KJ's Educational Institutes
K J College Of Engineering & Management Research, Pune.
Department of E & TC

CLASS: S. E. (E &TC) **SUBJECT:-DSA**

Ex. No: 3 **Date:**

AIM: Singly Linked List Operations

You are building a text editor where lines of text are stored dynamically. You need to allow insertion and deletion of lines at any position, and display text both normally and in reverse.

Use a singly linked list to implement: display, insert (front/end/middle), delete (front/end/middle), display in reverse, and reverse the list.

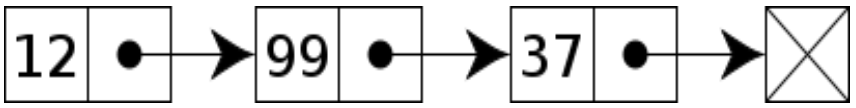
OBJECTIVES

After performing this experiment, student should be able to:

- Understand the concept of a singly linked list and various operations on it
- Write functions for other simple operations on such a linked list.
- Design a program for other variants of singly linked list that include use of a pointer to last node, header node, circular linked list, and circular list with header node.

THEORY

In computer science, a linked list is data structure that consists of a sequence of data records such that in each record there is a field that contains a reference (i.e., a *link*) to the next record in the sequence.



A linked list whose nodes contain two fields: an integer value and a link to the next node

Linked lists are among the simplest and most common data structures, and are used to implement many important abstract data structures, such as stacks, queues, hash tables, symbolic expressions, skip lists, and many more.

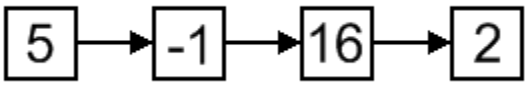
The principal benefit of a linked list over a conventional array is that the order of the linked items may be different from the order that the data items are stored in memory or on disk. For that reason, linked lists allow insertion and removal of nodes at any point in the list, with a constant number of operations.

On the other hand, linked lists by themselves do not allow random access to the data, or any form of efficient indexing. Thus, many basic operations — such as obtaining the last node of the list, or finding a node that contains a given data, or locating the place where a new node should be inserted — may require scanning most of the list elements.

Singly-linked list:

Linked list is a very important dynamic data structure. Basically, there are two types of linked list, singly-linked list and doubly-linked list. In a singly-linked list every element contains some data and a link to the next element, which allows to keep the structure. On the other hand, every node in a doubly-linked list also contains a link to the previous node. Linked list can be an underlying data structure to implement stack, queue or sorted list.

Sketchy, singly-linked list can be shown like this:



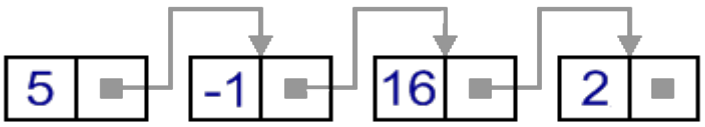
Each cell is called a **node** of a singly-linked list. First node is called **head** and it's a dedicated node. By knowing it, we can access every other node in the list. Sometimes, last node, called **tail**, is also stored in order to speed up add operation.

Singly-linked list. Internal representation:

Every node of a singly-linked list contains following information:

- A value (user's data);
- A link to the next element (auxiliary data).

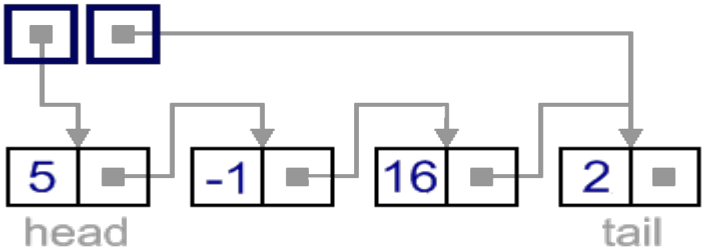
Sketchy, it can be shown like this:



First node called **head** and no other node points to it. Link to the head is usually stored it the class, which provides an interface to the resulting data structure. For empty list, head is set to *NULL*.

Also, it makes sense to store a link to the last node, called **tail**.

Though no node in the list can be accessed from the tail (because we can move forward only), it can accelerate an add operation, when adding to the end of the list. When list is big, it reduces add operation complexity essentially, while memory overhead is insignificant. Below you can see another picture, which shows the whole singly-linked list internal representation:



Algorithm:

Insertion in ordered list:

// x is the data element of the node to be inserted in linked list
// list is the starting node of the linked list
// next refers to the next node of the linked list
// info refers to info field of the node

```
Getnode (new)
Info (new) =x
next (new) =null
if list = null
 list = new
else
    begin
```

```
p=list
                    if x < info (p)
                    then next (new) = list
                            list = new
                else
                    begin
                            while (next (p) <> null) and (x >= info (next (p))) do
                            p= next (p)

                            next (new)= next(p)
                            next (p) = new
                     end else //end of inner else
                end else //end of outer else
```

Deleting from a linked list:

```
// current initially refers to first node of the linked list
// trail refers to one node previous to current
                    current = list
                    trail = null
                    while ( current <> null) and (info (current) <> x) do
                    begin
                            trail = current
                            current = next (current)
                    end/* end of the loop to search the node */
                    if current not null
                    then if trail = null
                            then
                                    list = next (list) /* delete the first node */
                            else
                                    next (trail) = next (current)
                            freenode (current)
                    else print ("node with the given value doesn't exist in the list" )
```

Search a node in a given linked list:

```
        // current initially refers to the first node of the linked list
```

```
// next refers to the next node
                count=0
                while (current <> null) and info (current) <> x) do
                        current = next (current)
                        count = count +1
                        end // end of the loop
                if (current = null)
                        call print(" node with the given value doesn't exist in the list")
                else
                        display the position and the content of the node
```

Display the content of linked list:

```
// list is the starting node of the linked list
                if (list = null)
                        call print(" linked list is empty ")
                else
                while (current <> null ) do
                        call print( info(current))
                        current = next (current)
                        end //end of loop
                        end //end else
```

Input:

Input is the data/element that is to be added to the linked list as a new node.

Output:

Output is the data/elements present in the linked list.

Application :

To implement any kind of database. Manly used in manipulating data. Can be used to store information like sparse matrix, polynomials, stack, queue.

CONCLUSION:-