

SINGLE LINKED LIST

What is Linked List?

When we want to work with an unknown number of data values, we use a linked list data structure to organize that data. The linked list is a linear data structure that contains a sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called "Node".

In computer science, **a linked list is one of the fundamental data Structures.**

A linked list consists of series of structure. This structure is not necessarily contiguous. Each structure contains one or more than one contiguous information fields and also containing its successor.

➤ Several different types of Linked List exists :

- [1] Singly linked list
- [2] Doubly linked list
- [3] Header linked list
- [4] Circular linked list

➤ Difference between Linked list and Array

Linked List	Array
Element inserted into linked list dynamically.	Array is a static.
It reduces memory wastage.	Many times increase memory wastage.
It allows only sequential access.	It allows random access using its index.
Some situation traversing becomes slow	Its traversing is faster as compare to Linked list.
It requires extra space to store reference of next node.	It does not required any extra space

What is Single Linked List?

Simply a list is a sequence of data, and the linked list is a sequence of data linked with each other.

The formal definition of a single linked list is as follows...

Single linked list is a sequence of elements in which every element has link to its next element in the sequence.

In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data field, and the next field. The data field is used to store actual value of the node and next field is used to store the address of next node in the sequence.

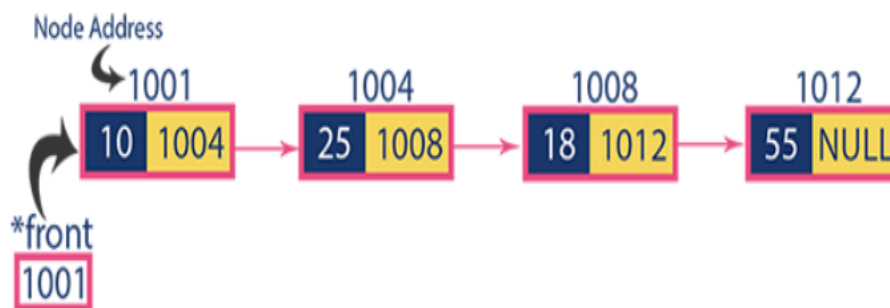
The graphical representation of a node in a single linked list is as follows...



✦ Important Points to be Remembered

- 🔗 In a single linked list, the address of the first node is always stored in a reference node known as "front" (Some times it is also known as "head").
- 🔗 Always next part (reference part) of the last node must be NULL.

