Data Structure and Algorithms - Linked List

If arrays accommodate similar types of data types, linked lists consist of elements with different data types that are also arranged sequentially.

But how are these linked lists created?

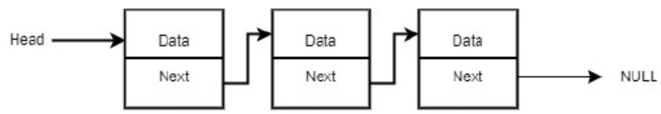
A linked list is a collection of “nodes” connected together via links. These nodes consist of the data to be stored and a pointer to the address of the next node within the linked list. In the case of arrays, the size is limited to the definition, but in linked lists, there is no defined size. Any amount of data can be stored in it and can be deleted from it.

There are three types of linked lists –

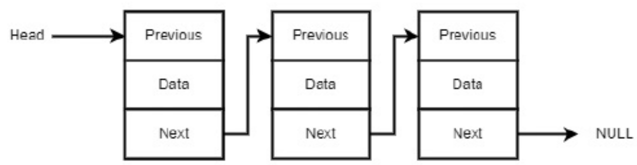
- **Singly Linked List** – The nodes only point to the address of the next node in the list.
- **Doubly Linked List** – The nodes point to the addresses of both previous and next nodes.
- **Circular Linked List** – The last node in the list will point to the first node in the list. It can either be singly linked or doubly linked.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



Single Linked List



Double Linked List

As per the above illustration, following are the important points to be considered.

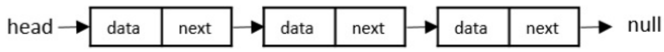
- Linked List contains a link element called first (head).
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

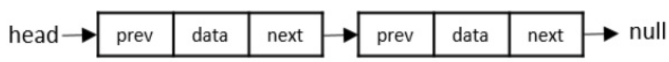
Singly Linked Lists

Singly linked lists contain two "buckets" in one node; one bucket holds the data and the other bucket holds the address of the next node of the list. Traversals can be done in one direction only as there is only a single link between two nodes of the same list.



Doubly Linked Lists

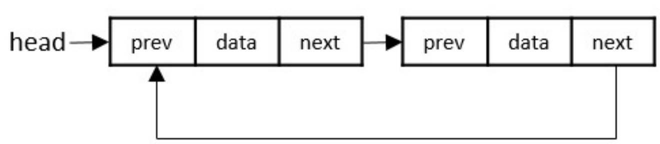
Doubly Linked Lists contain three “buckets” in one node; one bucket holds the data and the other buckets hold the addresses of the previous and next nodes in the list. The list is traversed twice as the nodes in the list are connected to each other from both sides.



Circular Linked Lists

Circular linked lists can exist in both singly linked list and doubly linked list.

Since the last node and the first node of the circular linked list are connected, the traversal in this linked list will go on forever until it is broken.



Basic Operations in the Linked Lists

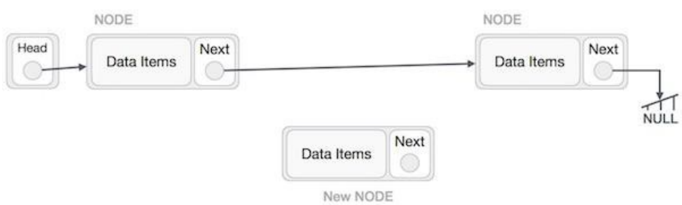
The basic operations in the linked lists are insertion, deletion, searching, display, and deleting an element at a given key. These operations are performed on Singly Linked Lists as given below –

- **Insertion** – Adds an element at the beginning of the list.

- **Deletion** – Deletes an element at the beginning of the list.
- **Display** – Displays the complete list.
- **Search** – Searches an element using the given key.
- **Delete** – Deletes an element using the given key.

