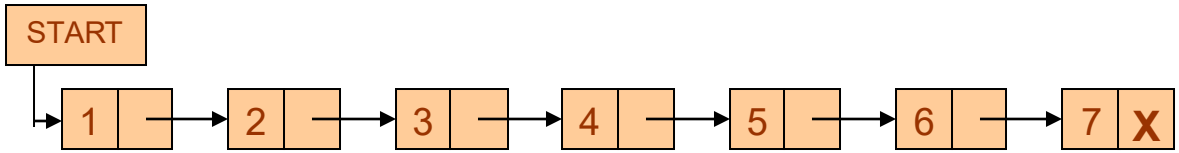
Linked List

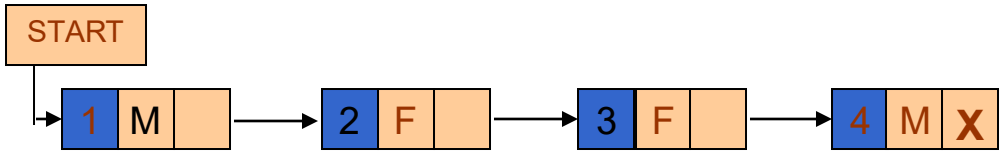
*A **linked list** is a linear collection of data elements called **nodes** in which linear representation is given by **links** from one node to the next node.*

*Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as **building block to implement data structures like stacks, queues** and their variations.*

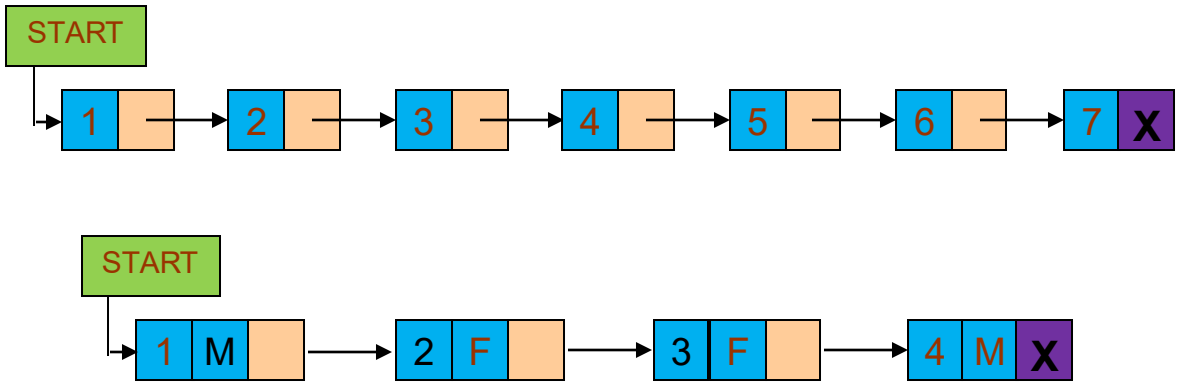
Linked List

A linked list can be perceived as a train or a sequence of nodes in which ***each node contains one or more data fields and a pointer to the next node.***





Linked List



The **left part** of the node which contains **data**

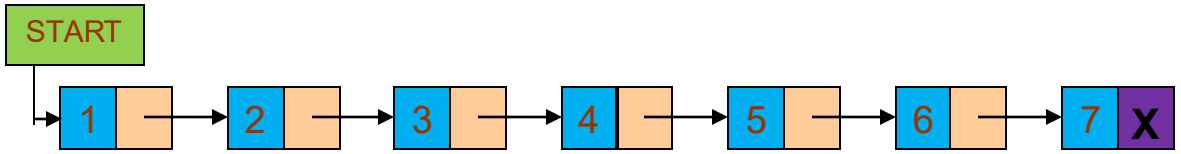
The **right part** of the node contains **a pointer to the next node**

The **last node** will have no next node connected to it, so it will store a special value called **NULL**.

