**DHA Suffa University**



**Department of Computer Science**

**CS 2001L – Data Structures and Algorithms Lab**

**Fall 2019**

**Lab 06 – Singly & Doubly Double Ended and Circular Link List**

Objective:

* Be familiar with basic techniques of doubly LL and circular LL
* Be familiar with writing algorithm of doubly LL and circular LL
* Be familiar with various method to implement doubly LL and circular LL

What is Doubly Linked List?

A doubly linked list is a linked list in which every node has a next pointer and a back pointer. Every node contains the address of the next node (except the last node), and every node contains the address of the previous node (except the first node). A doubly linked list can be traversed in either direction.

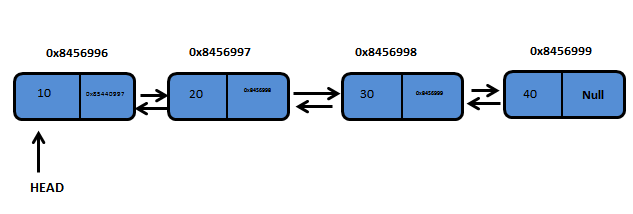


Figure 6.1: doubly Linked List

Doubly Linked List are more convenient than Singly Linked List since we maintain links for bi-directional traversing. We can traverse in both directions and display the contents in the whole List.

In Doubly Linked List we can traverse from Head to Tail as well as Tail to Head. Each Node contains two fields, called Links that are references to the previous and to the Next Node in the sequence of Nodes.

Insertion

In a doubly linked list, the insertion operation can be performed in three ways. They are as follows...

* Insertion At Beginning of the list
* Insertion At End of the list
* Insertion At Specific location in the list

Double Ended-Singly List

|  |
| --- |
| **Algorithm:** Insert (*value*)  **Pre:** *value* is the value to add to the list  **Post:** *value* has been placed at the tail of the list    temp  Node(value)  if head = NULL  head  temp  tail  temp  end if  else  tail.next  temp  tail  temp  end else  end Algorithm: Insert (*value*) |

|  |
| --- |
| **Algorithm:** Search (*value*)  **Pre:** *value* is the value to search for  **Post:** the item is either in the linked list, true; otherwise false  current <- head  loop current ≠ NULL AND current.value ≠ value  current <- current.next  end loop  if current = NULL  return false  end if  return true  **end Algorithm:**Insert (*value*) |

|  |
| --- |
| **Algorithm:** Delete (value)  **Pre:** value is the value to be deleted from list  **Post:** value is removed from the list, true; otherwise false  if head = NULL  return false  end if  current <- head  if current.value = value  if head = tail  head <- NULL  tail <- NULL  end if  else  head <- head.next  end else  return true  end if  loop current.next ≠ NULL AND current.next.value ≠ value  current <- current.next  end loop  if current.next ≠ NULL  if current.next = tail  tail <- current  end if  current.next<-current.next.next  return true  end if  return false  **end Algorithm:** Delete (value) |

Double Ended - Doubly Link List

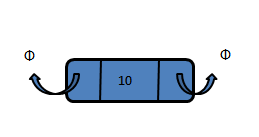
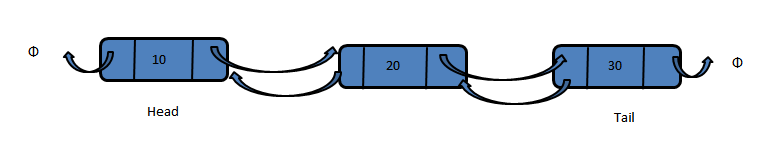


