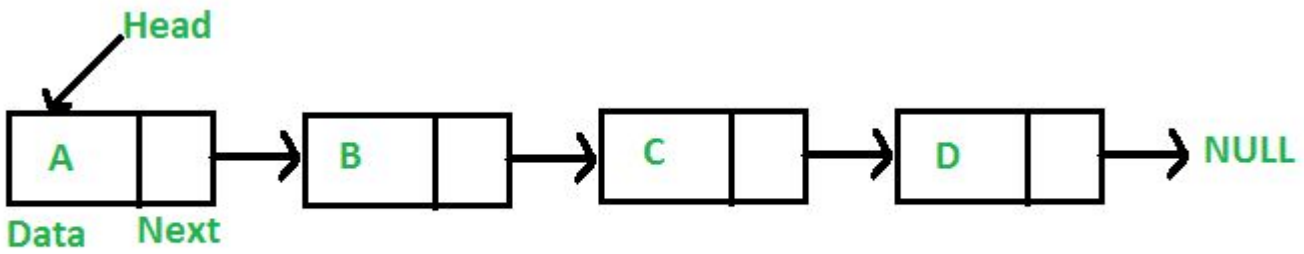
Linked List

Linked List

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.

2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

Advantages of linked list over arrays

1) Dynamic size

2) Ease of insertion/deletion

Drawbacks of linked list:

- Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.
- Extra memory space for a pointer is required with each element of the list.

Representation:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.

Each node in a list consists of at least two parts:

1) data

2) Pointer (Or Reference) to the next node

In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

```
struct Node {

        int data;

        struct Node* next;

        };
```

Program to create a simple linked list with 3 nodes

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

      };

int main(){

    struct Node* head = NULL;

    struct Node* second = NULL;

    struct Node* third = NULL;
```

```
head = (struct Node*)malloc(sizeof(struct Node));

second = (struct Node*)malloc(sizeof(struct Node));

third = (struct Node*)malloc(sizeof(struct Node));

head->data = 1; // assign data in first node

head->next = second; // Link first node with the second node

second->data = 2

second->next = third;

third->data = 3;

third->next = NULL;

return 0;

}
```

Linked List Traversal

```
void printList(struct Node* n)

{    while (n != NULL) {

        printf(" %d ", n->data);

        n = n->next;    }

}
```

Inserting a node to linked list

A node can be added in three ways

1) At the front of the linked list

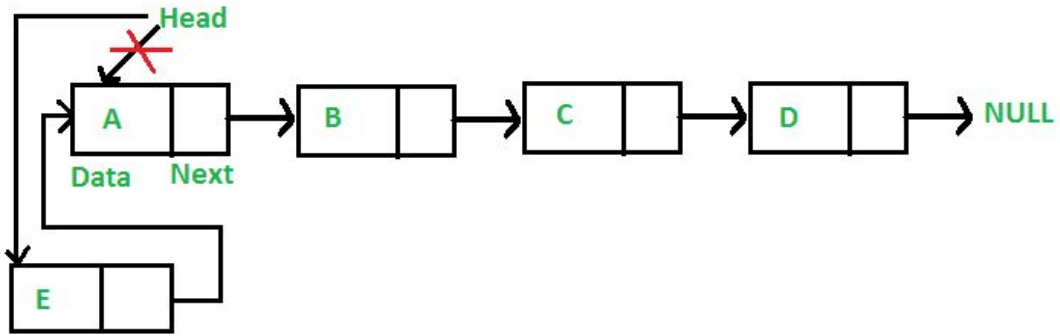
2) After a given node.

3) At the end of the linked list.

Add a node at the front: (A 4 steps process)

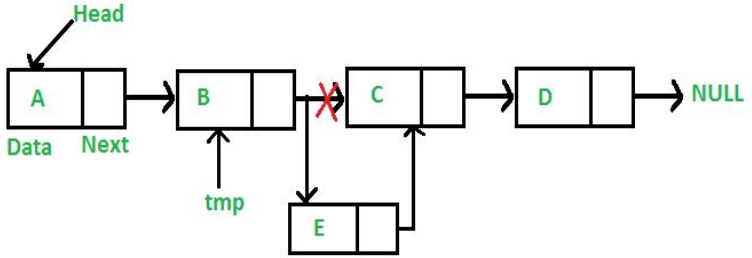
Here, newly added node becomes the new head of the Linked List.

```
void push(struct Node* head, int new_data)

{

    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    new_node->data  = new_data;

    new_node->next = head;

    head = new_node;

}
```



Add a node after a given node: (5 steps process)

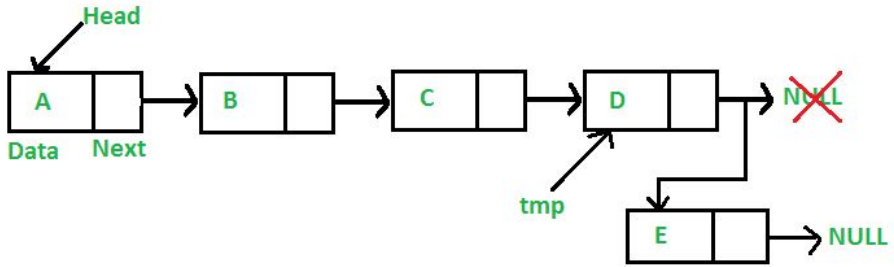
We are given pointer to a node, and the new node is inserted after the given node.




```
void insertAfter(struct Node* prev_node, int new_data)
{    if (prev_node == NULL)     {
       printf("the given previous node cannot be NULL");
       return;      }
     struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));
     new_node->data  = new_data;
     new_node->next = prev_node->next;
     prev_node->next = new_node;   }
```

Add a node at the end: (6 steps process)

```
void append(struct Node* head, int new_data)

{    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

     struct Node *last = head;  // used in step 5

     new_node->data  = new_data;

     new_node->next = NULL;

     if (head == NULL) { head = new_node;

                                 return;  }

      while (last->next != NULL)

                                last = last->next;

      last->next = new_node;

     return; }
```



Deleting a node

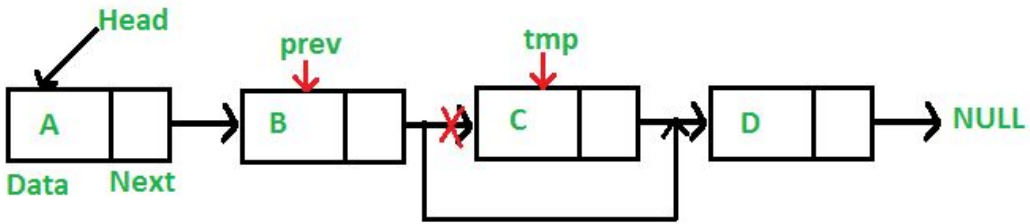
Given a 'key', delete the first occurrence of this key in linked list.

To delete a node from linked list, we need to do following steps.

1) Find previous node of the node to be deleted.

2) Change the next of previous node.

3) Free memory for the node to be deleted.



```
void deleteNode(struct Node *head, int key)
{    struct Node* temp = head, prev;
     if (temp != NULL && temp->data == key)   {
        head = temp->next;
        free(temp);
    return;    }
    while (temp != NULL && temp->data != key)
    {   prev = temp;
        temp = temp->next;    }
    if (temp == NULL) return; // If key was not present in linked list
    prev->next = temp->next;
    free(temp);  // Free memory
}
```