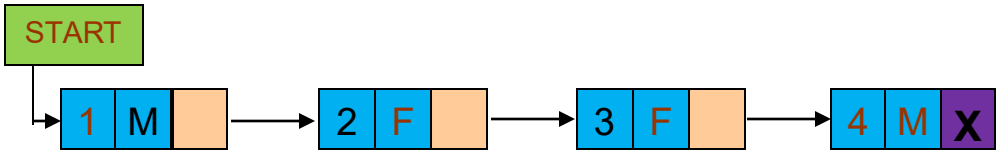
Linked List

NODE :

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

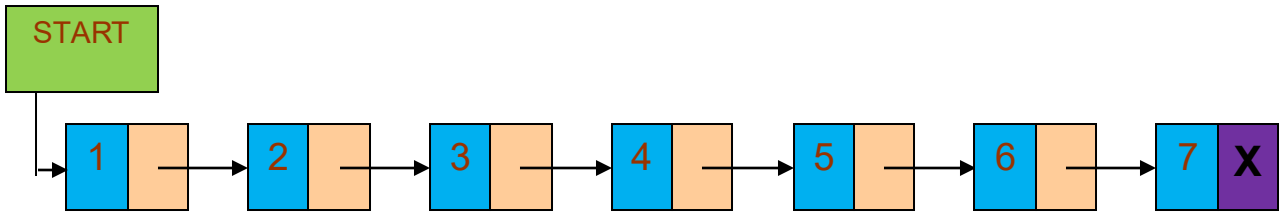



```
struct node
{
    int i;
    struct node *next;
};
```




```
struct node
{
    int i;
    char ch;
    struct node *next;
};
```

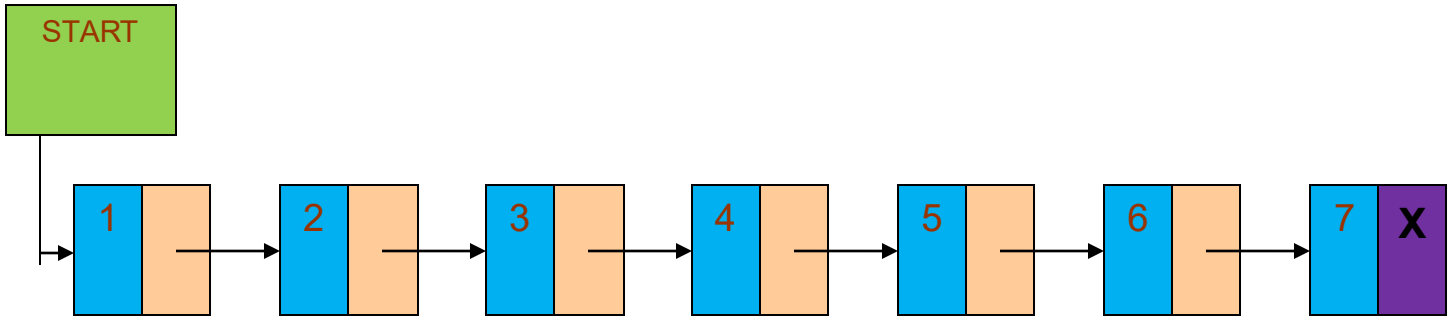
Linked List



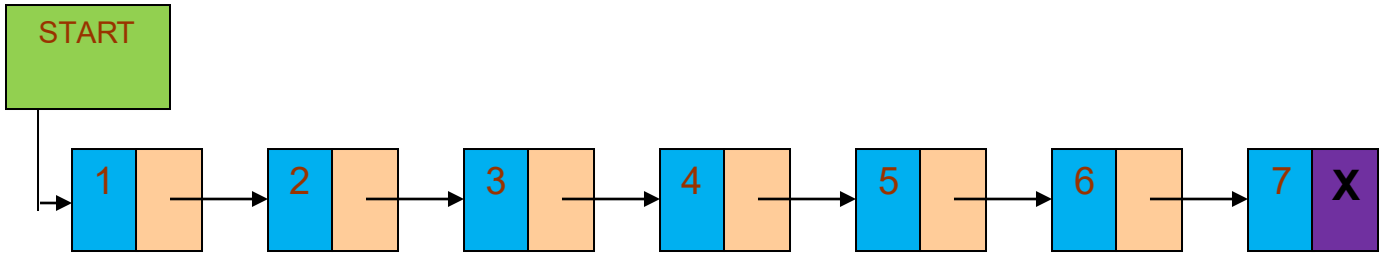
- We can traverse the entire linked list using a single pointer variable called **START**
- The **START** node contains the address of the first node
- The next part of the first node in turn stores the address of its succeeding node
- If **START = NULL**, this means that the linked list is empty and contains no nodes
- If **AVAIL = NULL**, this means that the memory not available so overflow condition

