and work forward ,or start at the tail of the list and work backward. This allows us to reach the ith node in O(min(i,n - i)) time.

1.2.3 Applications of Double linked list

There are many possible applications of stack. Some are listed below:

- The browser cache which allows you to hit the BACK buttons (a linked list of URLs).
- Applications that have a Most Recently Used (MRU) list (a linked list of file names).
- A stack, hash table, and binary tree can be implemented using a doubly linked list.
- Undo functionality in Photoshop or Word (a linked list of state).

1.3 Implementation

We will be implementing a double linked list. In double linked list, we have nodes linked to each other. Each node stores a data and links to the node next to and previous to it.

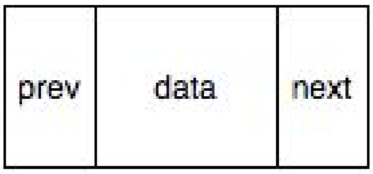

Figure 1.1: Representation of node

Implementing Node

To implement node, we will be making a struct to hold data and the links to store next and previous node . In this implementation we are making a node to store integer type data. However, it can be implemented to store any data type or object.

```
struct Node
{
    int data;
    Node* next;
    Node* prev;
    Node()
    {
        next = NULL;
        prev = NULL;
    }
};
```

Initializing the Double linked list

As a Double linked consists of many nodes, we should have many nodes inside our DLList class, but we will have only two nodes, "head" and "tail", which will act as reference nodes and allows us to iterate through all the nodes in the DLList. Moreover, we will include an integer "length" to keep track of the size of the list.

```
class DLList
{
    private:
        int length;
        Node* head;
        Node* tail;
    public:
        DLList()
        {
            head = NULL;
            tail = NULL:
        }
}
```

Our stack now consists of Node pointers which will hold the addresses of the head and the tail. To initialize, both are set to **NULL**, as the list is empty.

Figure 1.2: DLList head pointer

Figure 1.3: DLList tail pointer

Implementing Append operation

Inside the class of DLList, we will declare and define a function "Append(int data)" in public.

```
void Append(int data)
{
    if (length == 0)
    {
        head = tail = new Node();
        head->data = data;
    }
    else
    {
        tail->next = new Node();
        tail->next->prev = tail;
        tail = tail->next;
        tail->data = data;
    }
    length++;
}
```

■ **Example 1.1** Let's visualize an example Append() operation

```
    DLList lst;
    lst.Append(6);
```

Since the DLList is empty and head is set to NULL. Body of 'if' will be executed.
It first creates a new node and sets head and tail to it.

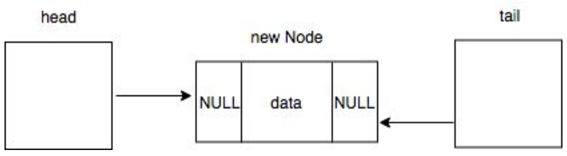

Figure 1.4: Creating new node

Then it sets its data to the data passed.

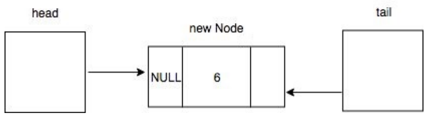

Figure 1.5: Setting data

■

Since, length is not '0', Append operation will now execute body of 'else'. Let's see an example

■ **Example 1.2**

```
lst.append(9);
```

It will first create a new node and sets tail's next to it.

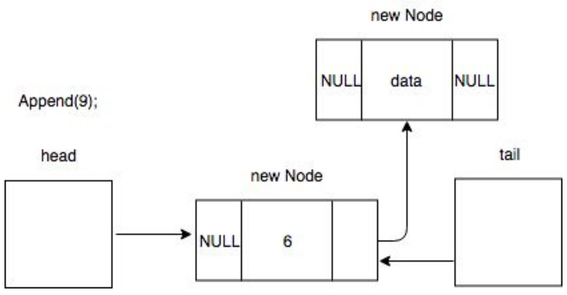

Figure 1.6: Creating new node

Then it sets the new node's prev to the tail

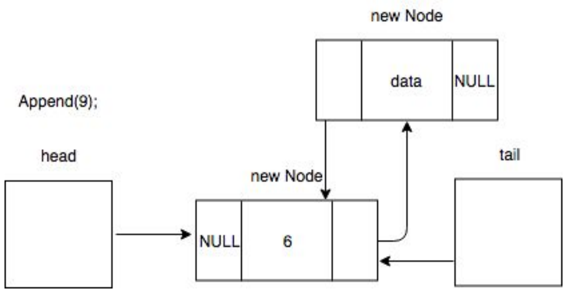

Figure 1.7: Setting new node's pointer

It now updates the tail and sets data.

