18ES611 Embedded System Programming

Sarath tv

Linked List

Common alternative to arrays in the implementation of data structures.

Each item in a linked list contains a data element of some type and a pointer to the next item in the list.

It is <u>easy to insert and delete elements </u>in a linked list, which are not natural operations on arrays, since arrays have a fixed size.

An item in a linked list consists of a struct containing the data element and a pointer to another linked list.

Linked lists and arrays are similar since they both store collections of data.

Arrays are convenient to declare and the provide the handy [ ] syntax to access any element by its index number. The entire array is allocated as one block of memory each element in the array gets its own space in the array. Any element can be accessed directly using the [ ] syntax

Disadvantages of arrays

The size of the array is fixed

Most convenient thing for programmers to do is to allocate arrays which seem "large enough“

Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.

Linked lists have their own strengths and weaknesses, but they happen to be strong where arrays are weak.

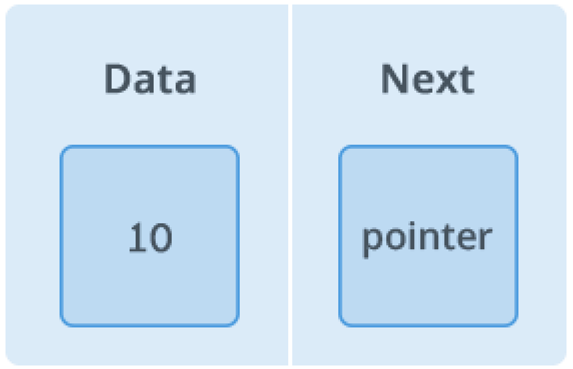
An array allocates memory for all its elements lumped together as one block of memory.

In contrast, a linked list allocates space for each element separately in its own block of memory called a "linked list element" or "node". The list gets is overall structure by using pointers to connect all its nodes together like the links in a chain.

Each node contains two fields: a "data" field to store whatever element type the list holds for its client, and a "next" field which is a pointer used to link one node to the next node. Each node is allocated in the heap with a call to malloc(), so the node memory continues to exist until it is explicitly deallocated with a call to free().

The front of the list is a pointer to the first node

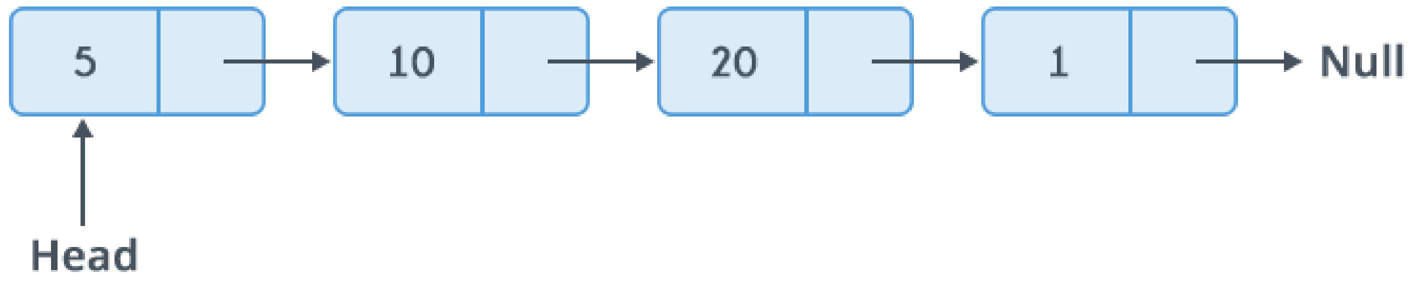
A **linked list** is a way to store a collection of elements. Like an array these can be character or integers. Each element in a linked list is stored in the form of a **node**



A node is a collection of two sub-elements or parts. A **data** part that stores the element and a **next** part that stores the link to the next node.

