# Self Referential Structures——

Self Referential structures are those [structures](https://www.geeksforgeeks.org/structures-c/) that have one or more pointers which point to the same type of structure, as their member.

In other words, structures pointing to the same type of structures are self-referential in nature.

struct node

{

    int data1;

    struct node\* next; //int \*a;

};

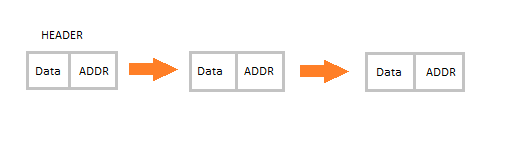
In the above example ‘next’ is a pointer to a structure of type ‘node’. Hence, the structure ‘node’ is a self-referential structure with ‘next’ as the referencing pointer.

A self-referential structure is used to create data structures like linked lists.

**Introduction to linked list**

Linked List is a very commonly used linear data structure which consists of group of **nodes** in a sequence.

Each node holds its own **data** and the **address of the next node** hence forming a chain like structure.



struct node

{

    int DATA;

    struct node\*ADDR ;

};

**Advantages of Linked Lists**

* They are dynamic in nature which allocates the memory when required.
* Insertion and deletion operations can be easily implemented. // in array these operation are difficult
* Linked List reduces the access time.

**Disadvantages of Linked Lists**

* The memory is wasted as pointers require extra memory for storage.
* No element can be accessed randomly; it has to access each node sequentially.
* Reverse Traversing is difficult in linked list. //

Traversing means to see all elements

**Applications of Linked Lists**

* Linked lists let you insert elements at the beginning and end of the list.
* In Linked Lists we don't need to know the size in advance.

## Types of Linked Lists

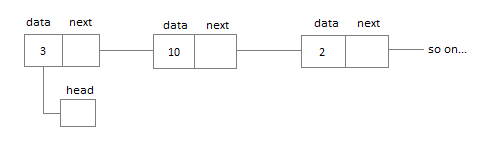
There are 3 different implementations of Linked List available, they are:

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 **insertion**, **deletion** and **traversal**.



struct node {

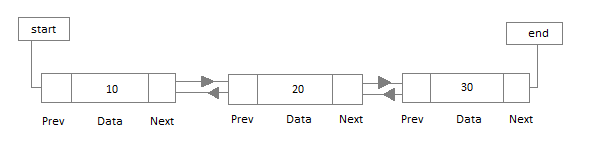
    int data;

    struct node\* next;

};

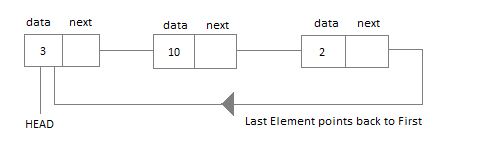
### **Doubly Linked List**

In a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.



### **Circular Linked List**

In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.



Imp------self referential—def+code

Linked list ----def+code+types+adv(read)