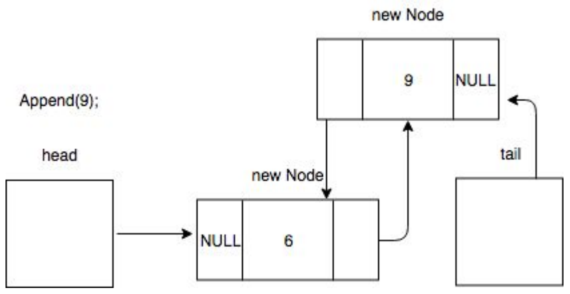

Figure 1.8: Updating tail and data

’9’ is added succesfully.

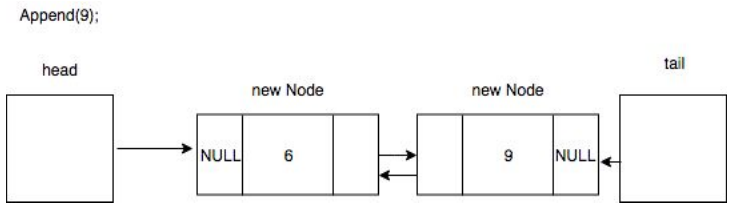

Figure 1.9: Succesful Append(9) operation

■

Implementing Add operation

Inside the class of DLList, we will declare and define a function "Append(int data)" in public.

```
void Add(int i, int data)
{
    if(i >= 0 && i <= length)
    {
        if (i == 0)
        {
            if (length == 0)
            {
                head = tail = new Node();
                head->data = data;
            }
            else
            {
                head->prev = new Node;
                head->prev->next = head;
                head = head->prev;
                head->data = data;
            }

        }
        else if (i == length)
        {
            tail->next = new Node();
            tail->next->prev = tail;
            tail = tail->next;
            tail->data = data;
        }
        else
        {
            Node* temp;
            if (i <= length/2)
            {
                temp = head;
                for (int j = 0; j < i; j++)
                {
                    temp=temp->next;
                }
            }
            else
            {
                temp = tail;
                for (int j = 0; j < length-i-1; j++)
                {
                    temp = temp->prev;
                }
            }

            Node* newNode = new Node;
            newNode->data = data;
            newNode->prev = temp->prev;
            newNode->next = temp;
            temp->prev->next = newNode;
            temp->prev = newNode;

        }
        length++;
    }
    else
    {
        cout << "Index is out of range" << endl;
    }
}
```

■ **Example 1.3** Let's visualize an example Add() operation

```
    DLList lst;
    lst.Add(0,6);
```

Since the DLList is empty, Body of 'if (i == 0) and if (length == 0)' will be executed.
It will carry out the same operations carried out in Append function.
It first creates a new node and sets head and tail to it.

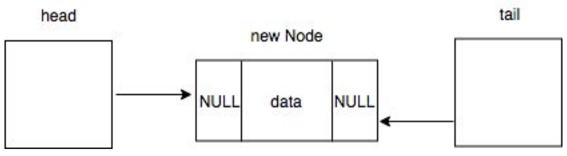

Figure 1.10: Creating new node

Then it sets its data to the data passed.

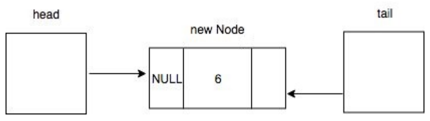

Figure 1.11: Setting data

■

If a number is added to the last, it will then also do the same tasks as append.

■ **Example 1.4**

```
    lst.Add(1,9);
```

Now length is not 0 and index is equal to the length.
It will first create a new node and sets tail's next to it.

