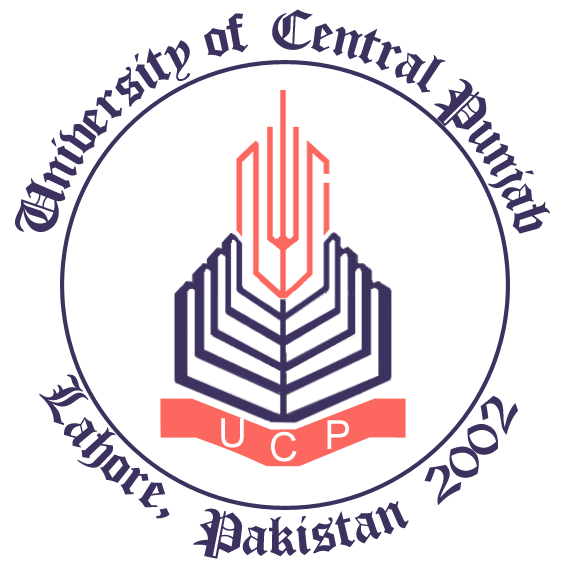
Data Structures  
Lab Manual 4



Topic: Doubly Link List

Session: Spring 2023

Faculty of Information Technology

UCP Lahore Pakistan

**Objectives:**The objective of a doubly linked list is to allow efficient insertion and deletion of nodes at any position in the list. Unlike a singly linked list, which only allows traversal in one direction (forward), a doubly linked list allows traversal in both directions (forward and backward), making it easier to insert or delete nodes at any position in the list. Some of the main objectives of using a doubly linked list are:

* Efficient insertion and deletion of nodes at any position in the list
* Efficient traversal in both directions (forward and backward)
* Flexibility in managing nodes in the list, such as easily swapping two adjacent nodes or reversing the order of the nodes in the list
* Efficient implementation of other data structures, such as stacks, queues, and hash tables, which can be built on top of a doubly linked list.

Overall, the objective of a doubly linked list is to provide a flexible and efficient data structure for managing sequences of data elements that require frequent insertion and deletion operations.

**Doubly Link List:**

A singly linked list is a collection of components, called nodes. Every node (except the last node)

contains the address of the next node. Thus, every node in a linked list has two components: one to

store the relevant information (that is, data) and one to store the address, called the link, of the next

node in the list. The address of the first node in the list is stored in a separate location, called the

head or first. Last node to list points to NULL

A singly linked list is a collection of components, called nodes. Every node (except the last node)

contains the address of the next node. Thus, every node in a linked list has two components: one to

store the relevant information (that is, data) and one to store the address, called the link, of the next

node in the list. The address of the first node in the list is stored in a separate location, called the

head or first. Last node to list points to NULL

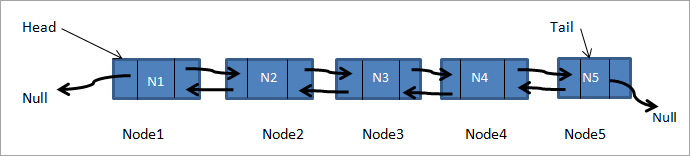
A singly linked list is a collection of components, called nodes. Every node (except the last node)

contains the address of the next node. Thus, every node in a linked list has two components: one to

store the relevant information (that is, data) and one to store the address, called the link, of the next

node in the list. The address of the first node in the list is stored in a separate location, called the

head or first. Last node to list points to NULL

A doubly linked list is a type of data structure in which each node contains a data element and two pointers, one pointing to the previous node and the other pointing to the next node. This allows for efficient traversal in both forward and backward directions, making it useful for applications such as implementing a cache or a music playlist. However, it requires more memory than a singly linked list due to the additional pointer for each node.  
  


The head node contains a pointer to the first node in the list. Since a doubly linked list can be traversed in both directions, the head node also contains a null pointer for the previous node.

The last node, is the final node in the list and contains a null pointer for the next node. It also contains a pointer to the second to last node in the list, allowing for traversal in reverse order.

By maintaining pointers to both the head and last nodes, we can easily insert or delete nodes at both the beginning and end of the list.

We can implement this organization by storing the information and handle the next and previous element with the other data as follows:-

  class DoubleListNode

{

Node \*prev; // Pointer to previous node  
 Node \*next; // Pointer to next node

int data;

};

A doubly list looks like as shown below:



**Basic Operations that can be performed on Linked List:**

* Traversal:  
   To traverse all the nodes one after another
* Append a new node:   
   To add a new node at the end
* Prepend a new node:   
   To add a new node at the beginning
* Insertion:

To add a node at the given position

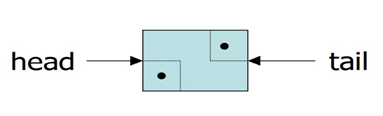
* Deletion:

To delete a node from the list

* Updating:  
   To update a node in the list
* Searching:   
   To search an element by value
* Sorting:  
   To arrange nodes in a linked list in a specific order
* Calculate length:

To count number of nodes in the list

**Adding first Node**



// Adding first Node

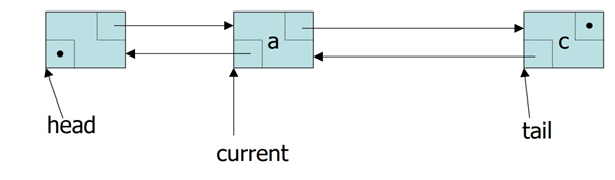
head = new DoubleListNode;

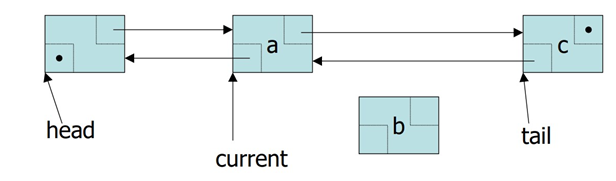
head->next = null;

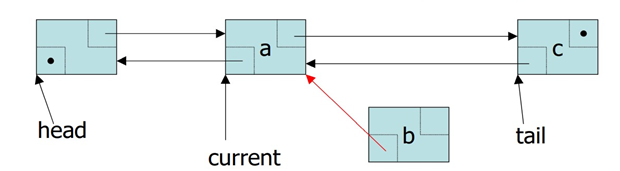
head->prev = null;

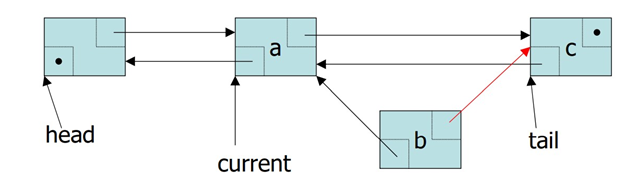
tail = head;

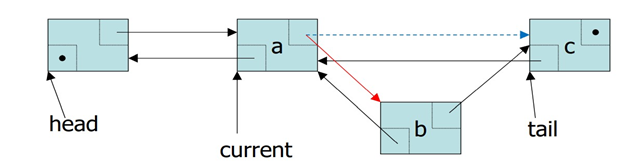
**Inserting a Node in a doubly-linked List**

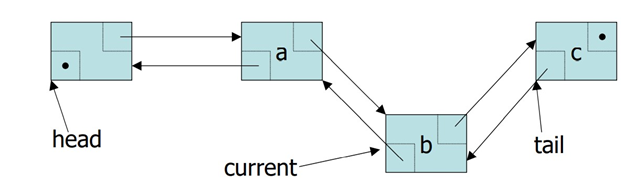
Add a new item after the linked list node pointed by current. ****

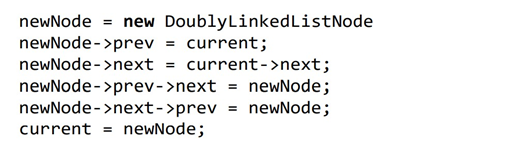
****

****

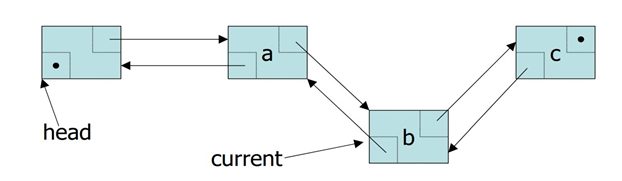
****

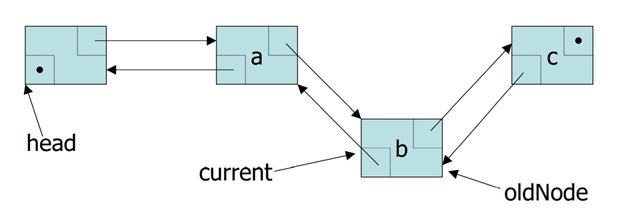
****

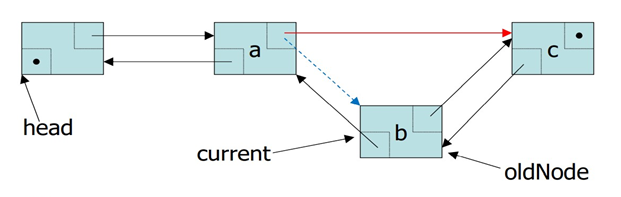


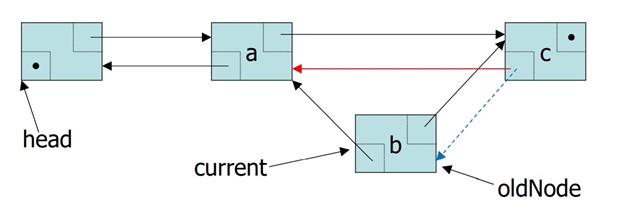


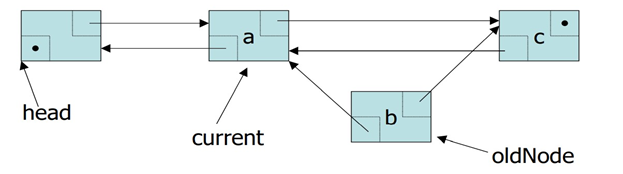
**Removing a Node from Doubly-linked List**Suppose Current points to the node to be removed from the list