Singly Linked List (SLL)

- A **singly linked list** is the simplest type of linked list in which every node contains **some data and a pointer to the next node** of the same data type



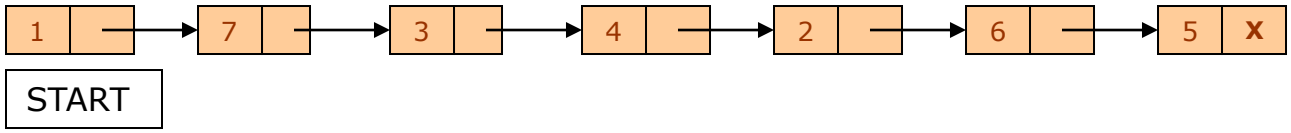
SLL - TRAVERSING A LINKED LIST

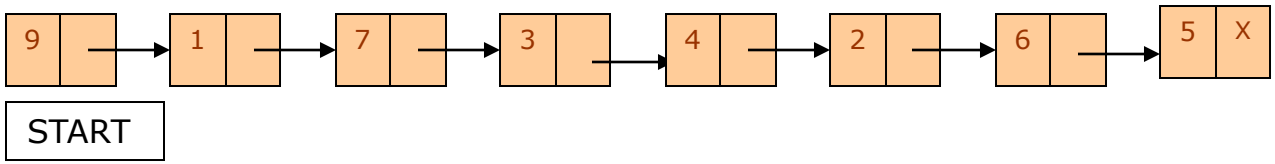


ALGORITHM FOR TRAVERSING A LINKED LIST

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:              Apply Process to PTR  ->  DATA
Step 4:              SET PTR = PTR  ->  NEXT
        [END OF LOOP]
Step 5: EXIT
```

SLL - INSERTING A NODE AT THE BEGINNING

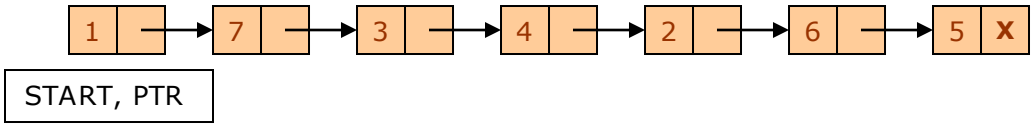


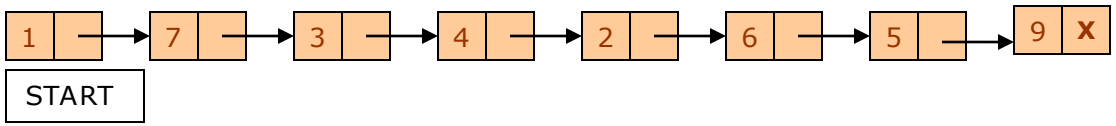


ALGORITHM TO INSERT A NEW NODE IN THE BEGINNING OF THE LINKED LIST

