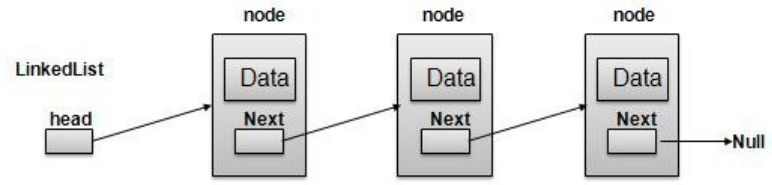
Working With Linked List

Linked List - A Self referential structure where items are arranged sequentially

Characteristics :

1. Unlike arrays the size need not be specified at the beginning
2. Operations on the list such as addition, deletion or shift of elements is easy as compared to arrays



Structure 1 **Structure 2** **Structure 3**

- Each item is part of a structure
- The structure also contains a pointer or link to the next structure
- Each structure of the list is called **Node**
- Node contains two fields :
    - one containing the data item
    - the other containing the address of the next item (a pointer)
- The data items comprising the linked list need not be contiguous
- They are ordered by logical links which are part of the data in the structure
- The link is a pointer to another structure of the same type

Structure to represent a node :

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

A more generic representation of a node:

```
struct node_name
{

     type member1;
     type member2;
     .....
     struct node_name *next;
}
```

Example 1 :

Consider the structure :

```
struct stud
{
     int roll;
     char name[30];
     int age;

     struct stud *next;
}
```

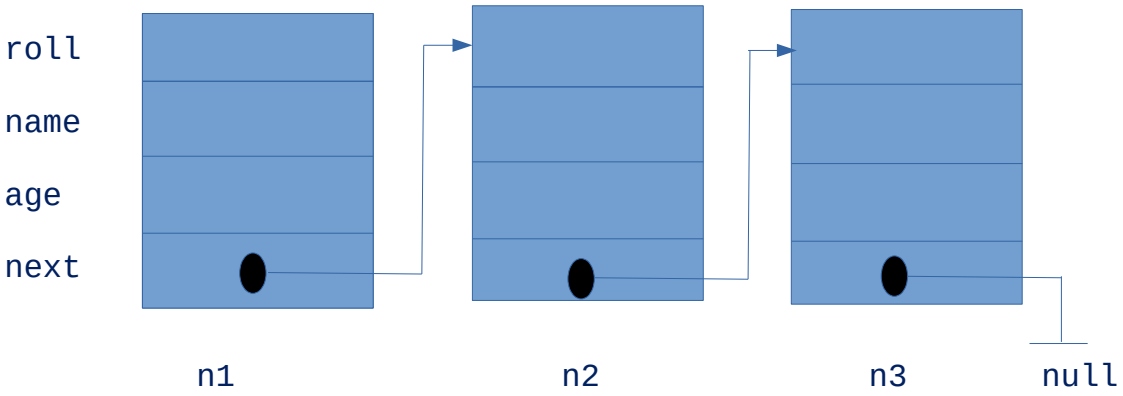
Assume the list contains three elements : n1, n2 and n3

```
struct stud n1,n2,n3;
```

To create the links :

```
n1.next = &n2;
n2.next = &n3;
n3.next = NULL; /* No more nodes follow */
```

List looks like :



null pointer indicates that no nodes follow

Traversing the list and print the items:

```
p = &n1;   /* a temporary pointer pointing to 1st element */

while (p != NULL)
{
     printf ("\n %d %s %d", p->roll, p->name, p->age);
     p = p->next;
}
```

Putting it all together:

```
#include <stdio.h>
struct stud
{
     int roll;
     char name[30];
     int age;
     struct stud *next;
}

int main()
{
     struct stud n1, n2, n3;
     struct stud *p;
     scanf (”%d %s %d”, &n1.roll, n1.name, &n1.age);
     scanf (”%d %s %d”, &n2.roll, n2.name, &n2.age);
     scanf (”%d %s %d”, &n3.roll, n3.name, &n3.age);

     n1.next = &n2;
     n2.next = &n3;
     n3.next = NULL;

/* Now traverse the list and print the elements */

     p = &n1; /* point to 1st element */

     while (p != NULL)
     {
          printf (”\n %d %s %d”, p->roll, p->name, p->age);
          p = p->next;
     }
}
```

To traverse a linked list, we need the first element of the list. Thus we use a pointer to the first element of the list and is known as the Head

Passing a list to a function requires passing the Head pointer to the function. For example, if we want to write a function to traverse a list, the function prototype would be :

```
traverse (struct stud *head)
```

