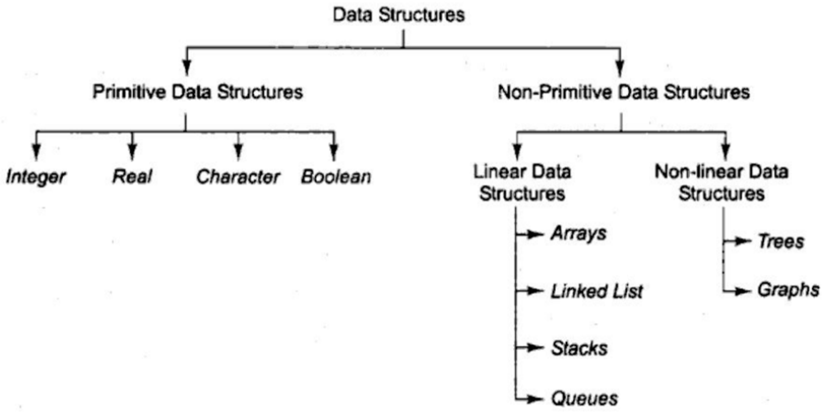
Lab 1

Introduction

Data structures

The primary objective of programming is to efficiently process the input to generate the desired output. We can achieve this objective in an efficient and neat style if the input data is organized in a way to help us meet our goal. Data structures is nothing but ways and means of organizing data so that it can be processed easily and efficiently. Data structure dictate the manner in which the data can be processed. In other words the choice of an algorithm depends upon the underlying data organization.

Classification of Data Structures:



Algorithm(set of logic statement / method)

An algorithm is a set of rules for carrying out calculation either by hand or on a machine. An algorithm is a finite step-by-step procedure to achieve a required result. An algorithm is a sequence of computational steps that transform the input into the output. An algorithm is a sequence of operations performed on data that have to be organized in data structures.

It is a detailed sequences of steps from beginning to end to carry out the whole process. logic or the basic plan of the action, behind of getting something done.it has rules to follow.

How to express an Algorithm?

Algorithms can be expressed in many kinds of notation, including:

- Natural language
- Pseudo Code
- Flowcharts
- Programming Language

Pseudo code (function of describing / outlining the Algorithm)

Pseudo code is like "intermediary" between algorithm and implemented program. Algorithm is the semantic of the process while pseudo code is the syntax to communication sake to solve any problem or do any task. No rules.
In short:
Algorithm is the "skeleton" of the plan/task and program is like "giving it its shape" and pseudo code is telling the program how the algorithm wants the plan/ task to turn out to be.

PSEUDO-CODE is the easiest for human brain to understand or to make.

Algorithm vs Pseudocode

An algorithm is simply a solution to a problem. An algorithm presents the solution to a problem as a well-defined set of steps or instructions. Pseudo-code is a general way of describing an algorithm. Pseudo-code does not use the syntax of a specific programming language, therefore cannot be executed on a computer. But it closely resembles the structure of a programming language and contains roughly the same level of detail

How to write a good pseudo code?

- Use indention for improved clarity
- Do not put “code” in pseudo code make your pseudo code language independent
- Don’t write pseudo code for yourself – write it in an unambiguous fashion so that anyone with a reasonable knowledge can understand and implement it
- Be consistent
- Prefer formulas over English language descriptions

LAB No: 1

Largest Element in Array

Objective

To find the Largest element in an Array.

Theory:

An Array is collection of cells of the same type. The cells are numbered with consecutive integers. Array cells are contiguous in computer memory. The memory can be thought of as an array.

Algorithm

A nonempty array DATA with N numerical values is given. This algorithm finds the location LOC and value MAX of the largest element of DATA. The variable K is used as a counter.

```
Step 1: [Initialize] set K: =1, LOC: =1 and MAX: =DATA[1]
Step 2: [Increment counter] set K: =K+1.
Step 3: [Test counter] if K>N, then
            Write LOC, MAX, and Exit.
            [End of If structure]
Step 4: [Compute and update] if MAX<DATA[K], then
            Set LOC: =K and MAX: =DATA[K].
            [End of If structure]
Step 5: [Repeat loop] Go to step 2                    [GOTO - To link one line with another.]
```

