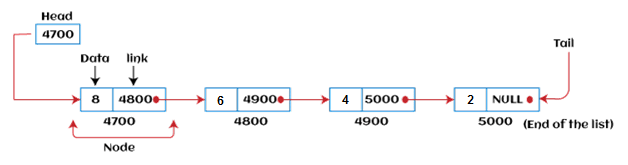
Linked list

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 -



## **Singly Linked List**

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist 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.

Consider an example where the marks obtained by the student in three subjects are stored in a linked list as shown in the figure.

DS Singly Linked List

# 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) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.



A doubly linked list containing three nodes having numbers from 1 to 3 in their data part, is shown in the following image.



# 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.

We traverse a circular singly linked list until we reach the same node where we started. The circular singly liked list has no beginning and no ending. There is no null value present in the next part of any of the nodes.

The following image shows a circular singly linked list.

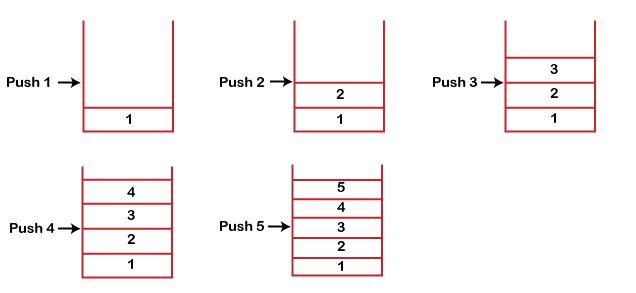


Circular linked list are mostly used in task maintenance in operating systems. There are many examples where circular linked list are being used in computer science including browser surfing where a record of pages visited in the past by the user, is maintained in the form of circular linked lists and can be accessed again on clicking the previous button

STACK

Stack works on the LIFO pattern. As we can observe in the below figure there are five memory blocks in the stack; therefore, the size of the stack is 5.

Suppose we want to store the elements in a stack and let's assume that stack is empty. We have taken the stack of size 5 as shown below in which we are pushing the elements one by one until the stack becomes full.



Since our stack is full as the size of the stack is 5. In the above cases, we can observe that it goes from the top to the bottom when we were entering the new element in the stack. The stack gets filled up from the bottom to the top.

QUEUES

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.



QUESTION 1-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

class LinearSearch {

public static int linearSearch(int array[], int x) {

int n = array.length;

// Going through array sequencially

for (int i = 0; i < n; i++) {

if (array[i] == x)

return i;

}

return -1;

}

public static void main(String args[]) {

int array[] = { 2, 4, 0, 1, 9 };

int x = 1;

int result = linearSearch(array, x);

if (result == -1)

System.out.print("Element not found");

else

System.out.print("Element found at index: " + result);

}

}

Question 2----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

class Main {

int binarySearch(int array[], int element, int low, int high) {

// Repeat until the pointers low and high meet each other

while (low <= high) {

// get index of mid element

int mid = low + (high - low) / 2;

// if element to be searched is the mid element

if (array[mid] == element)

return mid;

// if element is less than mid element

// search only the left side of mid

if (array[mid] < element)

low = mid + 1;

// if element is greater than mid element

// search only the right side of mid

else

high = mid - 1;

}

return -1;

}

public static void main(String args[]) {

// create an object of Main class

Main obj = new Main();

// create a sorted array

int[] array = { 3, 4, 5, 6, 7, 8, 9 };

int n = array.length;

// get input from user for element to be searched

Scanner input = new Scanner(System.in);

System.out.println("Enter element to be searched:");

// element to be searched

int element = input.nextInt();

input.close();

// call the binary search method

// pass arguments: array, element, index of first and last element

int result = obj.binarySearch(array, element, 0, n - 1);

if (result == -1)

System.out.println("Not found");

else

System.out.println("Element found at index " + result);

}

}