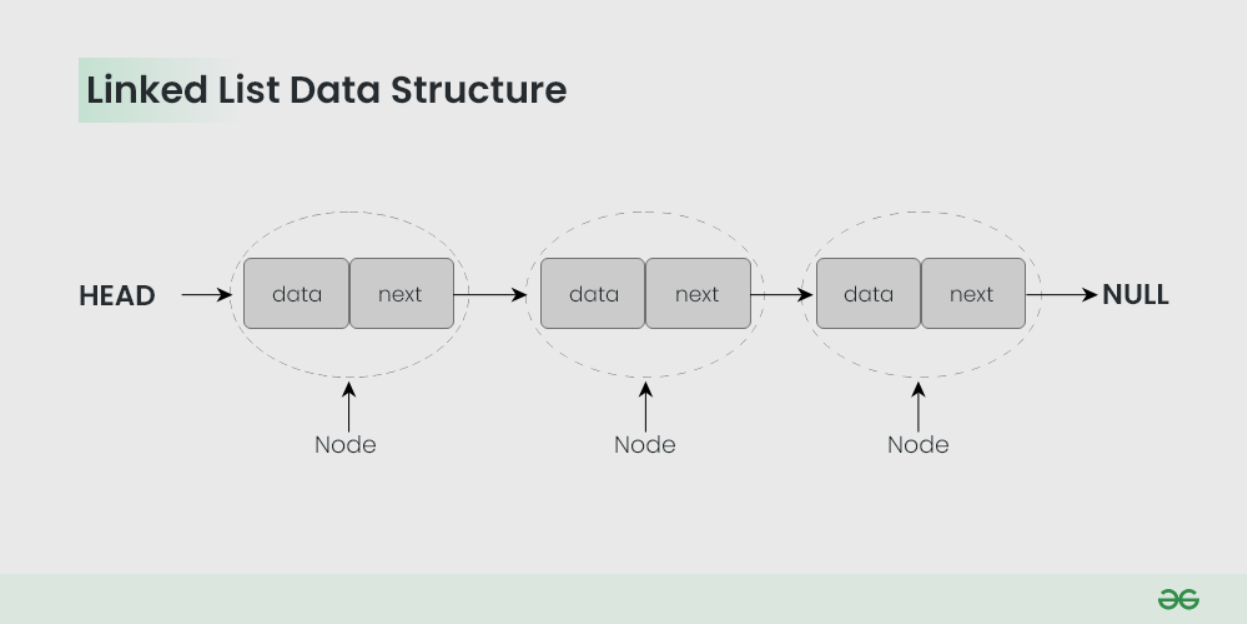
A linked list is a collection of “nodes” connected together via links. These nodes consist of the data to be stored and a pointer to the address of the next node within the linked list. In the case of arrays, the size is limited to the definition, but in linked lists, there is no defined size. Any amount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



There are three types of linked lists −

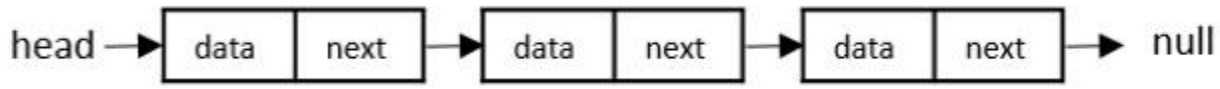
- **Singly Linked List** − The nodes only point to the address of the next node in the list.
- **Doubly Linked List** − The nodes point to the addresses of both previous and next nodes.
- **Circular Linked List** − The last node in the list will point to the first node in the list. It can either be singly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

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.



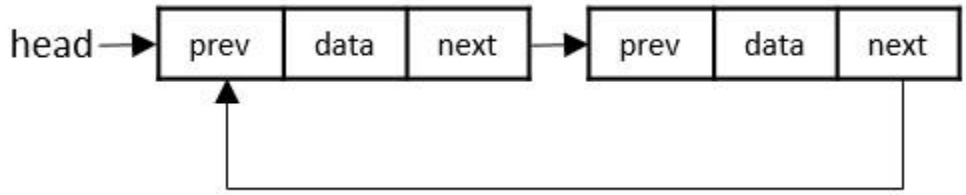
Doubly Linked Lists

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.