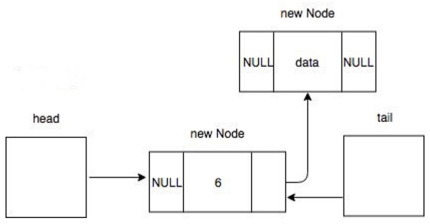

Figure 1.12: Creating new node

Then it sets the new node's prev to the tail

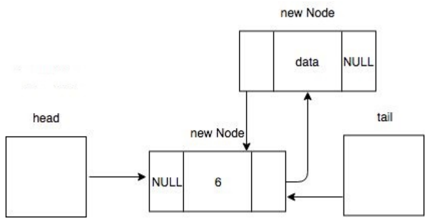

Figure 1.13: Setting new node's pointer

It now updates the tail and sets data.

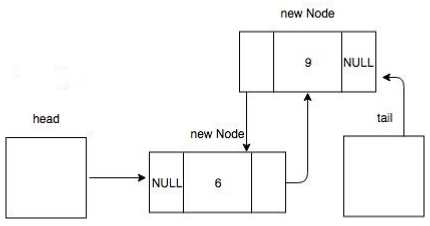

Figure 1.14: Updating tail and data

'9' is added succesfully.

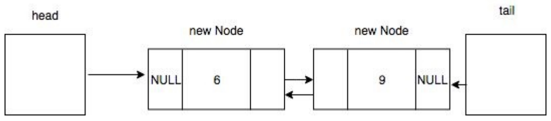

Figure 1.15: Succesful Add(1,9) operation

■

■ **Example 1.5** Let’s look at an example, when index is neither 0 nor equal to the length. Let’s visualize an example Add() operation

```
lst.Add(1,5);
```

Since i is less than half of the length, it will iterate from the head.
It will first create a temporary pointer and point it to head.

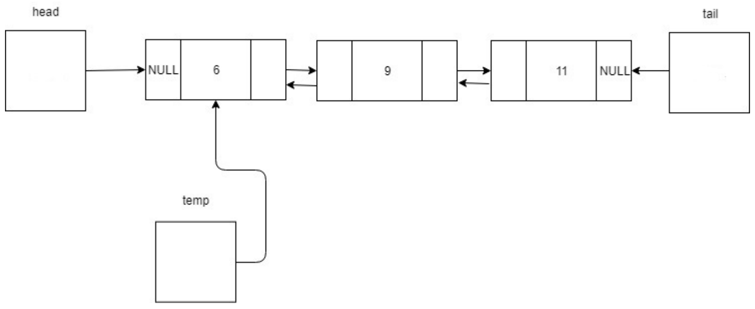

Figure 1.16: Creating a temporary pointer

It will iterate till it reaches the index where a new node is to be added.

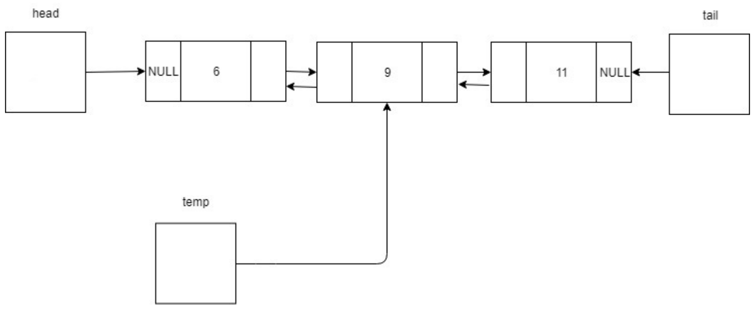

Figure 1.17: Iterating till it reaches the node

It will create a new Node.

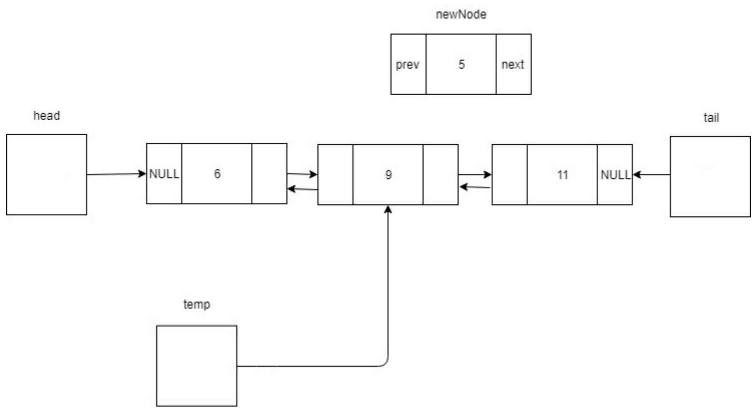

Figure 1.18: Creating new node

Points its 'prev' pointer to the temp's 'prev'.