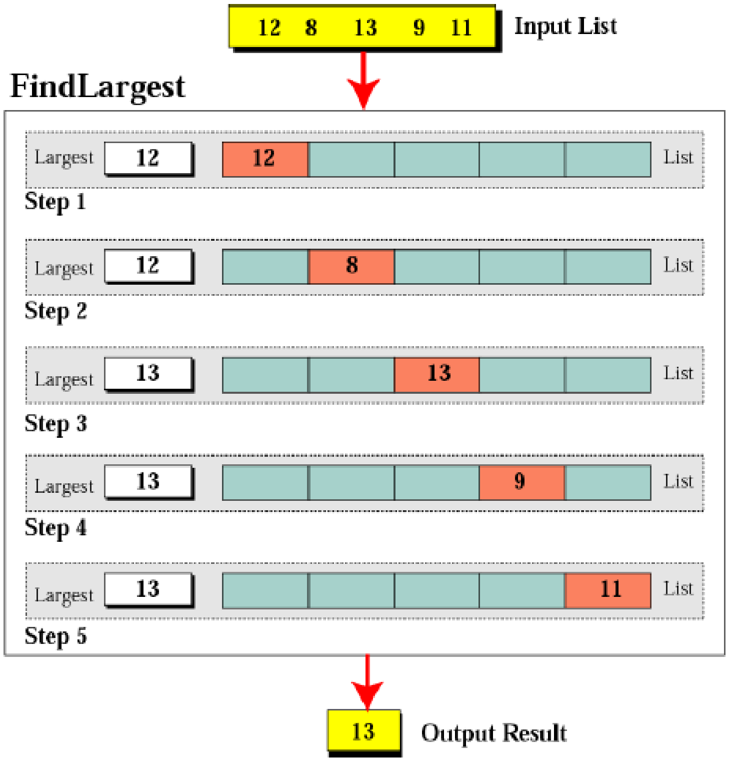
Algorithm (Rewritten using a repeat-while loop rather than Go to statement)

```
1. [Initialize] Set K: = 1 & LOC: = 1 and MAX: =DATA[1]
2. Repeat step 3 and 4 while K < = N:
3.      If MAX<DATA[K], then
             Set LOC: = K and MAX: =DATA[K].
        [End of If structure]
4.      Set K: = K + 1
   [End of Step 2 while loop]
5. Write: LOC, MAX.
6. Exit.
```

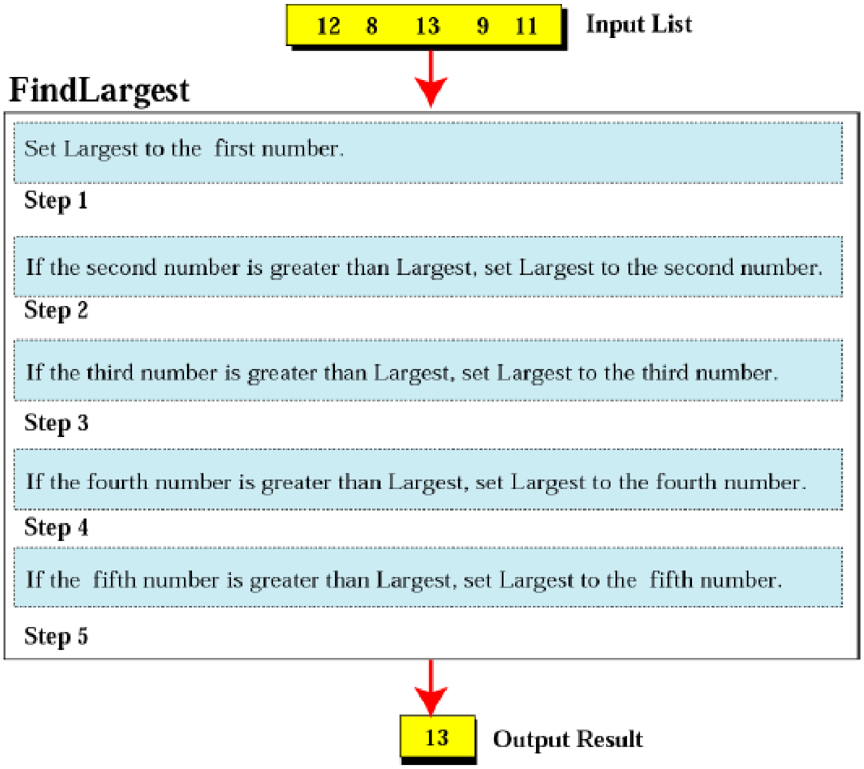
Lab Assignment

Find the largest element in array using the above two algorithms.