Example

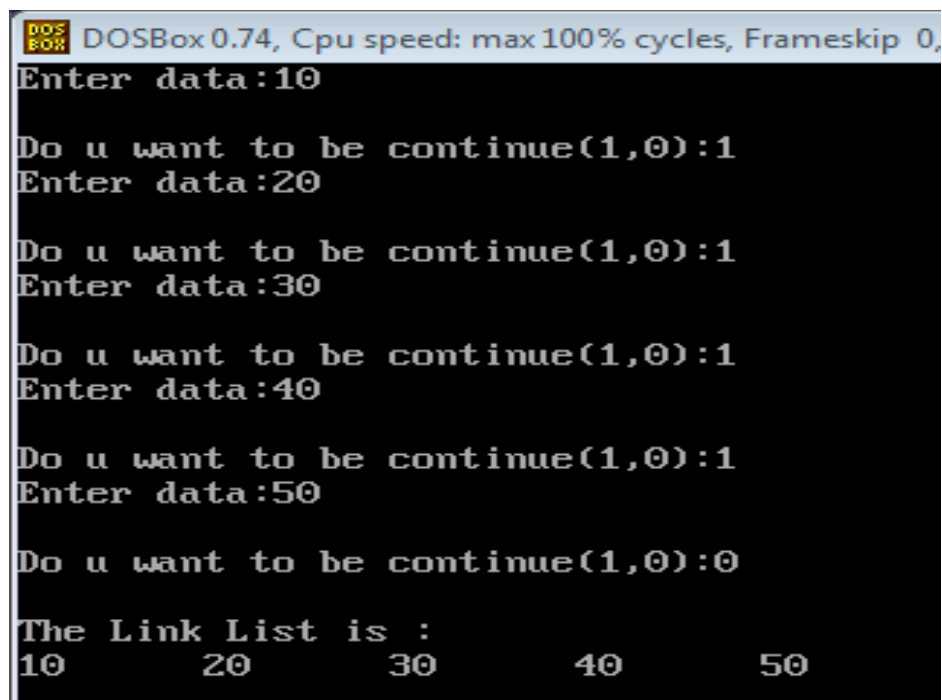


Exmple :

```
#include<stdio.h>
struct node
{
    int data;
    struct node *next;
};
int main()
{
    struct node *head,*newnode,*temp;
    int choice;
    clrscr();
    head=0;
    do
    {
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter data:");
        scanf("%d",&newnode->data);
        newnode->next=0;
        if(head==0)
        {
            head=temp=newnode;
        }
        else
        {
            temp->next=newnode;
            temp=newnode;
        }
        printf("\nDo u want to be continue(1,0):");
        scanf("%d",&choice);
    }while(choice);
    temp=head;
    printf("\nThe Link List is :\n");
    while(temp!=0)
    {
        printf("%d\t",temp->data);
```

```
        temp=temp->next;
    }
    getch();
    return 0;
}
```

Output :