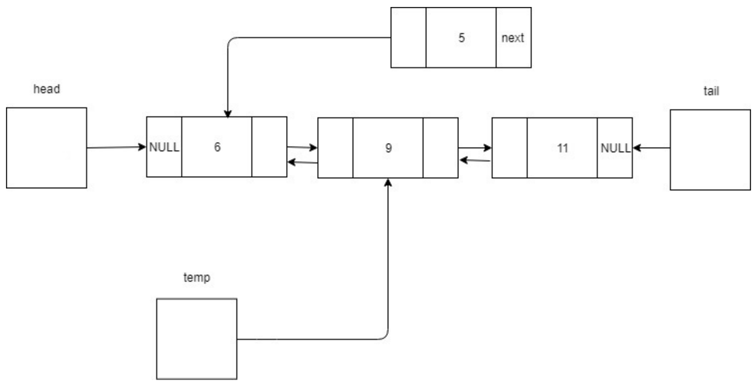

Figure 1.19: Setting node's prev pointer

Points its 'next' pointer to temp.

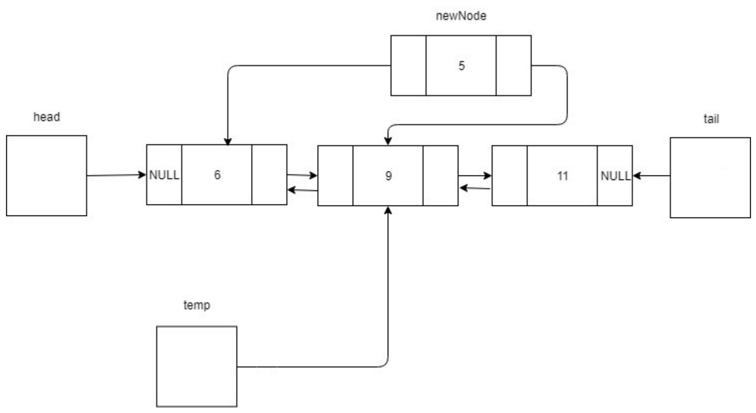

Figure 1.20: Setting node's next pointer

Point Node's prev's next pointer to itself.

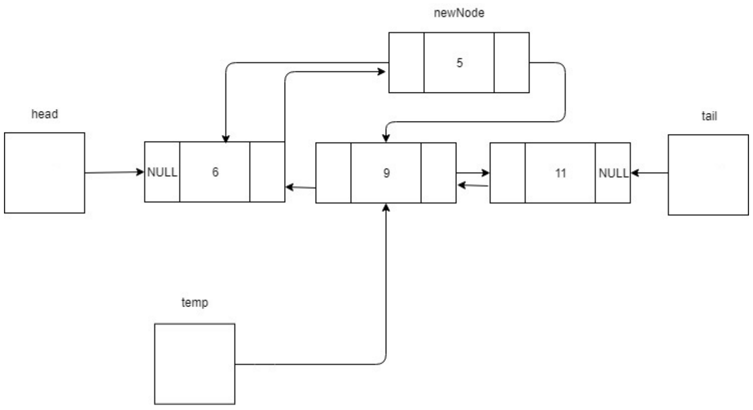

Figure 1.21: Setting the previous node's next pointer

And finally temp's prev to the new node.

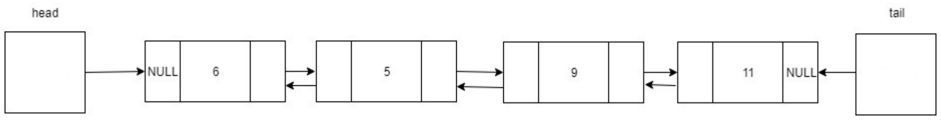

Figure 1.22: Succesful addition

■

■ **Example 1.6** Now, let's see an example where index is greater than the half of the length of the list.

```
lst.Add(2,13);
```

Since index is greater than the half of the length, it will start iteration from the last.
It will first create a temporary pointer and sets it to the tail.

