List ADT

List is the collection of elements in sequential order. In memory we can store the list in two ways.

- Sequential Memory Location - Array
- Pointer or links to associate the elements sequential – Linked List.

THE LIST ADT

List is an ordered set of elements. The general form of the list is

A1, A2, A3, , AN

A1 - First element of the list AN - Last element of the list N - Size of the list

If the element at position i is Ai then its successor is Ai+1 and its predecessor is Ai-1.

Various operations performed on List

- create an empty list
- printList () – prints all elements in the list construct a (deep) copy of a list
- find(x) – returns the position of the first occurrence of x
- remove(x) – removes x from the list if present
- insert (x, position) – inserts x into the list at the specified position isEmpty () – returns true if the list has no elements
- makeEmpty () – removes all elements from the list findKth (int k) – returns the element in the specified position

LINKED LIST

Definition

A linked list is a collection of data elements called nodes in which the linear representation is given by links from one node to the next node.

Reason for Linked List

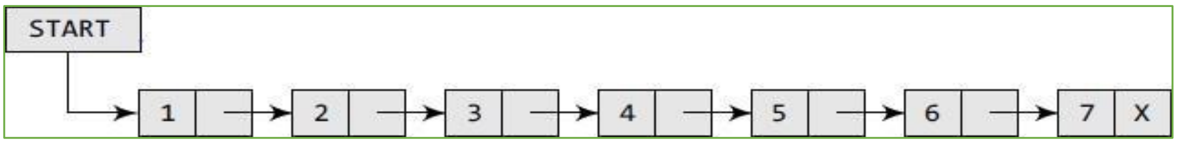
- Array is a linear collection of data elements in which the elements are stored in consecutive memory locations.
- While declaring arrays, we have to specify the size of the array, which will restrict the number of elements that the array can store. For example, if we declare an array as int marks [10], then the array can store a maximum of 10 data elements but not more than that.
- But what if we are not sure of the number of elements in advance? Moreover, to make efficient use of memory, the elements must be stored randomly at any location rather than in consecutive locations.
- So, there must be a data structure that removes the restrictions on the maximum number
- of elements and the storage condition to write efficient programs.

Advantages of using linked list

- A linked list does not store its elements in consecutive memory locations and the user can add any number of elements to it.
- However, unlike an array, a linked list does not allow random access of data. Elements in a linked list can be accessed only in a sequential manner.
- But like an array, insertions and deletions can be done at any point in the list in a constant time.

Basic Terminologies

- A linked list, in simple terms, is a linear collection of data elements. These data elements are called nodes.
- Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as a building block to implement data structures such as stacks, queues, and their variations.
- A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.
- Linked lists provide an efficient way of storing related data and perform basic operations such as insertion, deletion, and updation of information at the cost of extra space required for storing address of the next node.



- In above figure, we can see a linked list in which every node contains two parts, an integer and a pointer to the next node.
- The left part of the node which contains data may include a simple data type, an array, or a structure.
- The right part of the node contains a pointer to the next node (or address of the next node in sequence).
- The last node will have no next node connected to it, so it will store a special value called NULL. In above figure, the NULL pointer is represented by X.
- While programming, we usually define NULL as –1. Hence, a NULL pointer denotes the end of the list.
- Since in a linked list, every node contains a pointer to another node which is of the same type, it is also called a self-referential data type.

Use of START pointer variable

- Linked lists contain a pointer variable START that stores the address of the first node in the list. We can traverse the entire list using START which contains the address of the first node; the next part of the first node in turn stores the address of its succeeding node.
- Using this technique, the individual nodes of the list will form a chain of nodes. If
- START = NULL, then the linked list is empty and contains no nodes.

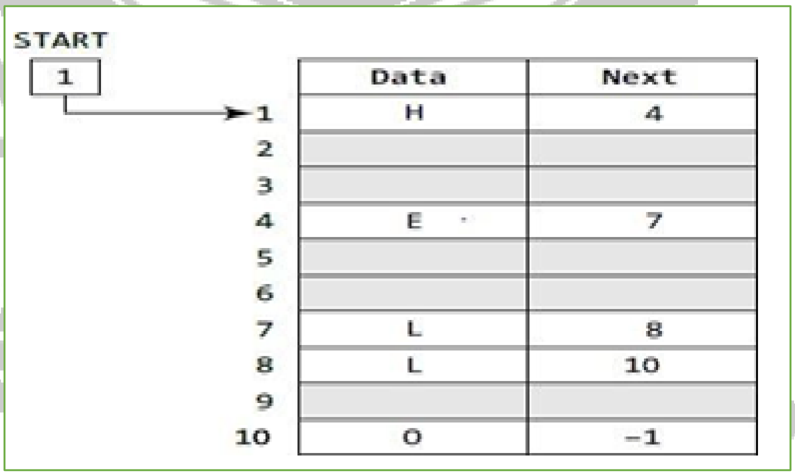
Sample Linked List code

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

Memory representation of Linked List

- Let us see how a linked list is maintained in the memory. In order to form a linked list, we need a structure called node which has two fields, DATA and NEXT.
- DATA will store the information part and NEXT will store the address of the next node in sequence.