```
Step 1: IF AVAIL = NULL, then
                    Write OVERFLOW
                    Go to Step 6
        [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET New_Node  ->  DATA = VAL
Step 4: SET New_Node  ->  Next = START
Step 5: SET START = New_Node
Step 6: EXIT
```

SLL - INSERTING A NODE AT THE END

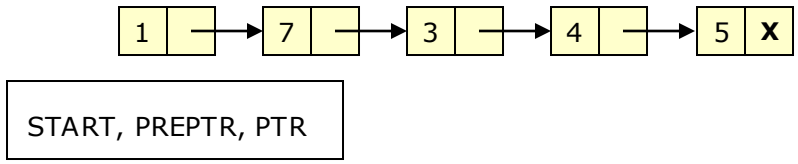


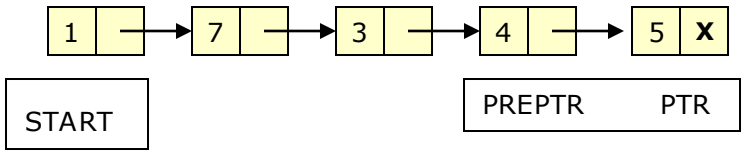


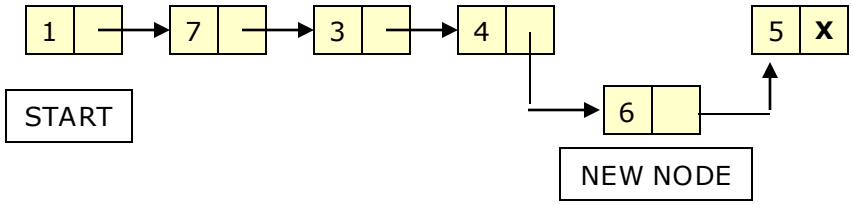
ALGORITHM TO INSERT A NEW NODE AT THE END OF THE LINKED LIST

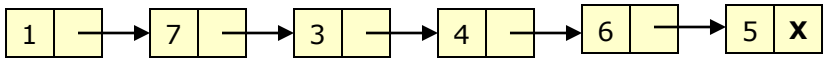
```
Step 1: IF AVAIL = NULL, then
                    Write OVERFLOW
                    Go to Step 9
          [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET New_Node -> DATA = VAL
Step 4: SET New_Node -> Next = NULL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != NULL
Step 7:             SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = New_Node
Step 9: EXIT
```

SLL - INSERTING A NODE AFTER NODE THAT HAS VALUE NUM









SLL - INSERTING A NODE AFTER NODE THAT HAS VALUE NUM

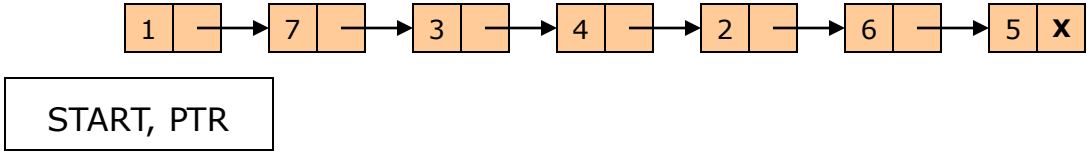
ALGORITHM TO INSERT A NEW NODE AFTER A NODE THAT HAS VALUE NUM

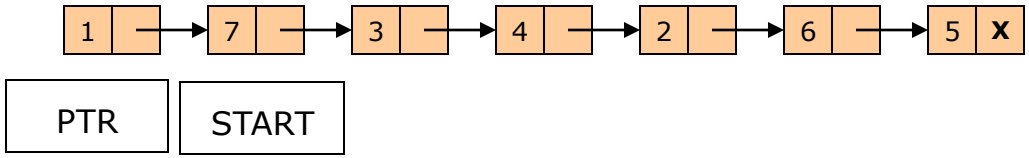
```
Step 1: IF AVAIL = NULL, then
                        Write OVERFLOW
                        Go to Step 12
        [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET New_Node - > DATA = VAL
Step 4: SET PTR = START
Step 5: SET PREPTR = PTR
Step 6: Repeat Steps 8 and 9 while PREPTR -> DATA  != NUM
Step 7:                 SET PREPTR = PTR
Step 8:                 SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: PREPTR -> NEXT = New_Node
Step 10: SET New_Node -> NEXT = PTR
Step 11: EXIT
```

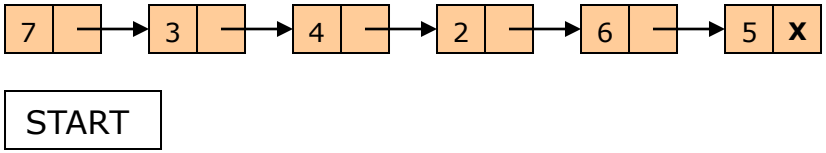
SLL - DELETING THE FIRST NODE

Algorithm to delete the first node from the linked list

```
Step 1: IF START = NULL, then
                    Write UNDERFLOW
                    Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```