Members in a linked list

A linked list is formed when many such nodes are linked together to form a chain. Each node points to the next node present in the order. The first node is always used as a reference to traverse the list and is called **HEAD**. The last node points to **NULL**.



Creating a linked list

Nodes

Each node will have two parts- data and pointer to next node.

Implementation of node –structure.

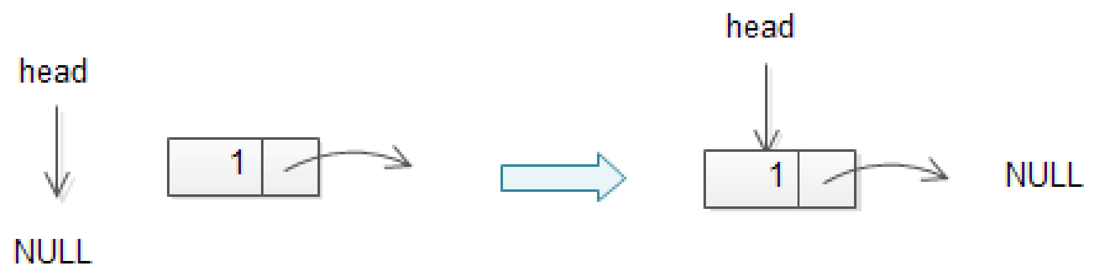
```
struct node {
   int data;
   struct node *next;
};
```

Operation on linked List

Important points to remember:

- Head points to the first node of the linked list
- Next pointer of last node is NULL, so if next of current node is NULL, we have reached end of linked list.
- All new node creation will be done from heap memory.
- HEAD =NULL implies empty list.

Insert a data at the beginning of linked list



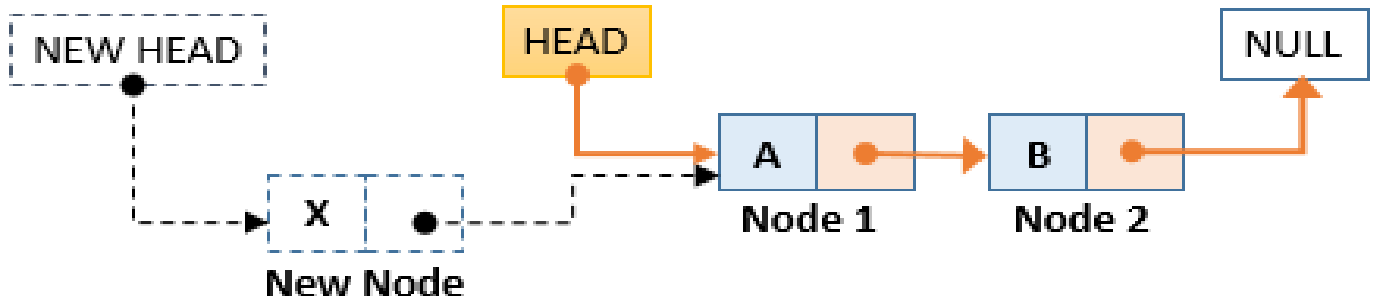
Argument for the function - number

Create a temporary pointer to node structure.

Dynamically allocate memory for the node .

Two cases arises

- List is empty
  - *Assign the address of the temp node to start(head)*
  - *Assign the number to the data field of the node.*
  - *Next pointer address as NULL.*
- List is not empty
  - *Assign the number to the data field of the node.*
  - *Modify start address so as to make currently created node as start and existing start address as next node).*