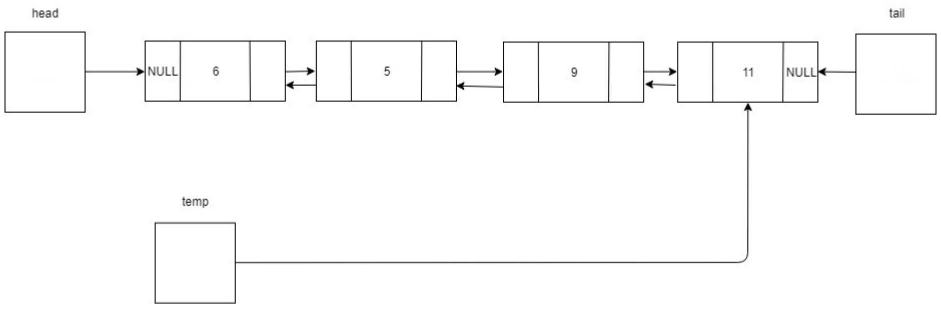

Figure 1.23: Creating a temporary pointer

It will iterate till it reaches the index where the new node has to be added

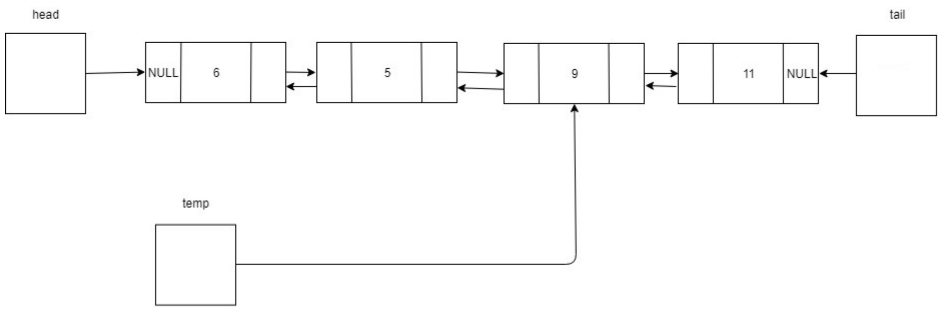

Figure 1.24: Iterated till the index

Now it repeats the same process of setting pointers as it did above and adds the new node succesfully.

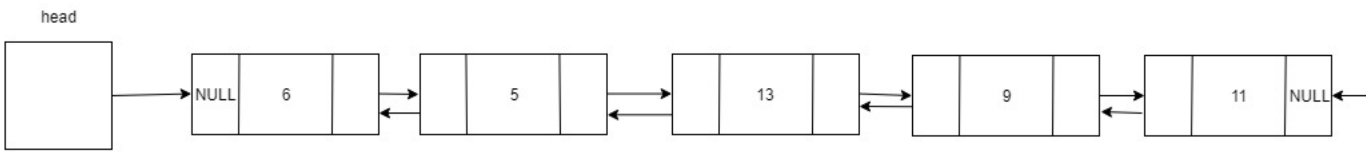

Figure 1.25: Succesful Add(2,13) operation

■

Let's see an example when the length is not '0' and the node is to be added at index '0'.

■ **Example 1.7**

```
lst.Add(0,4);
```

It will first create a new node and sets the prev's pointer, of the node pointed by head, to it.

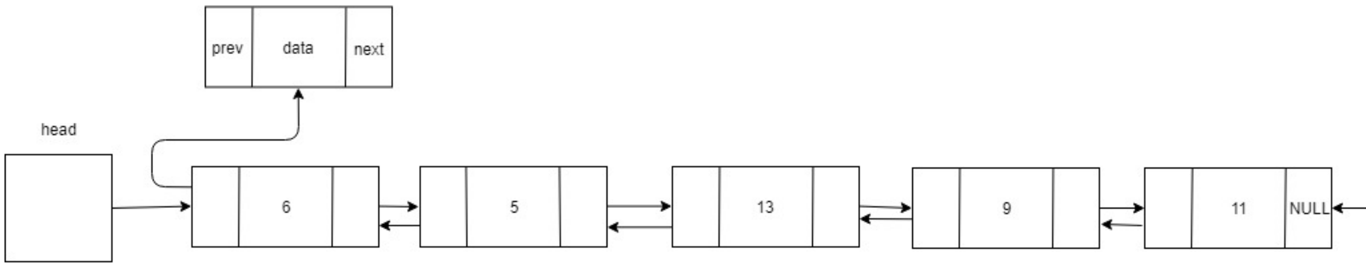

Figure 1.26: Creating new node and setting head's prev

Then it sets new node's next to the next of node pointed by head.