Insertion Operation

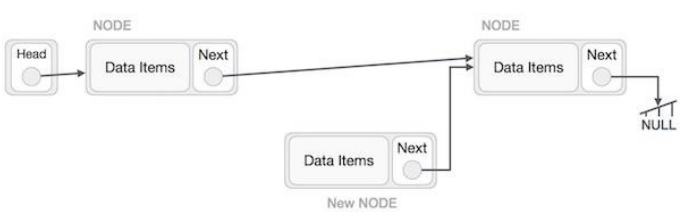
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B.next to C –

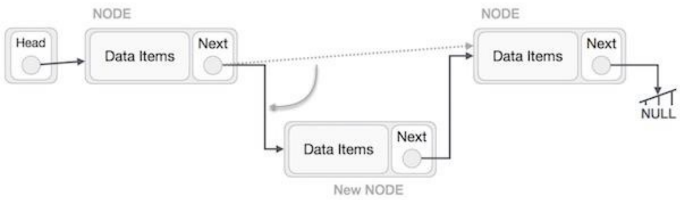
```
NewNode.next -> RightNode;
```

It should look like this –



Now, the next node at the left should point to the new node.

```
LeftNode.next -> NewNode;
```



This will put the new node in the middle of the two. The new list should look like this –

Insertion in linked list can be done in three different ways. They are explained as follows –

Insertion at Beginning

In this operation, we are adding an element at the beginning of the list.

Algorithm

```
1. START
2. Create a node to store the data
3. Check if the list is empty
4. If the list is empty, add the data to
5 If the list is not empty, add the data
6. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

// display the list
void printList(){
```

```
    struct node *p = head;
    printf("\n[");

    //start from the beginning
    while(p != NULL) {
       printf(" %d ",p->data);
       p = p->next;
    }
    printf("]");
}

//insertion at the beginning
void insertatbegin(int data){

    //create a link
    struct node *lk = (struct node*) mal
    lk->data = data;

    // point it to old first node
```

Output

```
Linked List:
[ 50  44  30  22  12 ]
```

Insertion at Ending

In this operation, we are adding an element at the ending of the list.

Algorithm

```
1. START
2. Create a new node and assign the data
3. Find the last node
4. Point the last node to new node
5. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```
   lk->next = head;

   //point first to new first node
   head = lk;
}
void insertatend(int data){

   //create a link
   struct node *lk = (struct node*) mal
   lk->data = data;
   struct node *linkedlist = head;

   // point it to old first node
   while(linkedlist->next != NULL)
      linkedlist = linkedlist->next;

   //point first to new first node
   linkedlist->next = lk;
}
void main(){
   int k=0;
   insertatbegin(12);
   insertatend(22);
   insertatend(30);
   insertatend(44);
   insertatend(50);
   printf("Linked List: ");

   // print list
   printList();
}
```

Output

```
Linked List:
[ 12 22 30 44 50 ]
```

Insertion at a Given Position

In this operation, we are adding an element at any position within the list.

Algorithm

```
1. START
2. Create a new node and assign data to i
3. Iterate until the node at position is
4. Point first to new first node
5. END
```

Example

Following are the implementations of this operation in various programming languages –

C C++ Java Python

```
}

//insertion at the beginning
void insertatbegin(int data){

   //create a link
   struct node *lk = (struct node*) ma
   lk->data = data;
```

```
    printList();
}
```

Output

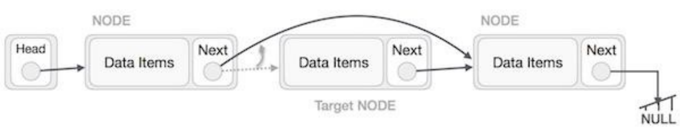
```
Linked List:
[ 22 12 30 ]
```

Deletion Operation

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.

