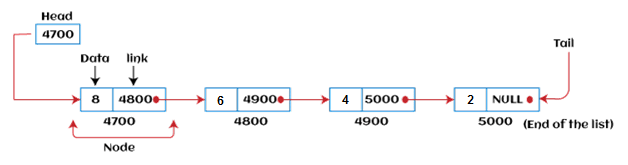
UNITE 3 - Linked list

Linked list, Algorithms and implementation of singly linked list, Doubly linked list, Circular linked list, Operations on linked list, Applications – polynomial representation, addition of two polynomials.

Linked list is a linear data structure that includes a series of connected nodes. Linked list can be defined as the nodes that are randomly stored in the memory. A node in the linked list contains two parts, i.e., first is the data part and second is the address part. The last node of the list contains a pointer to the null. After array, linked list is the second most used data structure. In a linked list, every link contains a connection to another link.

Representation of a Linked list

Linked list can be represented as the connection of nodes in which each node points to the next node of the list. The representation of the linked list is shown below -



Till now, we have been using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages that must be known to decide the data structure that will be used throughout the program.

Why use linked list over array?

Linked list is a data structure that overcomes the limitations of arrays. Let's first see some of the limitations of arrays -

* The size of the array must be known in advance before using it in the program.
* Increasing the size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
* All the elements in the array need to be contiguously stored in the memory. Inserting an element in the array needs shifting of all its predecessors.
* Linked list is useful because -
* It allocates the memory dynamically. All the nodes of the linked list are non-contiguously stored in the memory and linked together with the help of pointers.
* In linked list, size is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

How to declare a linked list?

It is simple to declare an array, as it is of single type, while the declaration of linked list is a bit more typical than array. Linked list contains two parts, and both are of different types, i.e., one is the simple variable, while another is the pointer variable. We can declare the linked list by using the user-defined data type **structure.**

The declaration of linked list is given as follows -

struct node

{

int data;

struct node \*next;

}

In the above declaration, we have defined a structure named as **node** that contains two variables, one is **data** that is of integer type, and another one is **next** that is a pointer which contains the address of next node.

Advantages of Linked list

The advantages of using the Linked list are given as follows -

* **Dynamic data structure -** The size of the linked list may vary according to the requirements. Linked list does not have a fixed size.
* **Insertion and deletion -** Unlike arrays, insertion, and deletion in linked list is easier. Array elements are stored in the consecutive location, whereas the elements in the linked list are stored at a random location. To insert or delete an element in an array, we have to shift the elements for creating the space. Whereas, in linked list, instead of shifting, we just have to update the address of the pointer of the node.
* **Memory efficient -** The size of a linked list can grow or shrink according to the requirements, so memory consumption in linked list is efficient.
* **Implementation -** We can implement both stacks and queues using linked list.

Disadvantages of Linked list

The limitations of using the Linked list are given as follows -

* **Memory usage -** In linked list, node occupies more memory than array. Each node of the linked list occupies two types of variables, i.e., one is a simple variable, and another one is the pointer variable.
* **Traversal -** Traversal is not easy in the linked list. If we have to access an element in the linked list, we cannot access it randomly, while in case of array we can randomly access it by index. For example, if we want to access the 3rd node, then we need to traverse all the nodes before it. So, the time required to access a particular node is large.
* **Reverse traversing -** Backtracking or reverse traversing is difficult in a linked list. In a doubly-linked list, it is easier but requires more memory to store the back pointer.

Types of Linked list

Linked list is classified into the following types -

* **Singly-linked list -** Singly linked list can be defined as the collection of an ordered set of elements. A node in the singly linked list consists of two parts: data part and link part. Data part of the node stores actual information that is to be represented by the node, while the link part of the node stores the address of its immediate successor.
* **Doubly linked list -** Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly-linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer), and pointer to the previous node (previous pointer).
* **Circular singly linked list -** In a circular singly linked list, the last node of the list contains a pointer to the first node of the list. We can have circular singly linked list as well as circular doubly linked list.
* **Circular doubly linked list -** Circular doubly linked list is a more complex type of data structure in which a node contains pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the nodes. The last node of the list contains the address of the first node of the list. The first node of the list also contains the address of the last node in its previous pointer.

Operations performed on Linked list

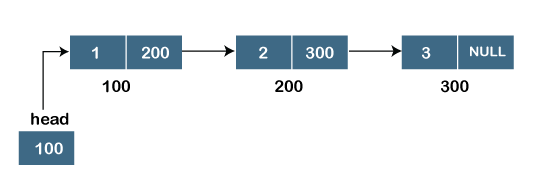
The basic operations that are supported by a list are mentioned as follows -

* **Insertion -** This operation is performed to add an element into the list.
* **Deletion -** It is performed to delete an operation from the list.
* **Display -** It is performed to display the elements of the list.
* **Search -** It is performed to search an element from the list using the given key.