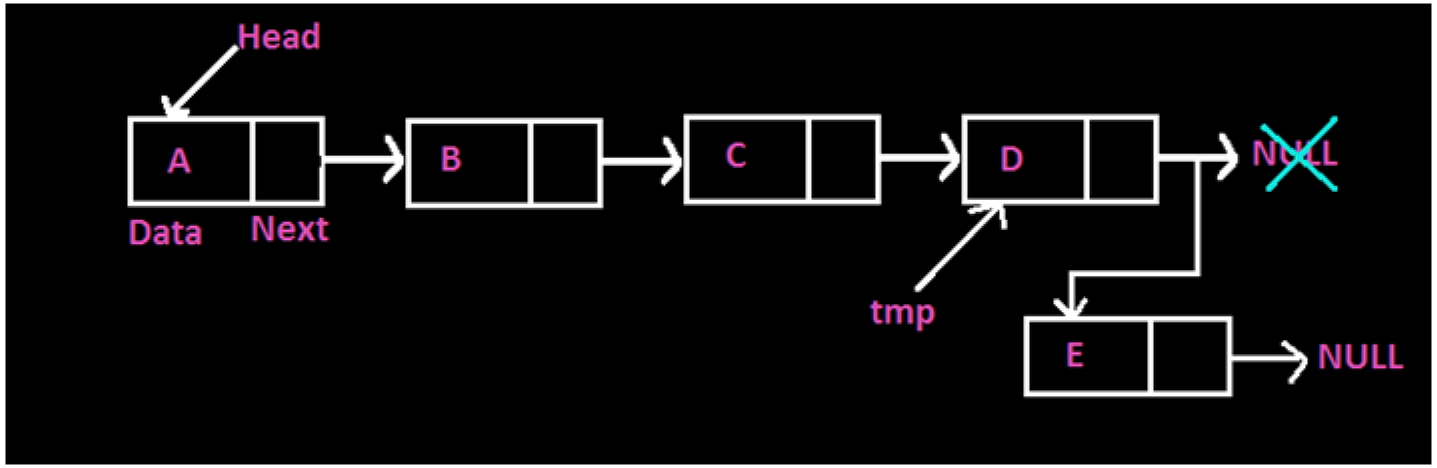


```
struct node *start = NULL;// global
void insert_at_begin(int x) {
   struct node *t;

   t = (struct node*)malloc(sizeof(struct node));

   if (start == NULL) {
      start = t;
      start->data = x;
      start->next = NULL;
      return;
   }

   t->data = x;
   t->next = start;
   start = t;
}
```

Empty case

Non empty case

Insert a data at the End of linked list



```
void insert_at_end(int x) {
  struct node *t, *temp; // two pointers to node structure

  t = (struct node*)malloc(sizeof(struct node));

  if (start == NULL) {
    start = t;
    start->data = x;
    start->next = NULL;
    return;
  }

  temp = start;

  while (temp->next != NULL) //
           {          temp = temp->next; //  move to next node
           }

  temp->next = t;
  t->data    = x;
  t->next    = NULL;
}
```

Empty case

Non empty case- Reach the last node

t holds address of dynamically allotted memory for new node.

Add to middle

Argument number, position

Allocate memory and store data for new node

Traverse to node just before the required position of new node

Change next pointers to include new node in between

Create your own function.