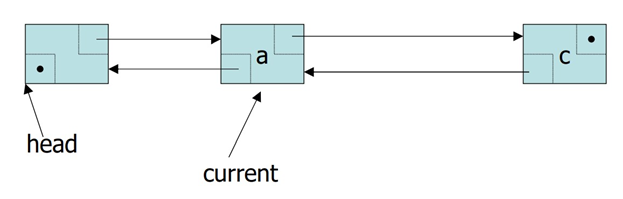


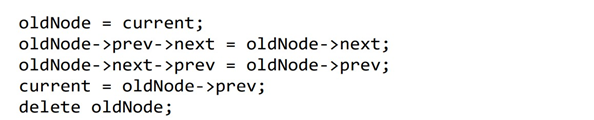




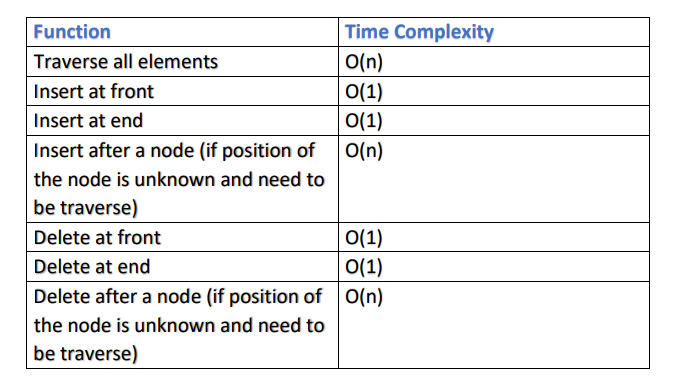








**Time Complexity of doubly linked list**



**Sample Task**A program that uses class DoubleLinkedList to create a doube linked list of integers and then display it on console.  
**Pseudocode:**

1. Define a class called Node with three fields: data, prev, and next.
2. Define a class called DoubleLinkedList with two fields: head and tail, both of type Node.
3. Define a method called addNode that takes an integer parameter called data.
4. Inside the addNode method, create a new Node with the given data, and set its prev field to the current tail and its next field to null.
5. If the current tail is null, set it to the new Node, and set the head to the same Node.
6. Otherwise, set the current tail's next field to the new Node, and set the new Node as the new tail.
7. Define a method called displayList that takes no parameters.
8. Inside the displayList method, create a Node variable called current and set it to the head.
9. Loop through the list while the current variable is not null.
10. Inside the loop, print the current Node's data to the console, and set current to the next Node in the list.
11. Define getter and setter methods for the head and tail fields in the DoubleLinkedList class.
12. Use the addNode method to add integers to the list.
13. Use the displayList method to display the list on the console.
14. End of the program.

**Implementation:**#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

};

class DoubleLinkedList {

public:

Node\* head;

Node\* tail;

DoubleLinkedList() {

head = NULL;

tail = NULL;

}

void addNode(int value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = NULL;

if (head == NULL) {

newNode->prev = NULL;

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

Node\* getHead() {

return head;

}

Node\* getTail() {

return tail;

}

};

int main() {

DoubleLinkedList myList;

myList.addNode(11);

myList.addNode(22);

myList.addNode(33);

myList.addNode(44);

Node\* currentNode = myList.getHead();

while (currentNode != NULL) {

cout << currentNode->data << " ";

currentNode = currentNode->next;

}

return 0;

}

**Task 1:**

Use the linked list that you created in the previous lab to perform the following tasks.

* 1. Function called **DeleteList** to delete the complete list.
  2. Function called **ReverseDisplay** to display the complete list in reverse order.

**Task 2:**

Create your own class of doubly link list which will have the following functions.

~~a. Function called InsertAtBegin to add node at the begging of list.~~

~~b. Function called InsertAtEnd to add node at the last of list.~~

~~c. Function called InsertBefore to inset new value before specific value in list.~~

~~d. Function called InsertAfter to inset new value after a specific value in list.~~

~~e. Function called DeleteFromBegin to delete node from the begging of list.~~

~~f. Function called DeleteFromEnd to delete node from the last of list.~~

~~g. Function called DeleteN to delete any node from the list.~~

~~h. Function called Delete to delete maximum value in list.~~

~~i. Function called DeleteList to delete the complete list.~~

~~j. Function called Search to search any node from the list.~~

~~k. Function called Display to display list on console.~~

~~l. Function called ReverseDisplay to display the complete list in reverse order.~~

**Note: First, work with a linked list that contains a single attribute, such as a name or an ID number. Perform each of the above-mentioned operations on this list. Next, update the list to include more attributes, such as an email address and a phone number, and perform the same operations on the updated list.**