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0,
Enter data:10

Do u want to be continue(1,0):1
Enter data:20

Do u want to be continue(1,0):1
Enter data:30

Do u want to be continue(1,0):1
Enter data:40

Do u want to be continue(1,0):1
Enter data:50

Do u want to be continue(1,0):0

The Link List is :
10      20      30      40      50
```

Operations on Single Linked List

The following operations are performed on a Single Linked List

- **Insertion**
- **Display**
- **Deletion**

Before we implement actual operations, first we need to set up an empty list. First, perform the following steps before implementing actual operations.

- Step 1 - Include all the **header files** which are used in the program.
- Step 2 - Declare all the **user defined functions**.
- Step 3 - Define a **Node** structure with two members **data** and **next**
- Step 4 - Define a Node pointer '**head**' and set it to **NULL**.
- Step 5 - Implement the main method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

1. Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

❖ Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

- Step 1 - Create a **newNode** with given value.
- Step 2 - Check whether list is **Empty** (**head == NULL**)
- Step 3 - If it is **Empty** then, set **newNode→next = NULL** and **head = newNode**.
- Step 4 - If it is **Not Empty** then, set **newNode→next = head** and **head = newNode**.

❖ Inserting At End of the list

We can use the following steps to insert a new node at end of the single linked list...

- Step 1 - Create a **newNode** with given value and **newNode → next** as **NULL**.
- Step 2 - Check whether list is **Empty** (**head == NULL**).
- Step 3 - If it is **Empty** then, set **head = newNode**.

- Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- Step 5 - Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
- Step 6 - Set **temp → next = newNode**.

❖ Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the single linked list...

- Step 1 - Create a **newNode** with given value.
- Step 2 - Check whether list is **Empty** (**head == NULL**)
- Step 3 - If it is **Empty** then, set **newNode → next = NULL** and **head = newNode**.
- Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- Step 5 - Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the **newNode** (until **temp1 → data** is equal to **location**, here location is the node value after which we want to insert the **newNode**).
- Step 6 - Every time check whether **temp** is reached to last node or not. If it is reached to last node then display '**Given node is not found in the list!!! Insertion not possible!!!**' and terminate the function. Otherwise move the **temp** to next node.
- Step 7 - Finally, Set '**newNode → next = temp → next**' and '**temp → next = newNode**'

2. Displaying a Single Linked List

We can use the following steps to display the elements of a single linked list...

- Step 1 - Check whether list is **Empty** (**head == NULL**)
- Step 2 - If it is **Empty** then, display '**List is Empty!!!**' and terminate the function.
- Step 3 - If it is **Not Empty** then, define a Node pointer '**temp**' and initialize with **head**.

- Step 4 - Keep displaying **temp** → **data** with an arrow (--->) until **temp** reaches to the last node
- Step 5 - Finally display **temp** → **data** with arrow pointing to **NULL** (**temp** → **data** ---> **NULL**).

3. Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

❖ Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the single linked list...

- Step 1 - Check whether list is **Empty** (**head == NULL**)
- Step 2 - If it is **Empty** then, display '**List is Empty!!! Deletion is not possible**' and terminate the function.
- Step 3 - If it is **Not Empty** then, define a Node pointer '**temp**' and initialize with **head**.
- Step 4 - Check whether list is having only one node (**temp** → **next == NULL**)
- Step 5 - If it is **TRUE** then set **head = NULL** and delete **temp** (Setting **Empty** list conditions)
- Step 6 - If it is **FALSE** then set **head = temp** → **next**, and delete **temp**.

❖ Deleting from End of the list

We can use the following steps to delete a node from end of the single linked list...

- Step 1 - Check whether list is **Empty** (**head == NULL**)
- Step 2 - If it is **Empty** then, display '**List is Empty!!! Deletion is not possible**' and terminate the function.

- Step 3 - If it is **Not Empty** then, define two Node pointers '**temp1**' and '**temp2**' and initialize '**temp1**' with **head**.
- Step 4 - Check whether list has only one Node (**temp1 → next == NULL**)
- Step 5 - If it is **TRUE**. Then, set **head = NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)
- Step 6 - If it is **FALSE**. Then, set '**temp2 = temp1** ' and move **temp1** to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next == NULL**)
- Step 7 - Finally, Set **temp2 → next = NULL** and delete **temp1**.

❖ Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the single linked list...

- Step 1 - Check whether list is **Empty** (**head == NULL**)
- Step 2 - If it is **Empty** then, display '**List is Empty!!! Deletion is not possible**' and terminate the function.
- Step 3 - If it is **Not Empty** then, define two Node pointers '**temp1**' and '**temp2**' and initialize '**temp1**' with **head**.
- Step 4 - Keep moving the **temp1** until it reaches to the exact node to be deleted or to the last node. And every time set '**temp2 = temp1**' before moving the '**temp1**' to its next node.
- Step 5 - If it is reached to the last node then display '**Given node not found in the list! Deletion not possible!!!**'. And terminate the function.
- Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
- Step 7 - If list has only one node and that is the node to be deleted, then set **head = NULL** and delete **temp1** (**free(temp1)**).
- Step 8 - If list contains multiple nodes, then check whether **temp1** is the first node in the list (**temp1 == head**).
- Step 9 - If **temp1** is the first node then move the **head** to the next node (**head = head → next**) and delete **temp1**.
- Step 10 - If **temp1** is not first node then check whether it is last node in the list (**temp1 → next == NULL**).
- Step 11 - If **temp1** is last node then set **temp2 → next = NULL** and delete **temp1** (**free(temp1)**).

- Step 12 - If **temp1** is not first node and not last node then set **temp2** → **next = temp1** → **next** and delete **temp1** (**free(temp1)**).

Example

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

```
void insertAtBeginning(int);
void insertAtEnd(int);
void insertBetween(int,int,int);
void display();
void removeBeginning();
void removeEnd();
void removeSpecific(int);
```

```
struct Node
{
    int data;
    struct Node *next;
}*head = NULL;
```

```
void main()
{
    int choice,value,choice1,loc1,loc2;
    clrscr();
    while(1){
        mainMenu: printf("\n\n***** MENU *****\n1. Insert\n2. Display\n3.
Delete\n4. Exit\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:    printf("Enter the value to be insert: ");
                      scanf("%d",&value);
                      while(1){
```