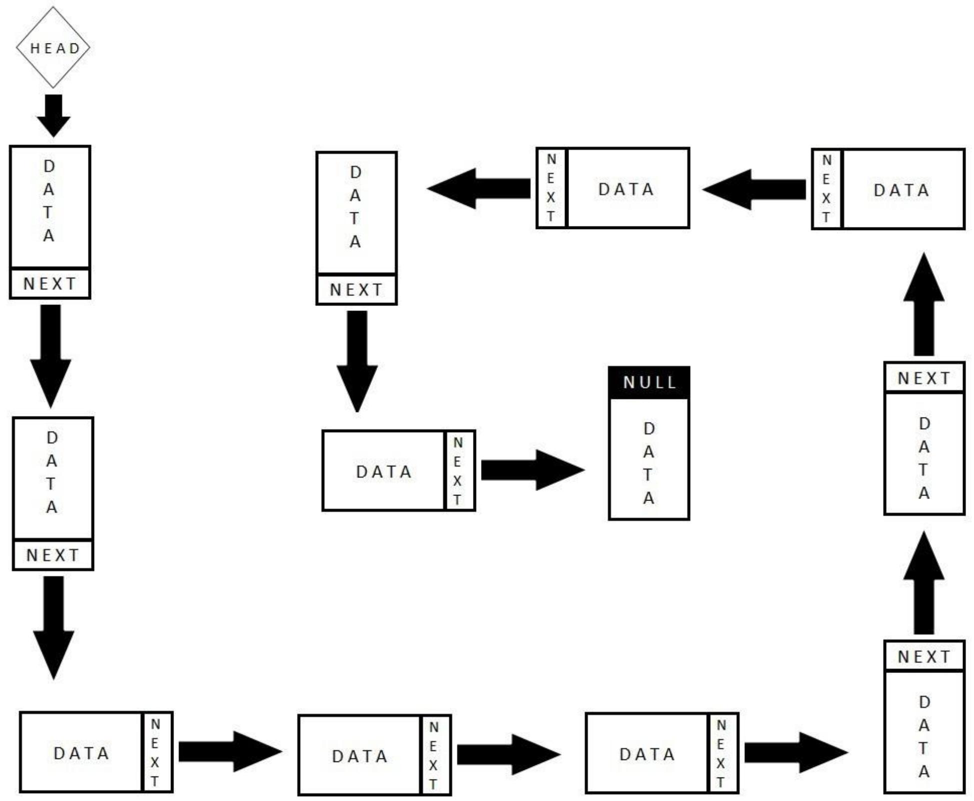
How to traverse a linked list

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

Logic- create a temporary node and assign it with address of start.

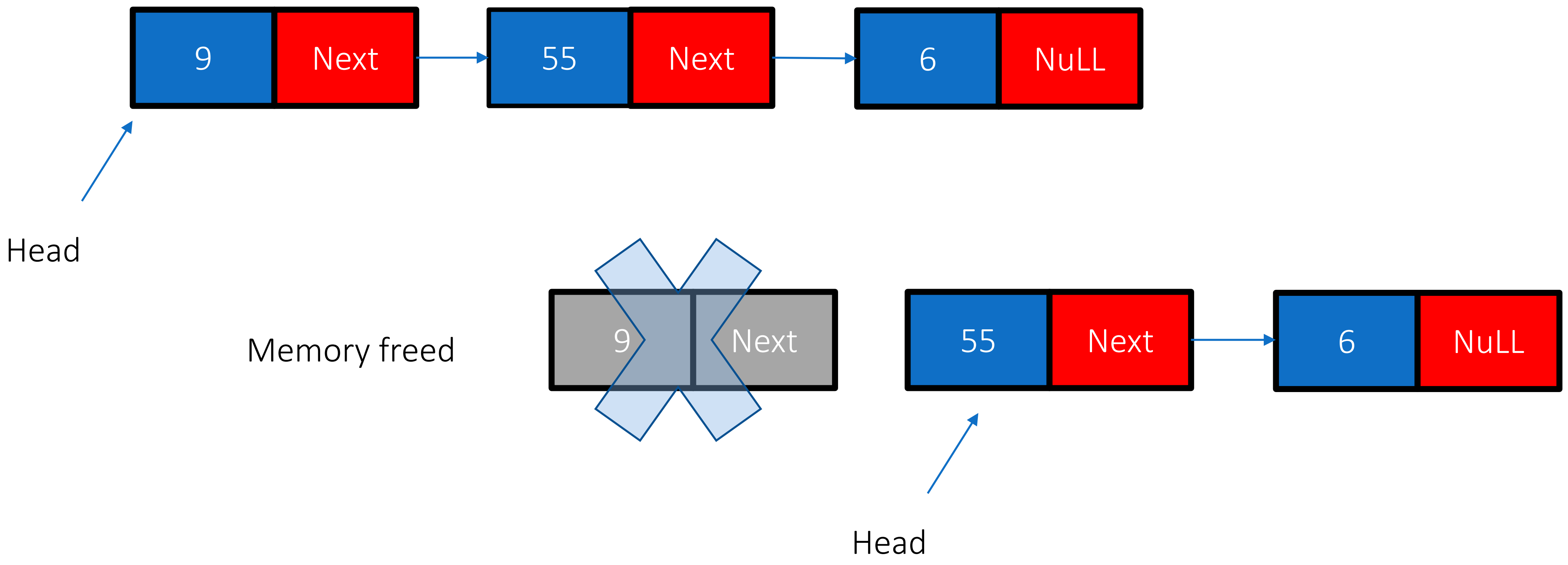
Then in a loop check if the link(pointer to next node) is null or not. If not assign the address of the next link to temp node.

