DHA Suffa University
Department of Computer Science
CS 2001L – Data Structures and Algorithms Lab
Spring 2021

Lab 06 – Doubly 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.

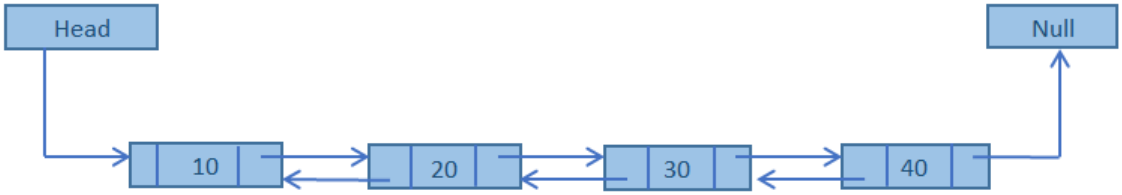

Figure 6.1: doubly Linked List

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 - Doubly Link List

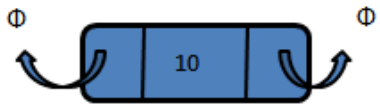

Figure 6.2: Doubly Single Node

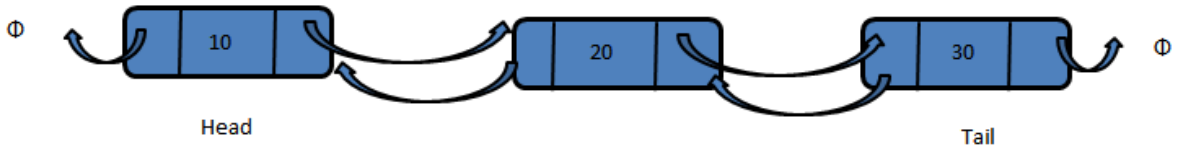

Figure 6.3: Structure of Doubly Link List

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

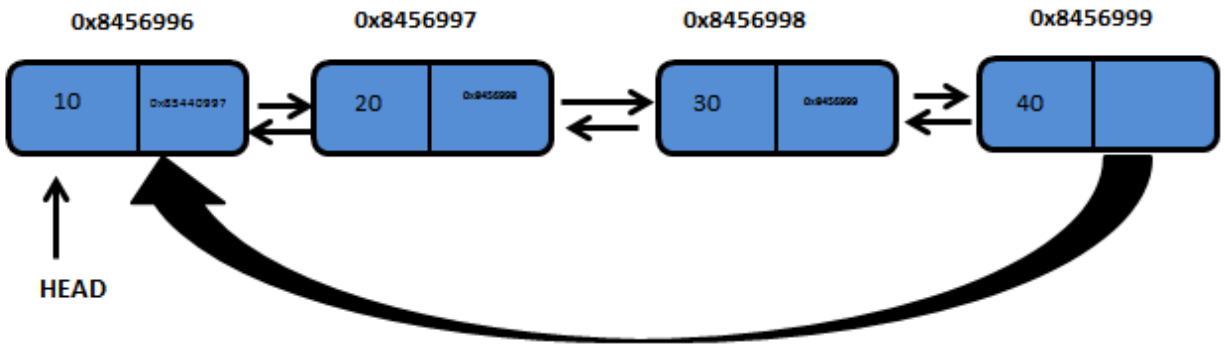

Figure 6.4: Circularly Linked List

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:

1. Write member functions of the doubly linked list class to insert at first, insert at a specific position and for forward traversal.

2. Create two doubly linked lists, list1 and list2 containing at least 6 values each and find the total number of common nodes in both the doubly linked lists. Also, write a member function to merge both the linked lists

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.**