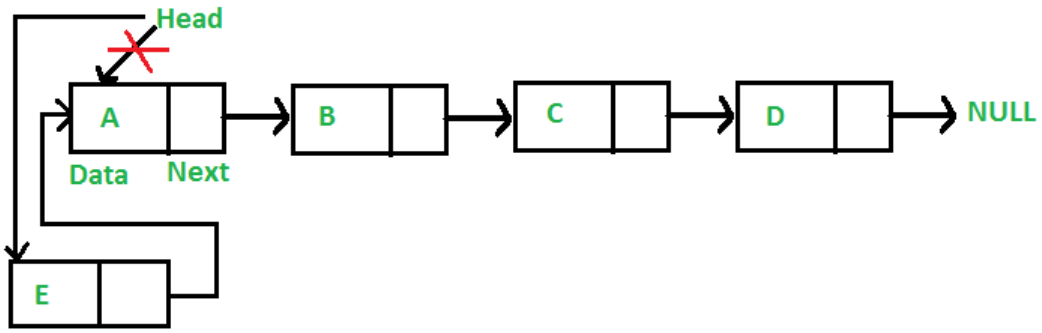
Example 2 : Function to carry out traversal of linked list

```
#include <stdio.h>

     struct stud
     {
          int roll;
          char name[30];
          int age;
          struct stud *next;
     }

     void traverse (struct stud *head)
     {
          while (head != NULL)
          {
               printf (”\n %d %s %d”, head->roll, head->name,
               head->age);
               head = head->next;
          }
     }

     int main()
     {
          struct stud n1, n2, n3, *p;
          scanf(”%d %s %d”, &n1.roll, n1.name, &n1.age);
          scanf(”%d %s %d”, &n2.roll, n2.name, &n2.age);

          scanf(”%d %s %d”, &n3.roll, n3.name, &n3.age);
          n1.next = &n2;
          n2.next = &n3;
          n3.next = NULL;
          p = &n1;
          traverse (p);
     }
```

Operation on Linked List :

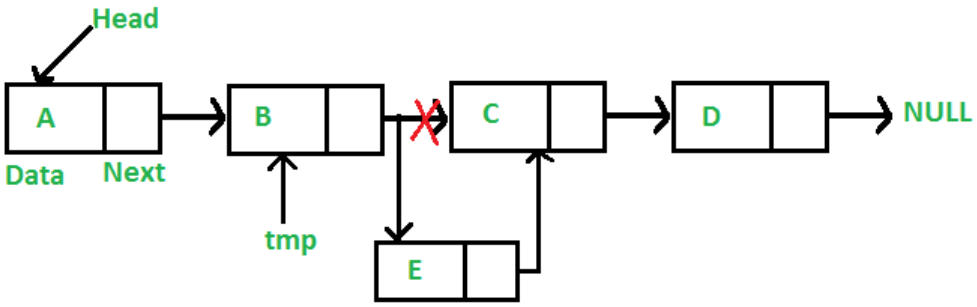
1. Insertion -

i. at the front :




```
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_ref); //Make next of new node as head//

  (*head_ref)    = new_node;//Move head to point to the new node//
}
```

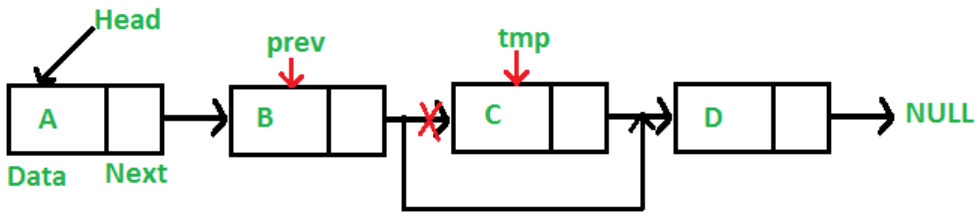
ii. after a given node



```
void insertAfter(struct Node* prev_node, int new_data)
{

/*1. check if the given prev_node is NULL */

if (prev_node == NULL) {
     printf("the given previous node cannot be NULL");
     return;
}

/* 2. allocate new node */

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

/* 3. put in the data */

new_node->data = new_data;

/* 4. Make next of new node as next of prev_node */

new_node->next = prev_node->next;

/* 5. move the next of prev_node as new_node */

prev_node->next = new_node;
}
```

2. Deletion:

To delete a node from the linked list, we need to do the following steps.
1) Find the previous node of the node to be deleted.
2) Change the next of the previous node.
3) Free memory for the node to be deleted.



```
// A complete working C program to demonstrate deletion in singly
linked list

#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct Node
{
     int data;
     struct Node* next;
};

/* Given a reference (pointer to pointer) to the head of a
list and an int, inserts a new node on the front of the
list. */

void push(struct Node** head_ref, int new_data)
{
     struct Node* new_node
          = (struct Node*)malloc(sizeof(struct Node));
     new_node->data = new_data;
     new_node->next = (*head_ref);
     (*head_ref) = new_node;
}

/* Given a reference (pointer to pointer) to the head of a
list and a key, deletes the first occurrence of key in
linked list */

void deleteNode(struct Node** head_ref, int key)
{
     // Store head node
     struct Node *temp = *head_ref, *prev;

     // If head node itself holds the key to be deleted
     if (temp != NULL && temp->data == key) {
          *head_ref = temp->next; // Changed head
          free(temp); // free old head
          return;
     }

// Search for the key to be deleted, keep track of the
// previous node as we need to change 'prev->next'

while (temp != NULL && temp->data != key) {
          prev = temp;
          temp = temp->next;
}
```

```
// If key was not present in linked list
     if (temp == NULL)
          return;

     // Unlink the node from linked list
     prev->next = temp->next;

     free(temp); // Free memory
}

// This function prints contents of linked list starting
// from the given node
void printList(struct Node* node)
{
     while (node != NULL) {
          printf(" %d ", node->data);
          node = node->next;
     }
}

// Driver code
int main()
{
     /* Start with the empty list */
     struct Node* head = NULL;

     push(&head, 7);
     push(&head, 1);
     push(&head, 3);
     push(&head, 2);

     puts("Created Linked List: ");
     printList(head);
     deleteNode(&head, 1);
     puts("\nLinked List after Deletion of 1: ");
     printList(head);
     return 0;
}
```

Output

```
Created Linked List:
 2  3  1  7

Linked List after Deletion of 1:
 2  3  7
```