When temp is NULL, we know that we have reached the end of linked list so we get out of the while loop.



```
void traverse() {
   struct node *t; // temp node

   t = start;


   while (t->next != NULL) {
      printf("%d\n", t->data);
      t = t->next;
   }
   printf("%d\n", t->data);
}
```

Deleting a node from beginning

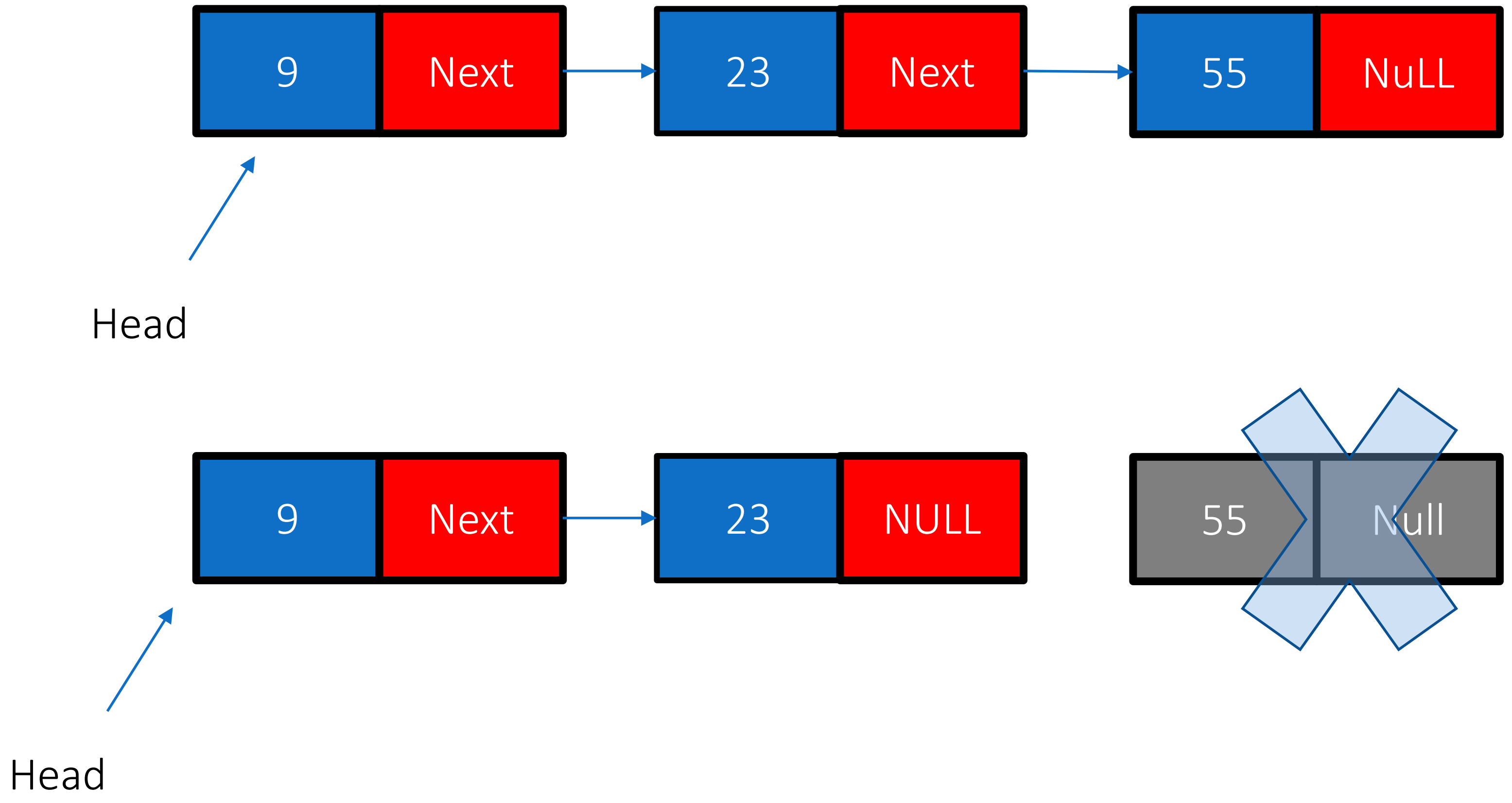


```
void delete_from_begin() {
  struct node *t;
  int n;

  if (start == NULL) {
    printf("Linked list is already empty.\n");
    return;
  }

  n = start->data;
  t = start->next;
  free(start);
step
  start = t;

  printf("%d deleted from beginning successfully.\n", n);
}
```