The left (previous) node of the target node now should point to the next node of the target node −

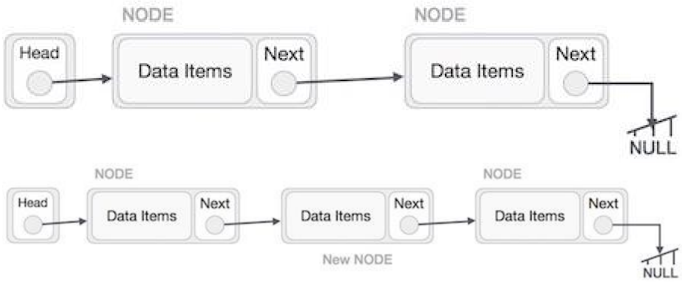
```
LeftNode.next −> TargetNode.next;
```



This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
TargetNode.next −> NULL;
```

We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.



Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

Deletion in linked lists is also performed in three different ways. They are as follows –

Deletion at Beginning

In this deletion operation of the linked, we are deleting an element from the beginning of the list. For this, we point the head to the second node.

Algorithm

```
1. START
2. Assign the head pointer to the next no
3. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java

```
    //point first to new first node
    head = lk;
}
void deleteatbegin(){
    head = head->next;
}
void main(){
    int k=0;
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    insertatbegin(40);
    insertatbegin(55);
    printf("Linked List: ");

    // print list
    printList();
    deleteatbegin();
    printf("\nLinked List after deletior

    // print list
    printList();
}
```

Output

```
Linked List:
[ 55  40  30  22  12 ]
Linked List after deletion:
[ 40  30  22  12 ]
```

Deletion at Ending

In this deletion operation of the linked, we are deleting an element from the ending of the list.

Algorithm

```
1. START
2. Iterate until you find the second last
3. Assign NULL to the second last element
```

4. END

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java

```
   lk->data = data;

   // point it to old first node
   lk->next = head;

   //point first to new first node
   head = lk;
}
void deleteatend(){
   struct node *linkedlist = head;
   while (linkedlist->next->next != NUI
      linkedlist = linkedlist->next;
   linkedlist->next = NULL;
}
void main(){
   int k=0;
   insertatbegin(12);
   insertatbegin(22);
   insertatbegin(30);
   insertatbegin(40);
   insertatbegin(55);
   printf("Linked List: ");

   // print list
   printList();
   deleteatend();
   printf("\nLinked List after deletior

   // print list
   printList();
}
```

Output

```
Linked List:
[ 55  40  30  22  12 ]
Linked List after deletion:
[ 55  40  30  22 ]
```

Deletion at a Given Position

In this deletion operation of the linked, we are deleting an element at any position of the list.

Algorithm

```
1. START
2. Iterate until find the current node at
3. Assign the adjacent node of current no
4. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java

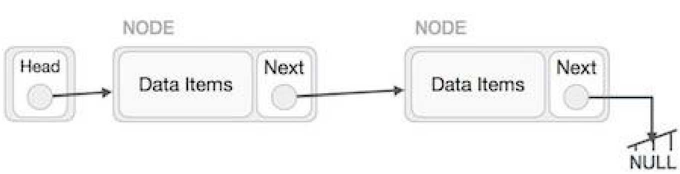
```
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    insertatbegin(40);
    insertatbegin(55);
    printf("Linked List: ");

    // print list
    printList();
    deletenode(30);
    printf("\nLinked List after deletior

