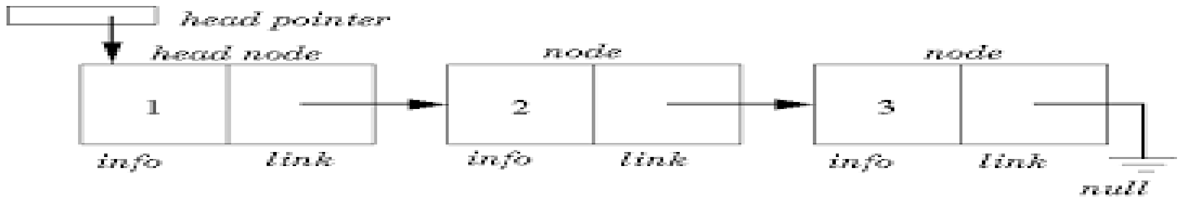
Linked Lists

Introduction

In computer science, a linked list is a linear data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a link to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.



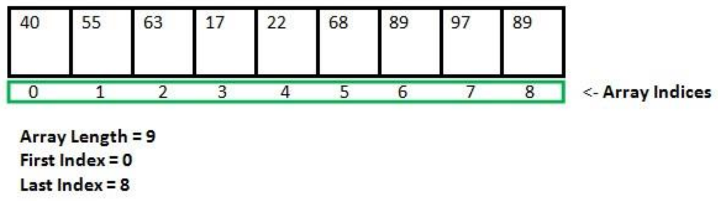
A Linked List

Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1) Fixed size.

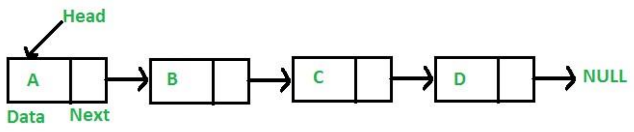
2)Inserting /deletion element in an array is expensive (time complexity) O(N)



So Linked list provides the following two advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

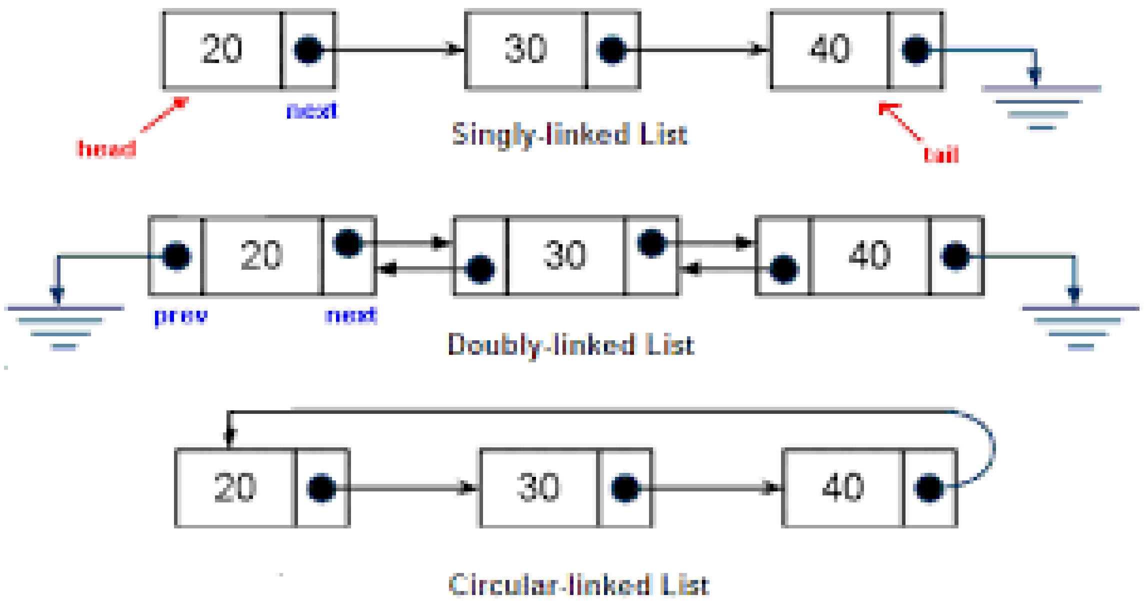


There are 3 different implementations of Linked List available, they are:

1. Singly Linked List.

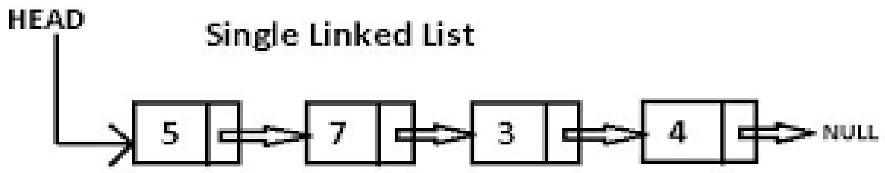
2. Doubly Linked List.

3. Circular Linked List.



Singly Linked List

Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in the sequence of nodes.



The operations we can perform on singly linked lists are:

1. Creating a list.
2. Insertion an element in a list.
3. Deletion an element from a list.
4. Searching a list.
5. Display a list.

Creating Linked list

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL. Each node in a list consists of at least two parts:

1) data

2) Pointer (Or Reference) to the next node

In C++, we can represent a node using structures. Below is an example of a linked list node with integer data.

```
// A linked list node
struct node
{
 int  data;
node * next;
 };*start;          // start points at the first node
start=NULL;      // initialized to NULL at beginning
```

```
node * create(int num)      //say num=1 pass from main
{
node* ptr;
ptr=new node;       // memory allocation dynamically
if (ptr==NULL)
'OVERFLOE'       //no memory available
exit(1);
else {
ptr ->data=num;
ptr->next=NULL;
return ptr;
}
```

To be called from main () as

```
void main()

{

node* ptr;

int data ;

cin>>data;

ptr =create (data);

}
```

Advantages of Linked Lists

1. They are a dynamic in nature which allocates the memory when required.
2. Insertion and deletion operations can be easily implemented.
3. Stacks and queues can be easily executed.
4. Linked List reduces the access time.

Disadvantages of Linked Lists

1. The memory is wasted as pointers require extra memory for storage.
2. No element can be accessed randomly; it has to access each node sequentially.
3. Reverse Traversing (display) is difficult in linked list.

Applications of Linked Lists

1. Linked lists are used to implement stacks, queues, graphs, etc.
2. Linked lists let you insert node at (the beginning, end , after the practical node) of the list.
3. Linked lists let you delete from (first node, last node, intermediate node) of the list.