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.

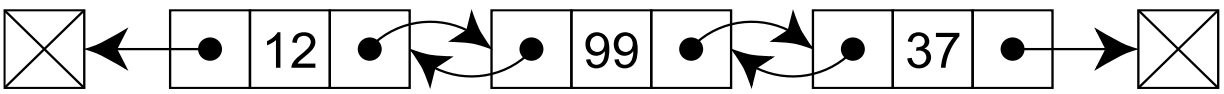
Types of Linked Lists

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

Doubly Linked List

A doubly linked list has 2 pointers, one pointing to the next node and one to the previous node. This allows for moving in any direction while traversing the list, which may be useful in certain situations.



The implementation and details are here: Link to Doubly linked list

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

typedef struct Node
{
     int data;
     struct Node *prev;
     struct Node *next;
} node;
node *head = NULL;

int get_data(int data)
{
     scanf("%d", &data);
     return data;
}

node *createNode(node *newNode, int value)
{
     newNode = (node *)malloc(sizeof(node));
     newNode->data = value;
     newNode->next = NULL;
     newNode->prev = NULL;
     return newNode;
}

void inserting_choices()
{
     printf("\nTYPE 01 ---> Insert at beginning\n");
     printf("TYPE 02 ---> Insert at the end\n");
     printf("TYPE 03 ---> Insert at a position\n");
     printf("-------<Choose from the list above>------- Choice = ");
}

void deleting_choices()
{
     printf("\nTYPE 01 ---> Delete from beginning\n");
```

```
     printf("TYPE 02 ---> Delete from the end\n");
     printf("TYPE 03 ---> Delete from a position\n");
     printf("-------<Choose from the list above>------- Choice = ");
}

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

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");
```

```
     sizeOfList();
     printf("\n-----------------------------------------\n");
}

void printRverse(node *head)
{
     if (head == NULL)
          return;
     printRverse(head->next);
     printf("%d->", head->data);
}

void display_reverse()
{
     printf("\n_____\n");
     printf("Given list in reverse order.\n");
     printRverse(head);
     printf("\n-----------------------------------------\n");
}

void insertAtBeginnig(int value)
{
     node *newNode;
     newNode = createNode(newNode, value);
     if (head == NULL)
          head = newNode;
     else
     {
          newNode->next = head;
          head->prev = newNode;
          head = newNode;
     }
}

void insertAtEnd(int value)
{
     node *newNode;
     newNode = createNode(newNode, value);
```

```
     if (head == NULL)
          head = newNode;
     else
     {
          node *temp = head;
          while (temp->next != NULL)
               temp = temp->next;
          newNode->prev = temp->next;
          temp->next = newNode;
     }
}

void insertAtPosition(int value, int pos)
{
     int flag = 1, i;
     node *newNode, *temp = head;
     newNode = createNode(newNode, value);
     if (head == NULL)
          head = newNode;
     else
     {
          if (pos == 0)
               insertAtBeginnig(value);
          else
          {
               for (i = 1; i < pos - 1; i++)
               {
                    temp = temp->next;
                    if (temp->next == NULL)
                    {
                         flag = 0;
                         break;
                    }
               }
               if (flag)
               {
                    newNode->next = temp->next;
                    newNode->prev = temp;
```

```
                    temp->next->prev = newNode;
                    temp->next = newNode;
               }
          }
     }
}

void deleteFromBeginnig()
{
     if (head == NULL)
          printf("List is already empty!\n");
     else
     {
          node *temp = head;
          if ((temp->prev == NULL) && (temp->next == NULL))
          {
               head = NULL;
               free(temp);
          }
          else
          {
               head = temp->next;
               head->prev = NULL;
               free(temp);
          }
          printf("Deleted successful.\n");
     }
}

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

```
    }
    else
    {
        while (temp->next != NULL)
        {
            temp2 = temp;
            temp = temp->next;
        }
        temp2->next = NULL;
        free(temp);
    }
}

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

int main()
{
    int choice, choice2, n_node, i, value, pos;
    char c;
    do
    {
```

```
printf("\nTYPE 01 ---> Inserting\n");
printf("TYPE 02 ---> Deleting\n");
printf("TYPE 03 ---> Display\n");
printf("TYPE 04 ---> Display reverse order.\n");
printf("Choice = ");
choice = get_data(choice);
switch (choice)
{
case 1:
     inserting_choices();
     choice2 = get_data(choice2);
     printf("\nEnter how many nodes you want to create: ");
     n_node = get_data(n_node);
     i = 0;
     printf("\nEnter data:\n");
     while (i < n_node)
     {
          switch (choice2)
          {
          case 1:
               value = get_data(value);
               insertAtBeginnig(value);
               break;
          case 2:
               value = get_data(value);
               insertAtEnd(value);
               break;
          case 3:
               if (n_node > 1)
                    printf("You can't insert more than one node.\n");
               value = get_data(value);
               pos = get_data(pos);
               insertAtPosition(value, pos);
               break;
          default:
               printf("Inavlid choice!! Try again.\n");
               break;
          }
```

```
            i++;
        }
        break;
    case 2:
        deleting_choices();
        choice2 = get_data(choice2);
        switch (choice2)
        {
        case 1:
            deleteFromBeginnig();
            break;
        case 2:
            deleteFromEnd();
            break;

        case 3:
            pos = get_data(pos);
            deleteFromPosition(pos);
            break;
        default:
            printf("Inavlid choice!! Try again.\n");
            break;
        }
        break;
    case 3:
        display();
        break;
    case 4:
        display_reverse();
        break;
    default:
        printf("Invalid choice!\n");
        break;
    }

    display();

    printf("To continue hit  (y/Y) else any letter: ");
```

```
        c = getch();
        fflush(stdin);
    } while (c == 'y' || c == 'Y');

    return 0;
}
```

```
/*
```

Edit by: MOHAMMED ELIDRISSI LAOUKILI
#idriCoding | #idriLogic | #idri

Instagram: @__elidrissii

لا اله الا الله

*/