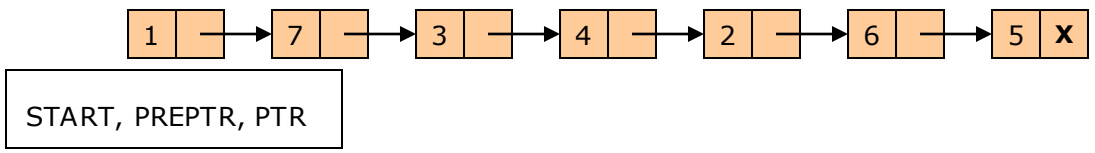


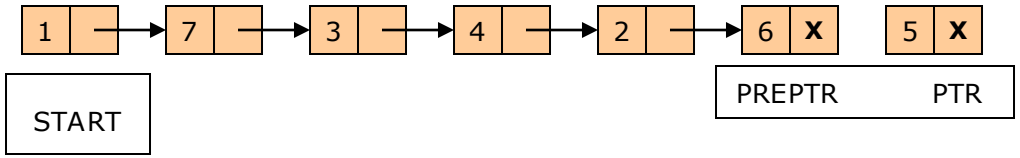


SLL - DELETING THE LAST NODE

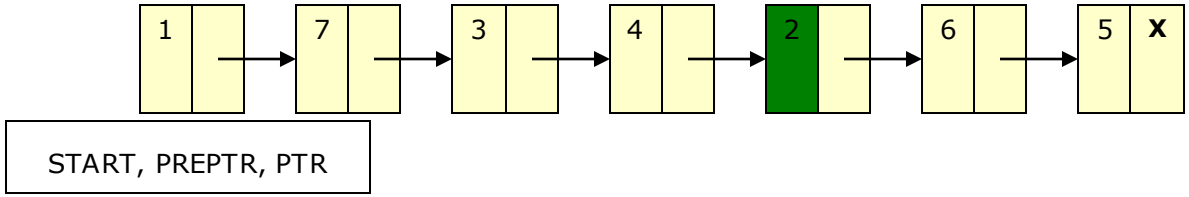
ALGORITHM TO DELETE THE LAST NODE OF THE LINKED LIST

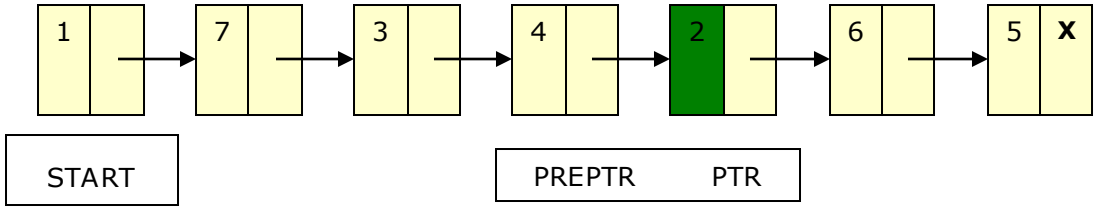
```
Step 1: IF START = NULL, then
                    Write UNDERFLOW
                    Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:             SET PREPTR = PTR
Step 5:             SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

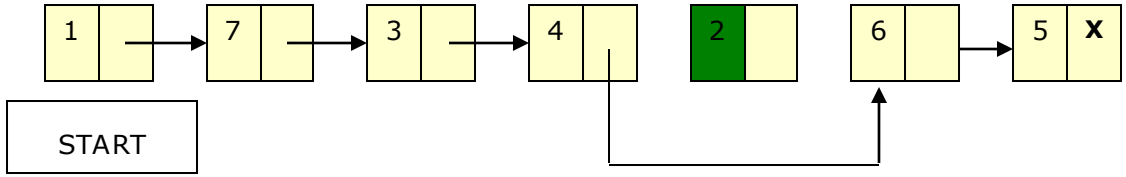


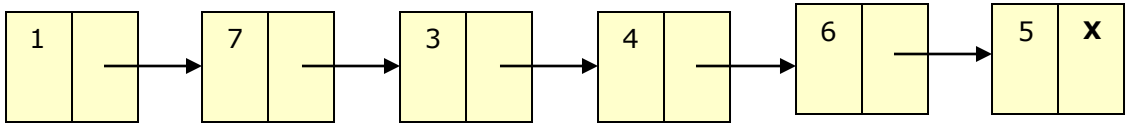


SLL - DELETING THE NODE WHOSE VALUE NUM









SLL - DELETING THE NODE AFTER A GIVEN NODE

ALGORITHM TO DELETE THE NODE AFTER A GIVEN NODE FROM THE LINKED LIST

```
Step 1: IF START = NULL, then
                    Write UNDERFLOW
                    Go to Step 10
          [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 and 6 while PREPTR -> DATA != NUM
Step 5:             SET PREPTR = PTR
Step 6:             SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 7: SET PREPTR -> NEXT = PTR -> NEXT
Step 8: FREE PTR
Step 9: EXIT
```

SLL – C Program

SINGLY LINKED LIST

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

struct node
{
    int data;
    struct node *next;
};

struct node *start = NULL;
```