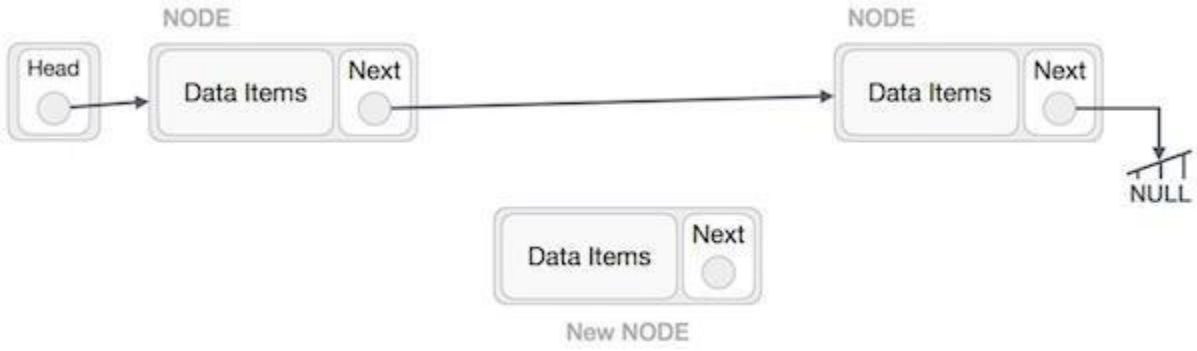
Basic Operations in the Linked Lists

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

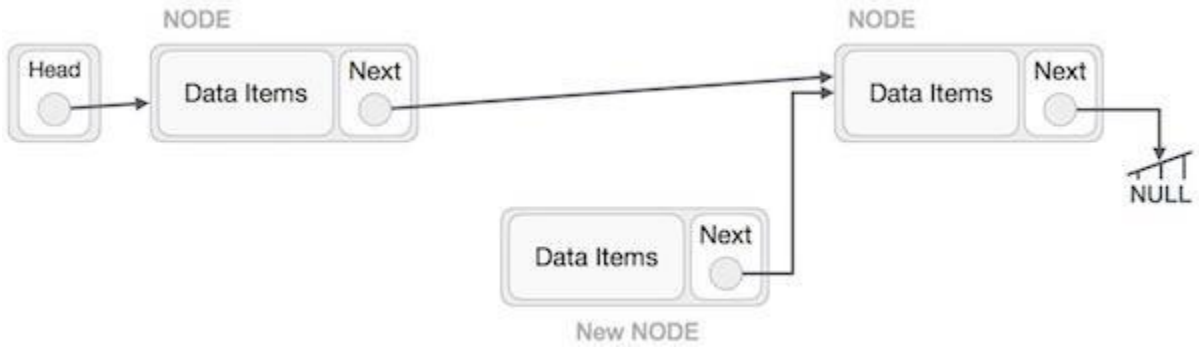
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



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

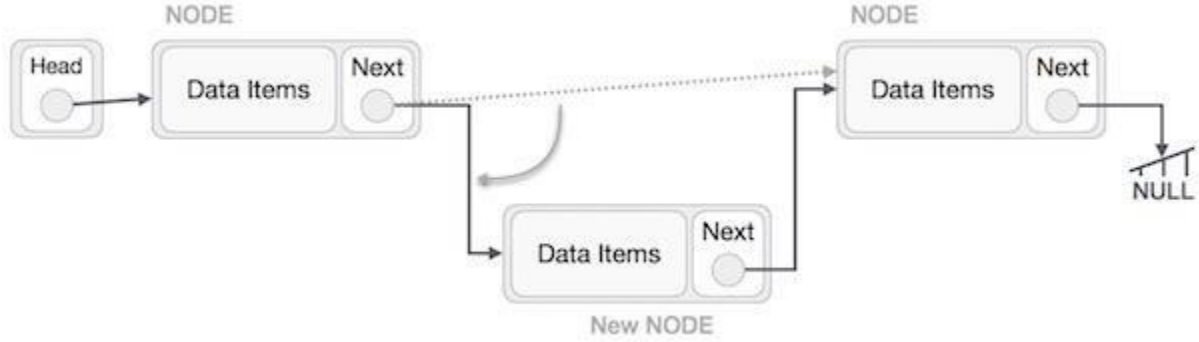
NewNode.next −> RightNode;