Empty

very important

Deleting a node from end



- No node
- Only one node
- Multiple nodes

empty

```
void delete_from_end() {
   struct node *t, *u;
   int n;

   if (start == NULL) {
      printf("Linked list is already empty.\n");
      return;
   }
```

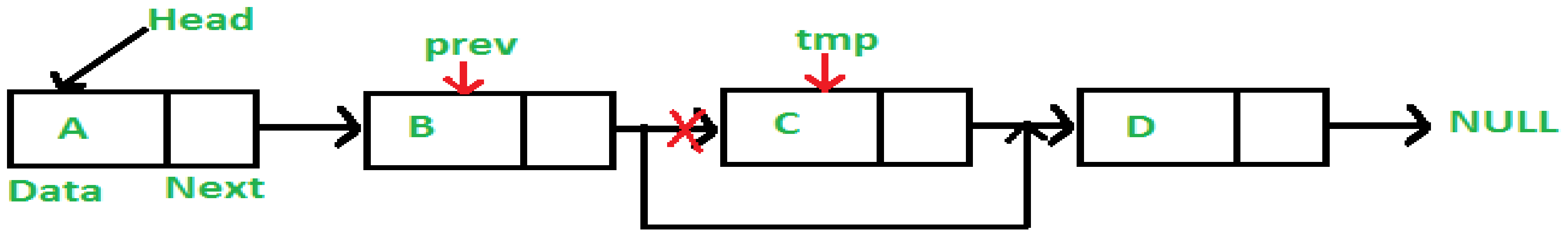
Only one node

```
   if (start->next == NULL) {
      n = start->data;
      free(start);
      start = NULL;
      printf("%d deleted from end successfully.\n", n);
      return;
   }

   t = start;
```

Many node

```
   while (t->next != NULL) {
      u = t;
      t = t->next;
   }

   n = t->data;
   u->next = NULL;
   free(t);

   printf("%d deleted from end successfully.\n", n);
}
```

Delete from Middle or any particular location



Add a count variable to track total number of elements in linked list

C Program to Implement a Stack using Structure

Repeat the coding done for Stack but now use structure.

APIs -push, pop and display

```
struct stack
{
int stk[MAXSIZE];
int top;
};

struct stack s;

s.top = s.top + 1;
s.stk[s.top] = num;
```

C Program to Implement a Stack using Linked list

```
struct node
{
    int info;
    struct node *ptr;
}
struct node *top,*temp;
```

```
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    }
```

THANK YOU!!!!!