    // print list
    printList();
}
```

Output

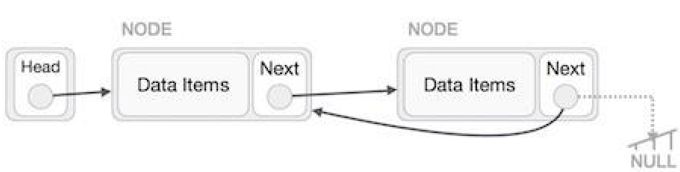
```
Linked List:
[ 55  40  30  22  12 ]
Linked List after deletion:
[ 55  40  22  12 ]
```

Reverse Operation

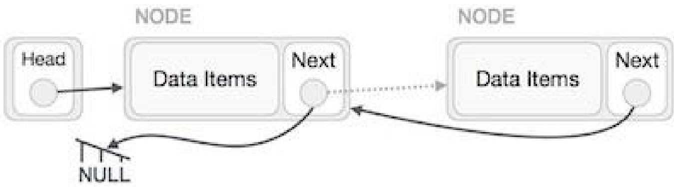
This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list.



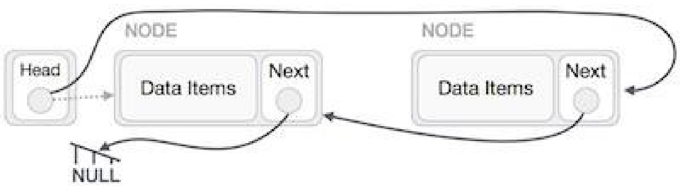
First, we traverse to the end of the list. It should be pointing to NULL. Now, we shall make it point to its previous node –



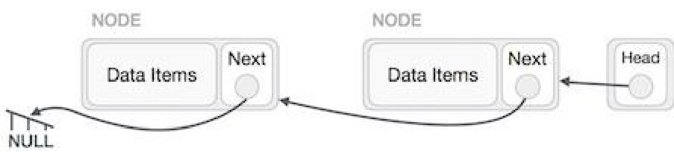
We have to make sure that the last node is not the last node. So we'll have some temp node, which looks like the head node pointing to the last node. Now, we shall make all left side nodes point to their previous nodes one by one.



Except the node (first node) pointed by the head node, all nodes should point to their predecessor, making them their new successor. The first node will point to NULL.



We'll make the head node point to the new first node by using the temp node.



Algorithm

Step by step process to reverse a linked list is as follows −

```
1 START
2. We use three pointers to perform the r
3. Point the current node to head and ass
4. Iteratively repeat the step 3 for all
5. Assign head to the prev node.
```

Example

Following are the implementations of this operation in various programming languages −

C | C++ | Java | Python

```
   // point it to old firs
   lk->next = head;

   //point first to new first node
   head = lk;
}
void reverseList(struct node** head){
   struct node *prev = NULL, *cur=*heac
   while(cur!= NULL) {
      tmp = cur->next;
      cur->next = prev;
      prev = cur;
      cur = tmp;
   }
   *head = prev;
}
void main(){
   int k=0;
   insertatbegin(12);
   insertatbegin(22);
   insertatbegin(30);
   insertatbegin(40);
   insertatbegin(55);
   printf("Linked List: ");

   // print list
   printList();
   reverseList(&head);
   printf("\nReversed Linked List: ");
   printList();
}
```

Output

```
Linked List:
[ 55  40  30  22  12 ]
Reversed Linked List:
[ 12  22  30  40  55 ]
```

Search Operation

Searching for an element in the list using a key element. This operation is done in the same way as array search; comparing every element in the list with the key element given.

Algorithm

```
START
If the list is not empty, iteratively cl
If the key element is not present in th
END
```

Example

Following are the implementations of this operation in various programming languages –

C C++ Java Python

```
        printf("\nElement is found");
    else
        printf("\nElement is not present
```

Output

```
Linked List:
[ 55  40  30  22  12 ]
Element is found
```

Traversal Operation

The traversal operation walks through all the elements of the list in an order and displays the elements in that order.

Algorithm

```
1. START
2. While the list is not empty and did no
3. END
```

Example

Following are the implementations of this operation in various programming languages –

C C++ Java Python

```
    lk->data = data;

    // point it to old first node
    lk->next = head;

    //point first to new first node
    head = lk;
}
void main(){
    int k=0;
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    printf("Linked List: ");

    // print list
    printList();
}
```

Output

```
Linked List:
[ 30  22  12 ]
```

Implementation of Linked Lists

Let us look at the implementation of the linked list data structures below –

To know more about the linked list implementation of Linked Lists in C programming language, **click here.**

For the complete implementation of a Singly Linked List in C++ programming language, **click here.**

To learn more about the Java implementation of the Singly Linked Lists, **click here.**

To see the Python Implementation of the Singly Linked Lists, **click here.**

Print Page