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 the node and assign the head pointer to it.
5 If the list is not empty, add the data to a node and link to the current head. Assign the head to the newly added node.
6. END

```
#include <bits/stdc++.h>
#include <string>
```

```
using namespace std;
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

// display the list
void printList(){
   struct node *p = head;
   cout << "\n[";

   //start from the beginning
   while(p != NULL) {
      cout << " " << p->data << " ";
      p = p->next;
   }
   cout << "]";
}

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

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

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

   //point first to new first node
   head = lk;
}
int main(){
   insertatbegin(12);
   insertatbegin(22);
   insertatbegin(30);
   insertatbegin(44);
   insertatbegin(50);
   cout << "Linked List: ";

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

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

```
#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*) malloc(sizeof(struct node));
   lk->data = data;

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

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

```
   //create a link
   struct node *lk = (struct node*) malloc(sizeof(struct node));
   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:
[50 30 12 22 44]

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 it
3. Iterate until the node at position is found
4. Point first to new first node
5. END

```
#include <bits/stdc++.h>
#include <string>
using namespace std;
struct node {
   int data;
   struct node *next;
};
```

```
struct node *head = NULL;
struct node *current = NULL;


// display the list
void printList(){
   struct node *p = head;
   cout << "\n[";

   //start from the beginning
   while(p != NULL) {
      cout << " " << p->data << " ";
      p = p->next;
   }
   cout << "]";
}

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

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

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

   //point first to new first node
   head = lk;
}
void insertafternode(struct node *list, int data){
   struct node *lk = (struct node*) malloc(sizeof(struct node));
   lk->data = data;
   lk->next = list->next;
   list->next = lk;
}
int main(){
   insertatbegin(12);
   insertatbegin(22);
   insertatbegin(30);
   insertafternode(head->next,44);
   insertafternode(head->next->next, 50);
   cout << "Linked List: ";

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

Output

Linked List:
[30 22 44 50 12]

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.

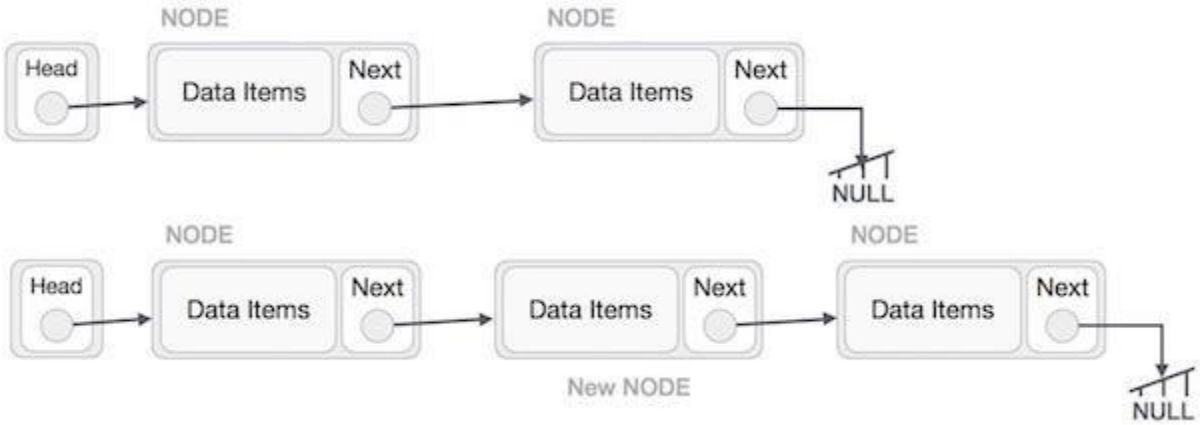
The left (previous) node of the target node now should point to the next node of the target node −