```

        printf("Where you want to insert: \n1. At Beginning\n2. At
End\n3. Between\nEnter your choice: ");
        scanf("%d",&choice1);
        switch(choice1)
        {
            case 1:      insertAtBeginning(value);
                        break;
            case 2:      insertAtEnd(value);
                        break;
            case 3:      printf("Enter the two values where you wanto insert:
");
                        scanf("%d%d",&loc1,&loc2);
                        insertBetween(value,loc1,loc2);
                        break;
            default:     printf("\nWrong Input!! Try again!!!\n\n");
                        goto mainMenu;
        }
        goto subMenuEnd;
    }
    subMenuEnd:
    break;
    case 2:      display();
    break;
    case 3:      printf("How do you want to Delete: \n1. From Beginning\n2.
From End\n3. Spesific\nEnter your choice: ");
        scanf("%d",&choice1);
        switch(choice1)
        {
            case 1:      removeBeginning();
                        break;
            case 2:      removeEnd();
                        break;
            case 3:      printf("Enter the value which you wanto delete: ");
                        scanf("%d",&loc2);
                        removeSpecific(loc2);
                        break;
            default:     printf("\nWrong Input!! Try again!!!\n\n");

```

```

                                goto mainMenu;
                                }
                                break;
    case 4:      exit(0);
    default: printf("\nWrong input!!! Try again!!\n\n");
}
}
}

```

```

void insertAtBeginning(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if(head == NULL)
    {
        newNode->next = NULL;
        head = newNode;
    }
    else
    {
        newNode->next = head;
        head = newNode;
    }
    printf("\nOne node inserted!!!\n");
}

```

```

void insertAtEnd(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if(head == NULL)
        head = newNode;
    else
    {
        struct Node *temp = head;

```

```

        while(temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    printf("\nOne node inserted!!!\n");
}

void insertBetween(int value, int loc1, int loc2)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if(head == NULL)
    {
        newNode->next = NULL;
        head = newNode;
    }
    else
    {
        struct Node *temp = head;
        while(temp->data != loc1 && temp->data != loc2)
            temp = temp->next;
        newNode->next = temp->next;
        temp->next = newNode;
    }
    printf("\nOne node inserted!!!\n");
}

```

```

void removeBeginning()
{
    if(head == NULL)
        printf("\n\nList is Empty!!!");
    else
    {
        struct Node *temp = head;
        if(head->next == NULL)
        {
            head = NULL;

```

```

        free(temp);
    }
    else
    {
        head = temp->next;
        free(temp);
        printf("\nOne node deleted!!!\n\n");
    }
}
}
void removeEnd()
{
    if(head == NULL)
    {
        printf("\nList is Empty!!!\n");
    }
    else
    {
        struct Node *temp1 = head,*temp2;
        if(head->next == NULL)
            head = NULL;
        else
        {
            while(temp1->next != NULL)
            {
                temp2 = temp1;
                temp1 = temp1->next;
            }
            temp2->next = NULL;
        }
        free(temp1);
        printf("\nOne node deleted!!!\n\n");
    }
}
void removeSpecific(int delValue)
{
    struct Node *temp1 = head, *temp2;

