LINKED LIST:

A linked list is a **linear** data structure where each element is a separate object, known as a **node** . Each node contains some **data** and points to the **next node** in the structure, forming a **sequence**. The nodes may be at different memory locations, unlike arrays where all the elements are stored continuously.

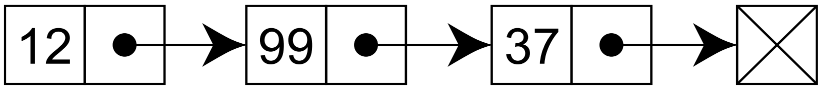
Types of Linked Lists

A linked list is designed depending on its use. The 3 most common types of a linked list are:

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List

This is the most common type of linked list, where each node has **one pointer** to the next node in the sequence. This means that the list can only be traversed from the beginning to the end in **one direction**. To access the last element, it is always required to traverse the whole list to the end.



The last node always points to **NULL** in a singly-linked list. This specifies that the list has ended with no more nodes to traverse to. Every time a loop traverses through the array, it checks for this NULL condition to know if the end of the linked list is there.

```c
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>


typedef struct Node

{

	int data;

	struct Node *next;

} node;


node *head = NULL;


node *allocate(node *newNode, int value)

{

	newNode = (node *)malloc(sizeof(node));

	newNode->data = value;

	newNode->next = NULL;

	return newNode;

}


void insrting_choices()

{

	printf("\nTYPE 01 ---> Insert at beginning,\n");

	printf("TYPE 02 ---> Insert at end,\n");

	printf("TYPE 03 ---> Insert at a position,\n");

	printf("Choice = ");

}

void deleteing_choices()

{

	printf("TYPE 01 ---> Delete beginning,\n");

	printf("TYPE 02 ---> Delete end,\n");

	printf("TYPE 03 ---> Delete from a position,\n");

	printf("Choice = ");

}

int get_data(int data)

{

	scanf("%d", &data);

	return data;
```

```c
}

void sizeOfList()
{
    node *temp = head;
    if (head == NULL)
        printf("List is contains 0 node.\n");
    else
    {
        int count = 0;
        while (temp->next != NULL)
        {
            count++;
            temp = temp->next;
        }
        printf("The list contains %d nodes.", count);
    }
}

void listInReverseOrder()
{
    node *prevNode, *curNode;

    if (head != NULL)
    {
        prevNode = head;
        curNode = head->next;
        head = head->next;

        prevNode->next = NULL; // convert the first node as last

        while (head != NULL)
        {
            head = head->next;
            curNode->next = prevNode;

            prevNode = curNode;
            curNode = head;
        }
        head = prevNode; // convert the last node as head
    }
}

void display()
{
    printf("\n_______________________________________________________\n");
    node *temp = head;
    if (head == NULL)
        printf("List is already empty!\n");
    else
    {
        while (temp->next != NULL)
        {
            printf("%d->", temp->data);
            temp = temp->next;
        }
        printf("%d", temp->data);
    }
    printf("\t\t\t|||| \t");
    sizeOfList();
    printf("\n-------------------------------------------------------\n");
}

void insertAtBeginning(int value)
{
    node *newNode;
    newNode = allocate(newNode, value);
    if (head == NULL)
        head = newNode;
    else
    {
        newNode->next = head;
        head = newNode;
    }
}
```

```c
void insertAtEnd(int value)
{
    node *newNode;
    newNode = allocate(newNode, value);
    if (head == NULL)
        head = newNode;
    else
    {
        node *temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
        // newNode->next = NULL;
    }
}

void insertAtPosition(int value, int pos)
{
    node *newNode;
    newNode = allocate(newNode, value);
    if (head == NULL)
        head = newNode;
    if (pos == 0 || pos == 1)
        insertAtBeginning(value);
    else
    {
        node *temp = head;
        for (int i = 1; i < pos - 1; i++)
            temp = temp->next;
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

void deleteAtBeginning()
{
    if (head == NULL)
        printf("List is already empty!\n");
    else
    {
        node *temp = head;
        if (temp->next == NULL)
        {
            head = NULL;
            free(temp);
        }
        else
        {
            head = temp->next;
            free(temp);
        }
    }
}

void deleteAtEnd()
{
    if (head == NULL)
        printf("List is already empty!\n");
    else
    {
        node *temp = head, *temp2;
        if (temp->next == NULL)
        {
            head = NULL;
            free(temp);
        }
        else
        {
            while (temp->next != NULL)
            {
                temp2 = temp;
                temp = temp->next;
            }
            temp2->next = NULL;
            free(temp);
        }
```

```c
    }
}

void deleteAtPosition(int pos)
{
    int i, flag = 0;
    if (head == NULL)
        printf("List is already empty!\n");
    if (pos == 0 || pos == 1)
        deleteAtBeginning();
    else
    {
        node *temp = head, *temp2;
        for (i = 0; i < pos - 1; i++)
        {
            temp2 = temp;
            temp = temp->next;
        }
        temp2->next = temp->next;
        free(temp);
    }
}

int main()
{
    int value, pos, i, n_node, choice, choice2;
    char c;
    do
    {
        printf("\nTYPE 01 ---> Inserting\n");
        printf("TYPE 02 ---> Deleting\n");
        printf("TYPE 03 ---> Display reverse order\n");
        printf("Your Choice is: ");
        choice2 = get_data(choice2);
        switch (choice2)
        {
        case 1:
            i = 0;
            printf("\n\nEnter number of nodes: ");
            n_node = get_data(n_node);
            insrting_choices();
            choice = get_data(choice);
            printf("\nEnter the values in one line:\n");
            while (i < n_node)
            {
                switch (choice)
                {
                case 1:
                    value = get_data(value);
                    insertAtBeginning(value);
                    break;
                case 2:
                    value = get_data(value);
                    insertAtEnd(value);
                    break;
                case 3:
                    if (n_node > 1)
                    {
                        printf("number of nodes must be equal one in this case.\n");
                        return main();
                    }
                    value = get_data(value);
                    pos = get_data(pos);
                    insertAtPosition(value, pos);
                    break;
                default:
                    printf("invalid choice!\n");
                    break;
                }
                i++;
            }
            break;
        case 2:
            deleteing_choices();
            choice = get_data(choice);
            switch (choice)
```

```c
            {
            case 1:
                deleteAtBeginning();
                break;
            case 2:
                deleteAtEnd();
                break;
            case 3:
                pos = get_data(pos);
                deleteAtPosition(pos);
                break;
            }
            break;
        case 3:
            printf("List in reverse order: ");
            listInReverseOrder();
            display();
            break;
        default:
            printf("Invalid choice.\n");
            break;
        }
        display();

        printf("\nif you want to continues reclick on (y/Y), else any letter: ");
        c = getch();
        fflush(stdin);
    } while (c == 'y' || c == 'Y');

    return 0;
}
```

/*

*/