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 node in the list
3. END

```
#include <bits/stdc++.h>
#include <string>
using namespace std;
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

// display the list
void printList(){
   struct node *p = head;
   cout << "\n[";

   //start from the beginning
   while(p != NULL) {
      cout << " " << p->data << " ";
      p = p->next;
   }
   cout << "]";
}

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

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

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

   //point first to new first node
   head = lk;
}
```

```
void deleteatbegin(){
  head = head->next;
}
int main(){
  insertatbegin(12);
  insertatbegin(22);
  insertatbegin(30);
  insertatbegin(44);
  insertatbegin(50);
  cout << "Linked List: ";

  // print list
  printList();
  deleteatbegin();
  cout << "Linked List after deletion: ";
  printList();
}
```

Output

Linked List:
[50 44 30 22 12]
Linked List after deletion:
[44 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 element in the list.
3. Assign NULL to the second last element in the list.
4. END

```
#include <bits/stdc++.h>
#include <string>
using namespace std;
struct node {
  int data;
  struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

// Displaying the list
void printList(){
  struct node *p = head;
  while(p != NULL) {
    cout << " " << p->data << " ";
    p = p->next;
  }
}
```

```
// Insertion at the beginning
void insertatbegin(int data){

  //create a link
  struct node *lk = (struct node*) malloc(sizeof(struct node));
  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 != NULL)
    linkedlist = linkedlist->next;
  linkedlist->next = NULL;
}
int main(){
  insertatbegin(12);
  insertatbegin(22);
  insertatbegin(30);
  insertatbegin(44);
  insertatbegin(50);
  cout << "Linked List: ";

  // print list
  printList();
  deleteatend();
  cout << "\nLinked List after deletion: ";
  printList();
}
```

Output

Linked List: 50 44 30 22 12
Linked List after deletion: 50 44 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 position in the list
3. Assign the adjacent node of current node in the list to its previous node.
4. END

```
#include <bits/stdc++.h>
#include <string>
using namespace std;
struct node {
  int data;
  struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

// display the list
void printList(){
  struct node *p = head;
  cout << "\n[";

  //start from the beginning
  while(p != NULL) {
    cout << " " << p->data << " ";
    p = p->next;
  }
  cout << "]";
}

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

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

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

  //point first to new first node
  head = lk;
}
void deletenode(int key){
  struct node *temp = head, *prev;
  if (temp != NULL && temp->data == key) {
    head = temp->next;
    return;
  }

  // Find the key to be deleted
  while (temp != NULL && temp->data != key) {
    prev = temp;
    temp = temp->next;
  }

  // If the key is not present
  if (temp == NULL) return;

  // Remove the node
```

```
   prev->next = temp->next;
}
int main(){
   insertatbegin(12);
   insertatbegin(22);
   insertatbegin(30);
   insertatbegin(44);
   insertatbegin(50);
   cout << "Linked List: ";

   // print list
   printList();
   deletenode(30);
   cout << "Linked List after deletion: ";
   printList();
}
```

Output

Linked List:
[50 44 30 22 12]Linked List after deletion:
[50 44 22 12]

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

1 START
2 If the list is not empty, iteratively check if the list contains the key
3 If the key element is not present in the list, unsuccessful search
4 END

```
#include <bits/stdc++.h>
#include <string>
using namespace std;
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

// display the list
void printList(){
   struct node *p = head;
   cout << "\n[";

   //start from the beginning
   while(p != NULL) {
```

```
    cout << " " << p->data << " ";
    p = p->next;
  }
  cout << "]";
}

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

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

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

  //point first to new first node
  head = lk;
}
int searchlist(int key){
  struct node *temp = head;
  while(temp != NULL) {
    if (temp->data == key) {
      return 1;
    }
    temp=temp->next;
  }
  return 0;
}
int main(){
  int k = 0;
  insertatbegin(12);
  insertatbegin(22);
  insertatbegin(30);
  insertatbegin(44);
  insertatbegin(50);
  cout << "Linked List: ";

  // print list
  printList();
  k = searchlist(16);
  if (k == 1)
    cout << "\nElement is found";
  else
    cout << "\nElement is not present in the list";
}
```

Output

Linked List:
[50 44 30 22 12]
Element is not present in the list