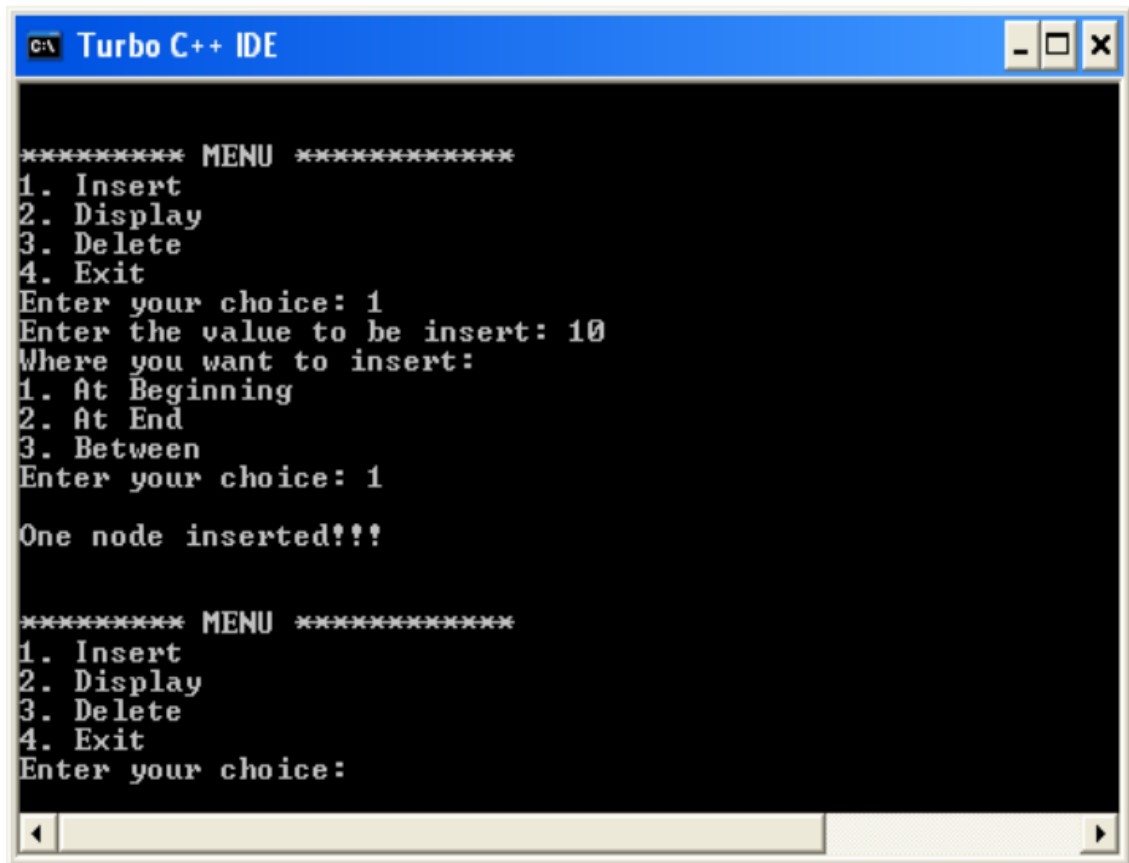
```

```

while(temp1->data != delValue)
{
    if(temp1 -> next == NULL){
        printf("\nGiven node not found in the list!!!");
        goto functionEnd;
    }
    temp2 = temp1;
    temp1 = temp1 -> next;
}
temp2 -> next = temp1 -> next;
free(temp1);
printf("\nOne node deleted!!!\n\n");
functionEnd:
}
void display()
{
    if(head == NULL)
    {
        printf("\nList is Empty\n");
    }
    else
    {
        struct Node *temp = head;
        printf("\n\nList elements are - \n");
        while(temp->next != NULL)
        {
            printf("%d --->",temp->data);
            temp = temp->next;
        }
        printf("%d --->NULL",temp->data);
    }
}

```

Output

A screenshot of the Turbo C++ IDE window. The title bar is blue and says "C:\ Turbo C++ IDE". The main area is black with white text. The text shows a menu-driven program. The first menu has options 1. Insert, 2. Display, 3. Delete, and 4. Exit. The user has entered choice 1. Then, the user enters the value 10 to be inserted. Next, a second menu asks where to insert: 1. At Beginning, 2. At End, 3. Between. The user has entered choice 1. The output then says "One node inserted!!!". The second menu is shown again at the bottom of the screen.

```
C:\ Turbo C++ IDE

***** MENU *****
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 1
Enter the value to be insert: 10
Where you want to insert:
1. At Beginning
2. At End
3. Between
Enter your choice: 1

One node inserted!!!

***** MENU *****
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice:
```