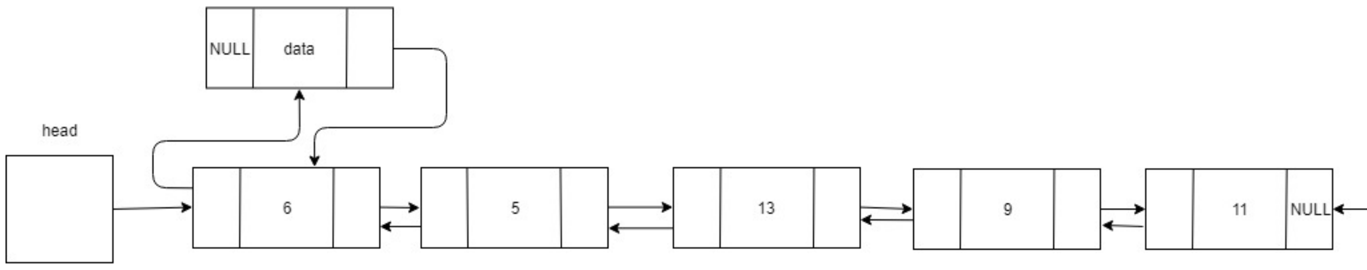

Figure 1.27: Setting node's next pointer

Finally, head is set to the new node and the node is added succesfully.

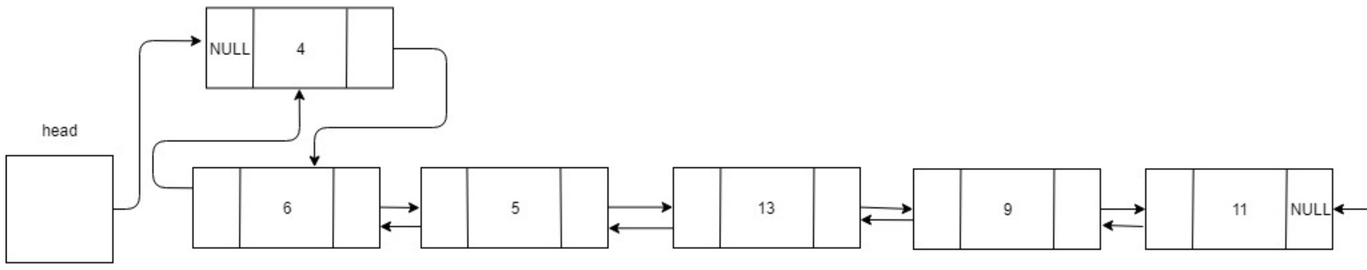

Figure 1.28: Updating head

■

Implementing Show() function

The Show() function allows to output all the values in the DLList.
Inside the class of DLList, we will declare and define a function "void Show()" in public.

```
void Show()
{
    Node* temp = head;
    while(temp!=NULL)
    {
        std::cout<<temp->data<<std::endl;
        temp = temp->next;
    }
}
```

Let’s visualize an example of Show() operation

■ Example 1.8

```
lst.Show();
```

It creates a temp pointer and is pointed to the node pointed by head. It also outputs its value

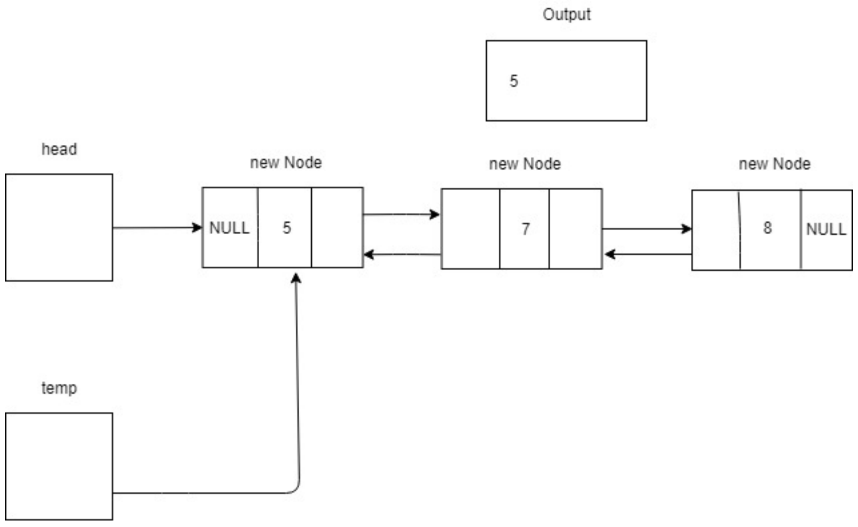

Figure 1.29: Creating temp pointer and outputting first value

It updates the temp pointer and moves to the next node. Then it outputs the value of the node pointed by temp.