- Consider the below figure, we can see that the variable START is used to store the address of the first node.
- Here, in this example, START = 1, so the first data is stored at address 1, which is H. The corresponding NEXT stores the address of the next node, which is 4. So, we will look at address 4 to fetch the next data item.
- The second data element obtained from address 4 is E. Again, we see the corresponding NEXT to go to the next node.
- From the entry in the NEXT, we get the next address, that is 7, and fetch L as the data. We repeat this procedure until we reach a position where the NEXT entry contains –1 or NULL, as this would denote the end of the linked list.
- When we traverse DATA and NEXT in this manner, we finally see that the linked list in the above example stores characters that when put together form the word HELLO.
- Note that in the below figure shows a chunk of memory locations which range from 1 to 10. The shaded portion contains data for other applications.
- Remember that the nodes of a linked list need not be in consecutive memory locations. In our example, the nodes for the linked list are stored at addresses 1, 4, 7, 8, and 10.

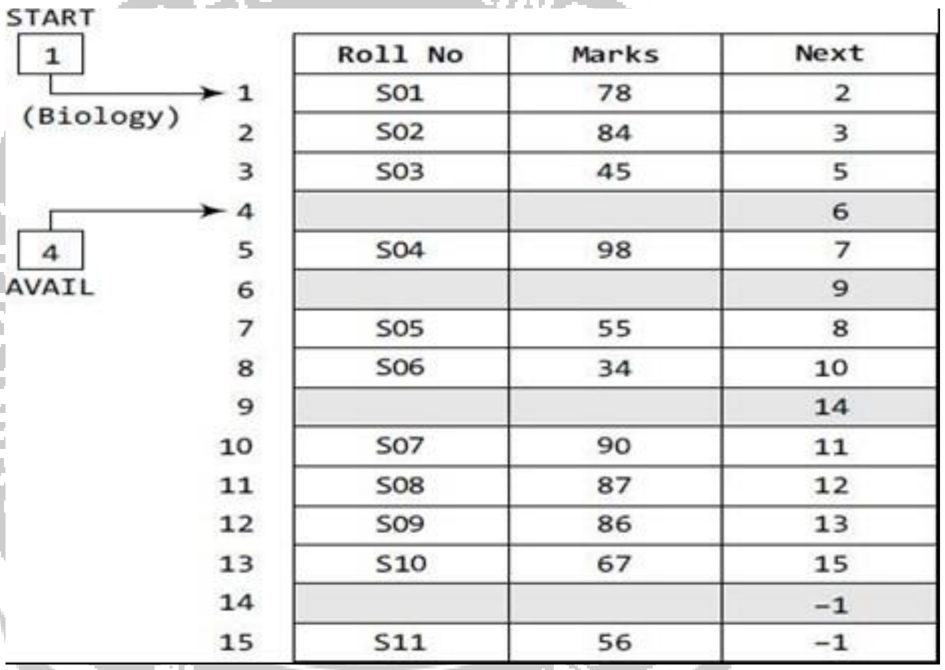


	Data	Next
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	–1

FREE POOL AND AVAIL POINTER VARIABLE

- The computer maintains a list of all free memory cells. This list of available space is called the free pool.

- We have seen that every linked list has a pointer variable START which stores the address of the first node of the list. Likewise, for the free pool (which is a linked list of all free memory cells), we have a pointer variable AVAIL which stores the address of the first free space.
- Let us revisit the memory representation of the linked list storing all the students' marks in Biology.
- Now, when a new student's record has to be added, the memory address pointed by AVAIL will be taken and used to store the desired information.
- After the insertion, the next available free space's address will be stored in AVAIL.



	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			6
5	S04	98	7
6			9
7	S05	55	8
8	S06	34	10
9			14
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			–1
15	S11	56	–1

- For example, in the above Figure when the first free memory space is utilized for inserting the new node, AVAIL will be set to contain address 6.
- This was all about inserting a new node in an already existing linked list. Now, we will discuss deleting a node or the entire linked list.
- When we delete a particular node from an existing linked list or delete the entire linked list, the space occupied by it must be given back to the free pool so that the memory can be reused by some other program that needs memory space.
- The operating system does this task of adding the freed memory to the free pool. The operating system will perform this operation whenever it finds the CPU idle or whenever the programs are falling short of memory space.

- The operating system scans through all the memory cells and marks those cells that are being used by some program.
- Then it collects all the cells which are not being used and adds their address to the free pool, so that these cells can be reused by other programs. This process is called garbage collection.