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

Lab 05 – Linked List

Objective:

To learn about

- Singly linked list
- Insertion in Singly linked list
- Deletion in Singly linked list
- Traversal in Singly linked list

Linked List:

A linked list is a linear data structure where each element is a separate object. Linked list elements are not stored at contiguous location; the elements are linked using pointers. Each node of a list is made up of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.

Pros:

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

Cons:

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

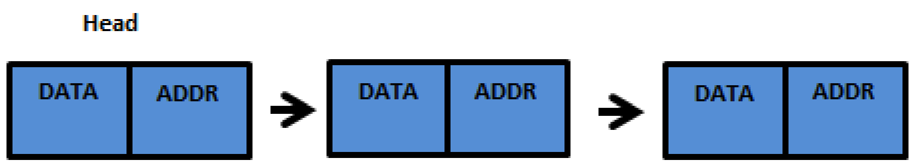

Figure 5.1: Linked List

For creating a node, you can use the following structure to hold the node data structure:

```
struct node {
     int data;
     node* next;
};
```

<u>Singly Linked List</u>

Simply a list is a sequence of data, and linked list is a sequence of data linked with each other.
The formal definition of a single linked list is as follows...
Single linked list is a sequence of elements in which every element has link to its next element in the sequence.
In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data and next.

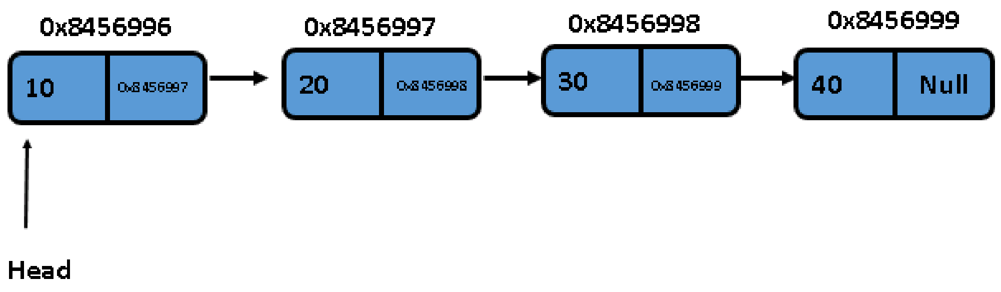

Figure 5.2: Singly Linked List

In a single linked list we perform the following operations...

- Insertion
- Deletion
- Traverse

Insertion

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

- Inserting At End of the list
- Inserting At Beginning of the list
- Inserting At Specific location in the list

Algorithm: InsertAtLast (value)

Pre: value is the value to add to the list

Post: value has been placed at the end of the list

```
temp <- Node (value)
if head = NULL
     head <- temp
end if
else
Current <- head
loop current.next ≠ NULL
current <-  current.next
end loop
current.next <-  temp
end else
```

end Algorithm: InsertAtLast (value)

Algorithm: InsertAtFirst (value)

Pre:value is the value to add to the List

Post: value has been placed at the

```
head of the list
temp <-  Node (value)
if head = NULL
head <- temp
end if
else
temp.next <- head
head <- temp
end else
```

end Algorithm: InsertAtFirst (value)

Algorithm: InsertAtPosition (value, position)

Pre: value is the value to add to the list position is the position at which the value will be inserted

Post: value has been placed at the specified position of the list

```
temp <- Node (value)
prev <- Node (NULL)
if head = NULL
head <- temp
end if
else
current <- head
loop i <- 1 to position -1
prev = current
current <- current.next
end loop
if prev = NULL
temp.next <- head
head = temp
end if
else
temp.next <- current.next
current.next<- temp
end else
end else
```

end Algorithm: InsertAtPosition (value, position)

Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows…

- Deleting At Beginning of the list
- Deleting At Specific position in the list
- Deleting At End of the list

Algorithm: DeleteAtFirst ( )

Pre: LinkList is already created

Post: node deleted from the head of the list

```
if head = NULL
print "List Empty"
```

```
end if
else
temp <- head
head <- head.next
end else
return temp
```
end Algorithm: DeleteAtFirst ( )

Algorithm: DeleteAtLast ( )

> **Pre:** LinkList is already created
> **Post:** node deleted from the end of the list

```
if head = NULL
print "List Empty"
end if
else
current <- head
loop current.next.next ≠ NULL
current <- current.next
end loop
temp <- current
current.next <- NULL
end else
return temp
```
end Algorithm: DeleteAtLast ( )

Algorithm: DeleteNode (value)

> **Pre:** value is the value to remove from the list
> **Post:** value is removed from the list, true; otherwise false

```
//Fill by yourself
//Delete a node with a particular value
```
end Algorithm: DeleteNode ( )

Algorithm: DeleteAtPosition (position)

> **Pre:** position is the position at which the value will be deleted
> **Post:** node has been deleted from the specified position of the list

```
if head = NULL
print "List Empty"
end if
else
current <- head
loop i <- 1 to position -1
current <- current.next
end loop
if current = head
   head = current.next
end if
current.next <- current.next.next
end else
```
end Algorithm: DeleteAtPosition (position)

Traversal in Singly Link list

Algorithm: ForwardTraversal ( )

  Pre: LinkList is already created

  Post: the items in the list have been traversed

```
if head = NULL
print "List Empty"
end if
else
current = head
loop current != NULL
yield current.value
current = current.next
end loop
end else
```
end Algorithm: ForwardTraversal ( )

Algorithm: ReverseTraversal ( )

  Pre: LinkList is already created

  Post: the items in the list have been traversed

```
//Fill it on your own
```
end Algorithm: ReverseTraversal ( )

LAB ASSIGNMENT

1. Write a member function of the linked list class to search the node with the given data.

2. Write a member function of the linked list class to delete the node with the given data.

3. Write a function which returns the reversed form of the given linked list.

SUBMISSION GUIDELINES

- Take a screenshot of each task (code and its output), labeled properly.
- Place all the screenshots and the code files (properly labeled) in a single folder labeled with Roll No and Lab No. e.g. **'cs191xxx_Lab01'**.
- Submit the folder at LMS
- **-100%** policies for plagiarism.