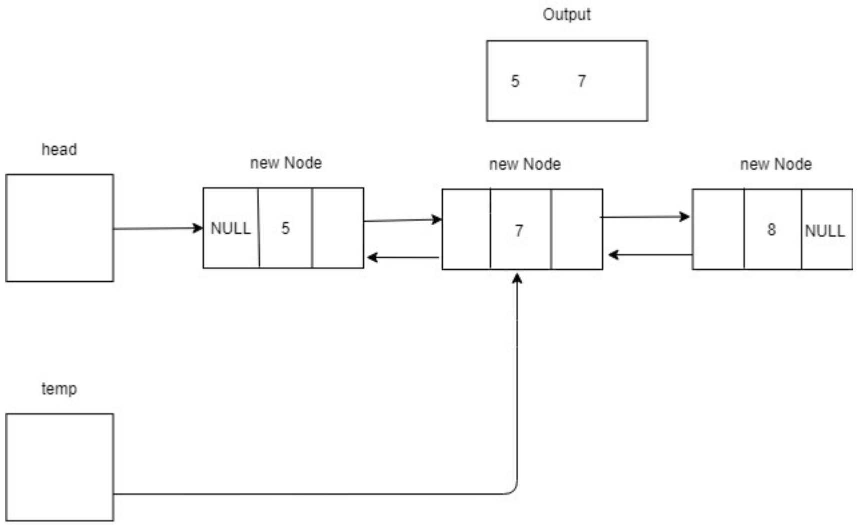

Figure 1.30: Outputting second value

And finally it moves to the last node, outputs its value and since it’s next is equal to NULL, the loop is ended.

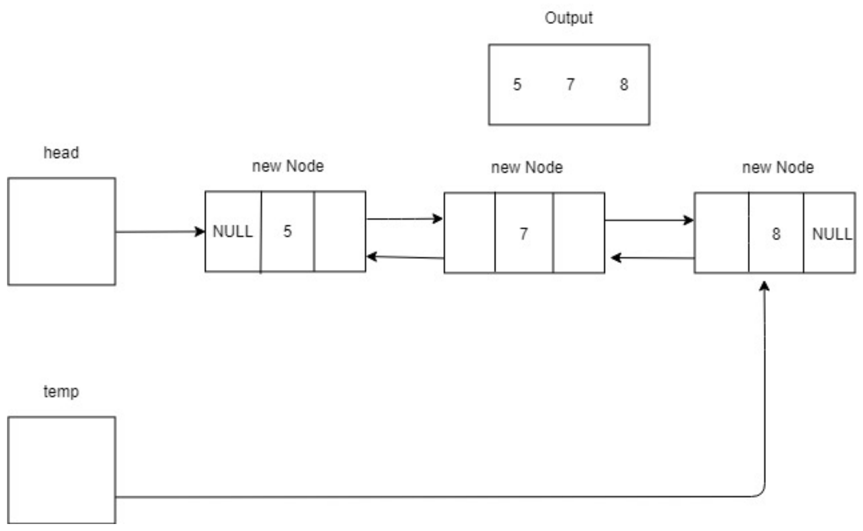

Figure 1.31: Outputting third value

■

1.4 Problems

Problem 1.1 Implement the Remove(int index) function in your DLList class. The function should remove the element from the given index and returns the removed element. The function should run in O(min(i, n - i)).

Instructions

- Think for all special cases such as, removal at zero index and removal at last index.
- Update pointers carefully.
- Try to take an idea from given Add() function for traversal.

Problem 1.2 Implement the function Reverse() which reverses the list, without making a new list.

Instructions

- Traverse through the whole list and update its pointers accordingly

1.5 Feedback

Please write the things you’ve learned from this lab and suggestions to make it more better and easy to learn.