SINGLY LINKED LIST

```
void create();
void display();
void insert_beg();
void insert_end();
void insert_middle();
void delete_beg();
void delete_end();
void delete_middle();
```

SINGLY LINKED LIST

```
int main()
{
    int option;
    do
    {
        printf("\n\n *****MAIN MENU *****");
        printf("\n 1: Create a list");
        printf("\n 2: Display the list");
        printf("\n 3: Add a node at the beginning");
        printf("\n 4: Add a node at the end");
        printf("\n 5: Add a node in the middle");
        printf("\n 6: Delete a node from the beginning");
        printf("\n 7: Delete a node from the end");
        printf("\n 8: Delete a node in the middle");
        printf("\n 9: EXIT");
        printf("\n\n Enter your option : ");
        scanf("%d", &option);
        switch(option)
        {
            case 1: create();
                    break;
            case 2: display();
                    break;
            case 3: insert_beg( );
                    break;
            case 4: insert_end( );
                    break;
            case 5: insert_ middle( );
                    break;
            case 6: delete_beg( );
                    break;
            case 7: delete_end( );
                    break;
            case  8: delete_ middle( );
                     break;
            case  9: exit(0);
        }
     }while(option  < 9);
        return 0;
}
```

SINGLY LINKED LIST

```
void create()
{
    struct node *new_node;
    struct node *ptr;
    int val;
    printf("\n Enter –1 to end");
    printf("\n Enter the data : ");
    scanf("%d", &val);
    while(val != -1)
    {
        new_node = (struct node*)
              malloc(sizeof(struct node));
        new_node -> data = val;
        new_node -> next = NULL;
        if(start == NULL)
        {
            start = new_node;
        }
```

```
        else
            {
                ptr=start;
                while(ptr -> next != NULL)
                    ptr=ptr->next;
                ptr -> next = new_node;
            }
            printf("\n Enter the data : ");
            scanf("%d", &val);
        }
}
```

SINGLY LINKED LIST

```
void display()
{
    struct node *ptr;
    ptr = start;
    while(ptr  != NULL)
    {
        printf("\t %d", ptr -> data);
        ptr = ptr -> next;
    }
}
```

SINGLY LINKED LIST

```
 void insert_beg()
{
    struct node *new_node;
    int val;
    printf("\n Enter the data : ");
    scanf("%d", &val);
    new_node = (struct node *) malloc(sizeof(struct node));
    new_node -> data = val;
    new_node -> next = start;
    start = new_node;
}
```

SINGLY LINKED LIST

```
void insert_end()
{
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &val);
    new_node = (struct node *) malloc(sizeof(struct node));
    new_node -> data = val;
    new_node -> next = NULL;
    ptr = start;
    while(ptr -> next != NULL)
        ptr = ptr -> next;
    ptr -> next = new_node;
}
```

SINGLY LINKED LIST

```
void insert_middle()
{
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &val);
    printf("\n Enter the value after which the node has to be inserted : ");
    scanf("%d", &num);
    new_node = (struct node *) malloc(sizeof(struct node));
    new_node -> data = val;
    ptr = start;
    preptr = start;
    while(preptr -> data != num)
    {
        preptr = ptr;
        ptr = ptr -> next;
    }
    preptr -> next = new_node;
    new_node -> next = ptr;
}
```

SINGLY LINKED LIST

```
void delete_beg()
{
    struct node *ptr;
    ptr = start;
    start = start -> next;
    free(ptr);
}
```

SINGLY LINKED LIST

```
void delete_end()
{
    struct node *ptr, *preptr;
    ptr = start;
    while(ptr -> next != NULL)
    {
        preptr = ptr;
        ptr = ptr -> next;
    }
    preptr -> next = NULL;
    free(ptr);
}
```

SINGLY LINKED LIST

```
void delete_middle()
{
    struct node *ptr, *preptr;
    int num;
    printf("\n Enter the number after which the node has to be deleted : ");
    scanf("%d", &num);
    ptr = start;
    preptr = ptr;
    while(preptr -> data != num)
    {
            preptr = ptr;
            ptr = ptr -> next;
    }
    preptr -> next = ptr -> next;
    free(ptr);
}
```