Finding the Largest Integer among five Integers



Defining actions in Find the Largest Algorithm



Set Largest to the first number.

Step 1

If the second number is greater than Largest, set Largest to the second number.

Step 2

If the third number is greater than Largest, set Largest to the third number.

Step 3

If the fourth number is greater than Largest, set Largest to the fourth number.

Step 4

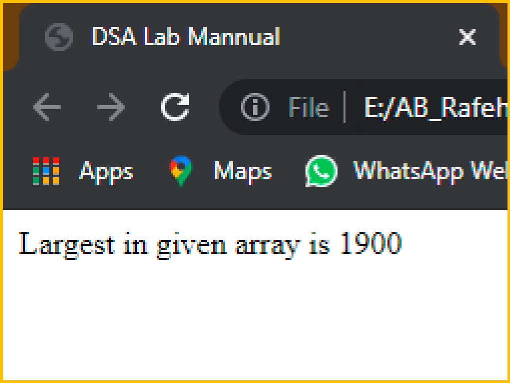
If the fifth number is greater than Largest, set Largest to the fifth number.

Step 5

LAB 1

```
function largest(arr) {
    let i;

    // Initialize maximum element
    let max = arr[0];

    for (i = 1; i < arr.length; i++) {
        if (arr[i] > max)
            max = arr[i];
    }

    return max;
}

let arr = [150, 1150, 1485, 1900, 1048];
document.write("Largest in given array is " + largest(arr));
```

Output:



```
Largest in given array is 1900
```

LAB No: 2(A)

Linear and Binary Search of an Array

Objective

To find the element in an Array using Linear Search

Theory

The linear search compares each element of the array with the ***search key*** until the search key is found. To determine that a value is not in the array, the program must compare the search key to every element in the array. It is also called "***Sequential Search***" because it traverses the data sequentially to locate the element.

Algorithm (Linear Search)

A Linear DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in array DATA or sets LOC=0.

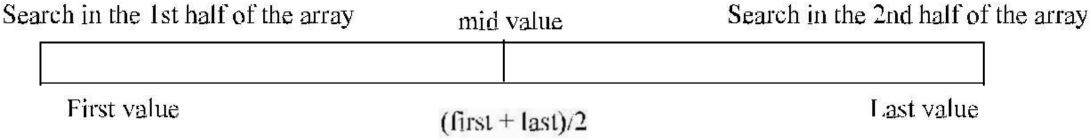
```
1. [Initialize] Set k:=1 and LOC:=0.
2. Repeat Step 3 and 4 while LOC=0 AND k<=N.
3.          If ITEM = DATA[k], then:
               Set LOC:=k.
4.          Else
               Set k:=k+1.   [Increments counter]
            [End of If Structure.]
    [End of Step 2 loop.]
5. [Successful?]
   If LOC=0 then:
            Write: ITEM is not in the array DATA.
   Else
            Write: LOC is the location of ITEM.
   [End of If structure]
6. Exit.
```