Figure 6.2: Doubly Single Node

 Figure 6.3: Structure of Doubly Link List

|  |
| --- |
| **Algorithm:** Insert (*value*)  **Pre:** *value* is the value to add to the list  **Post:** *value* has been placed at the tail of the list  temp <- Node(value)  if head = NULL  head <- temp  tail <- temp  end if  else  temp.previous <- tail  tail.next <- temp  tail <- temp  end else  **end Algorithm:** Insert (*value*) |

|  |
| --- |
| **Algorithm:** Delete (*value*)  **Pre:** *value* is the value to be deleted from list  **Post:** *value* is removed from the list, true; otherwise false  if head = NULL  return false  end if  if value = head.value  if head = tail  head <- NULL  tail <- NULL  end if  else  head <- head.next  head.previous <- NULL  end else  return true  end if  current <- head.next  loop current ≠ NULL AND value ≠ current.value  current <- current.next  end loop  if current = tail  tail <- tail.previous  tail.next <- NULL  return true  else if current ≠ NULL  current.previous.next <- current.next  current.next.previous <- current.previous  return true  end if  return false  **end Algorithm:** Delete (*value*) |

|  |
| --- |
| **Algorithm:** ReverseTraverse ()  **Pre:** link list already created  **Post:** the items in the list have been traversed in reverse order    temp  Node(NULL)  temp  tail  loop temp ≠ NULL  yield temp.value  temp temp.previous  end loop  end Algorithm: ReverseTraverse () |

Circular linked list

In single linked list, every node points to its next node in the sequence and the last node points NULL. But in circular linked list, every node points to its next node in the sequence but the last node points to the first node in the list.

Circular linked list is a sequence of elements in which every element has link to its next element in the sequence and the last element has a link to the first element in the sequence.

That means circular linked list is similar to the single linked list except that the last node points to the first node in the list

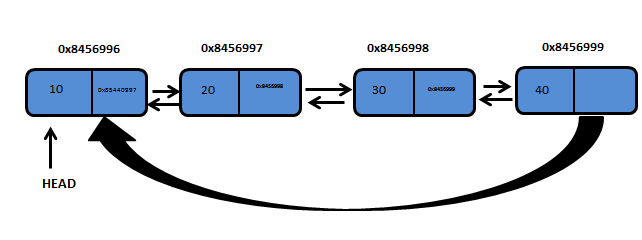


Figure 6.4: Circularly Linked List

Insertion

In a circular linked list, the insertion operation can be performed in three ways. They are as follows...

* Insertion At Beginning of the list
* Insertion At End of the list
* Insertion At Specific location in the list

|  |
| --- |
| **Algorithm:** InsertAtLast (*value*)  **Pre:** A circular linked list is initialized  **Post:** Insert an element at the end  temp  Node(value)  if (head = NULL)  head <- temp  head.next <- head  end if  else  temp.next <- head  current  Node(NULL)  current<-head  loop current.next ≠ head  current<-current.next  end loop  current.next<-temp    end else  **end Algorithm:**InsertAtLast (*value*) |

|  |
| --- |
| **Algorithm:** DeleteFirst ( )  **Pre:** A circular linked list  **Post:** delete an element from the beginning  if head = NULL  return false  end if  Node current <- head;  if(head.next = head)  head <- NULL  return true  end if  else  loop current.next ≠ head  current <- current.next;  end loop  head <- head.next;  current.next <- head;  return true;  end else    **end Algorithm:**DeleteFirst () |

**Assignment:**

**Q.1)**  Given an input file, write a program to read that file in a doubly link list. Each row in a file is a node, linked to every other node (row) of the file.

**a.** Your program should sort the doubly linked list using the sorting algorithms

discussed in Lab 3(sort on the basis of Student Name).

**Q.2)** Given an input file, write a program to read that file in a circular link list. Each row in a file is a node, linked to every other node (row) of the file.

**a.** Using the above created circular linked list, delete a particular record by asking

from user the value to be deleted. Test your program by deleting any record from empty

list. In such case, display a proper error message saying that the value doesn’t exist in

the list otherwise display the updated list after deletion.

# **Submission Guidelines**

* **Write C++ code , separate function for each operation.**
* **Place your file in a folder named with your rollNo (cs172xxx) where xxx is your 3 digit rollno.**
* **Upload it on LMS.**