Singly Linked list

It is the commonly used linked list in programs. If we are talking about the linked list, it means it is a singly linked list. The singly linked list is a data structure that contains two parts, i.e., one is the data part, and the other one is the address part, which contains the address of the next or the successor node. The address part in a node is also known as a **pointer**.

Suppose we have three nodes, and the addresses of these three nodes are 100, 200 and 300 respectively. The representation of three nodes as a linked list is shown in the below figure:



We can observe in the above figure that there are three different nodes having address 100, 200 and 300 respectively. The first node contains the address of the next node, i.e., 200, the second node contains the address of the last node, i.e., 300, and the third node contains the NULL value in its address part as it does not point to any node. The pointer that holds the address of the initial node is known as a ***head pointer***.

The linked list, which is shown in the above diagram, is known as a singly linked list as it contains only a single link. In this list, only forward traversal is possible; we cannot traverse in the backward direction as it has only one link in the list.

**Representation of the node in a singly linked list**

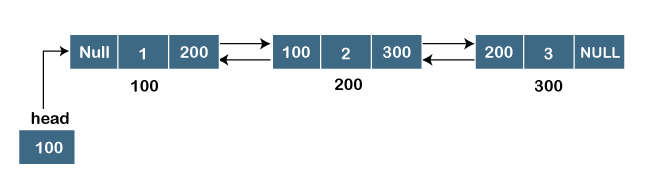
1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. }

In the above representation, we have defined a user-defined structure named a **node** containing two members, the first one is data of integer type, and the other one is the pointer (next) of the node type.

Doubly linked list

As the name suggests, the doubly linked list contains two pointers. We can define the doubly linked list as a linear data structure with three parts: the data part and the other two address part. In other words, a doubly linked list is a list that has three parts in a single node, includes one data part, a pointer to its previous node, and a pointer to the next node.

Suppose we have three nodes, and the address of these nodes are 100, 200 and 300, respectively. The representation of these nodes in a doubly-linked list is shown below:

As we can observe in the above figure, the node in a doubly-linked list has two address parts; one part stores the ***address of the next*** while the other part of the node stores the ***previous node's address***. The initial node in the doubly linked list has the **NULL** value in the address part, which provides the address of the previous node.

**Representation of the node in a doubly linked list**

1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. struct node \*prev;
6. }

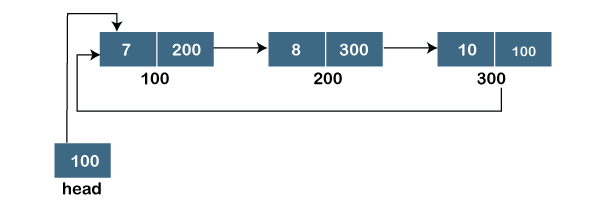
In the above representation, we have defined a user-defined structure named ***a node*** with three members, one is **data** of integer type, and the other two are the pointers, i.e., **next and prev** of the node type. The **next pointer** variable holds the address of the next node, and the **prev pointer** holds the address of the previous node. The type of both the pointers, i.e., **next and prev** is **struct node** as both the pointers are storing the address of the node of the ***struct node*** type.

Circular linked list

A circular linked list is a variation of a singly linked list. The only difference between the ***singly linked list*** and a ***circular linked*** list is that the last node does not point to any node in a singly linked list, so its link part contains a NULL value. On the other hand, the circular linked list is a list in which the last node connects to the first node, so the link part of the last node holds the first node's address. The circular linked list has no starting and ending node. We can traverse in any direction, i.e., either backward or forward. The diagrammatic representation of the circular linked list is shown below:

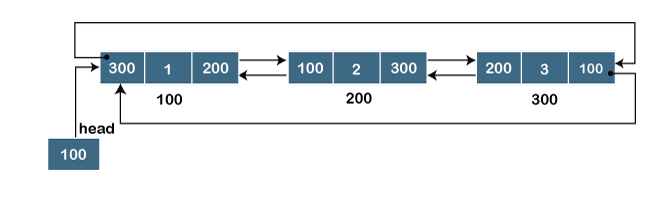
1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. }

A circular linked list is a sequence of elements in which each node has a link to the next node, and the last node is having a link to the first node. The representation of the circular linked list will be similar to the singly linked list, as shown below:



Doubly Circular linked list

The doubly circular linked list has the features of both the ***circular linked list*** and ***doubly linked list***.



The above figure shows the representation of the doubly circular linked list in which the last node is attached to the first node and thus creates a circle. It is a doubly linked list also because each node holds the address of the previous node also. The main difference between the doubly linked list and doubly circular linked list is that the doubly circular linked list does not contain the NULL value in the previous field of the node. As the doubly circular linked contains three parts, i.e., two address parts and one data part so its representation is similar to the doubly linked list.

1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. struct node \*prev;
6. }