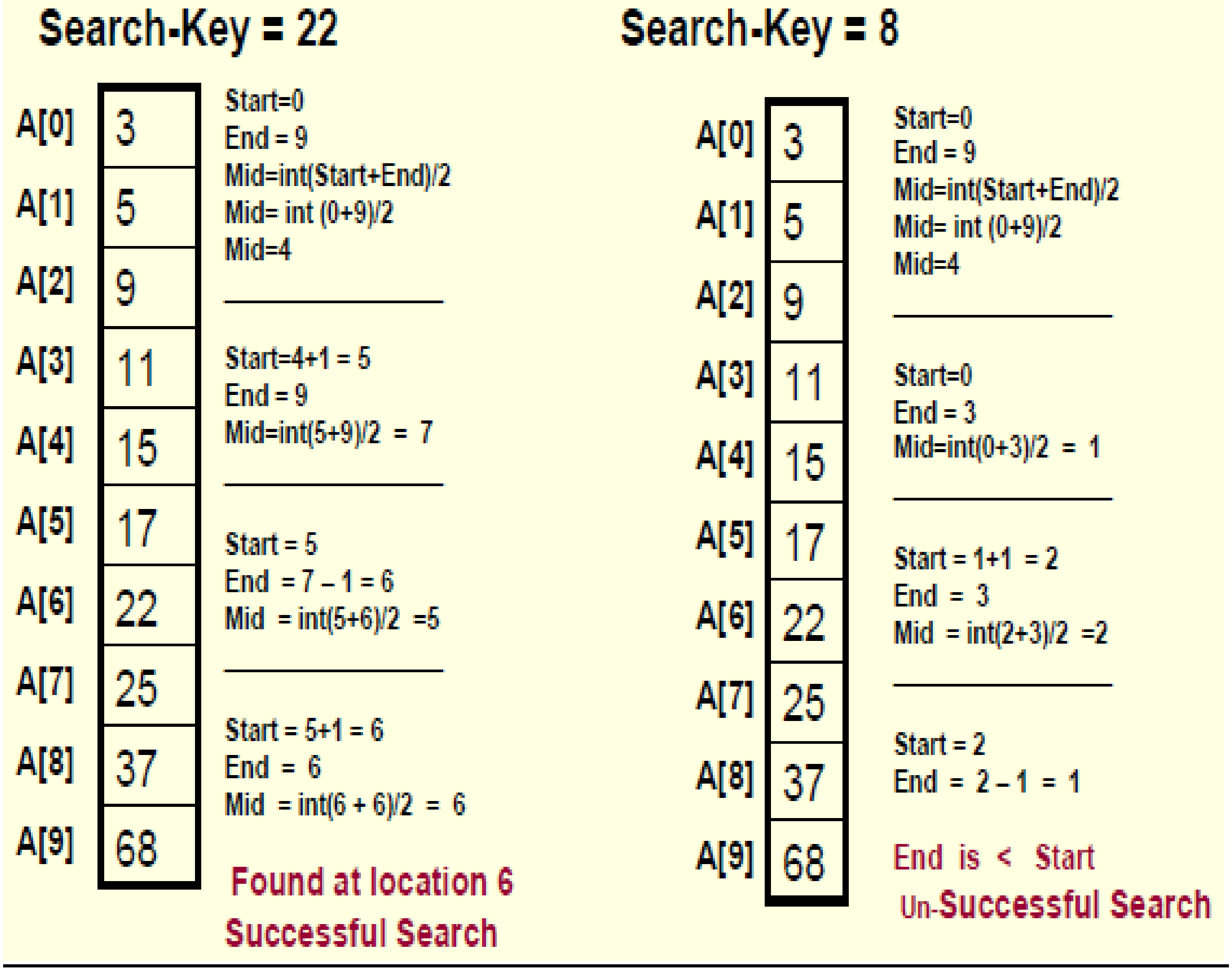
LAB No: 2(B)

Objective

To find the element in an Array using Binary Search

Theory: Search a *sorted array* by repeatedly dividing the search interval in half.
It is useful for the large sorted arrays. The binary search algorithm can only be used with ***sorted array*** and eliminates one half of the elements in the array being searched after each comparison. The algorithm locates the middle element of the array and compares it to the search key. If they are equal, the search key is found and array subscript of that element is returned. Otherwise the problem is reduced to searching one half of the array. If the search key is less than the middle element of array, the first half of the array is searched. If the search key is not the middle element of in the specified sub array, the algorithm is repeated on one quarter of the original array. The search continues until the sub array consist of one element that is equal to the search key (search successful). But if Search-key not found in the array then the value of END of new selected range will be less than the START of new selected range. This will be explained in the following example:


Search in the 1st half of the array
mid value
Search in the 2nd half of the array
First value
(first + last)/2
Last value



Search-Key = 22

A[0]	3
A[1]	5
A[2]	9
A[3]	11
A[4]	15
A[5]	17
A[6]	22
A[7]	25
A[8]	37
A[9]	68

Start=0
End = 9
Mid=int(Start+End)/2
Mid= int (0+9)/2
Mid=4

Start=4+1 = 5
End = 9
Mid=int(5+9)/2 = 7

Start = 5
End = 7 – 1 = 6
Mid = int(5+6)/2 =5

Start = 5+1 = 6
End = 6
Mid = int(6 + 6)/2 = 6

Found at location 6
Successful Search

Search-Key = 8

A[0]	3
A[1]	5
A[2]	9
A[3]	11
A[4]	15
A[5]	17
A[6]	22
A[7]	25
A[8]	37
A[9]	68

Start=0
End = 9
Mid=int(Start+End)/2
Mid= int (0+9)/2
Mid=4

Start=0
End = 3
Mid=int(0+3)/2 = 1

Start = 1+1 = 2
End = 3
Mid = int(2+3)/2 =2

Start = 2
End = 2 – 1 = 1

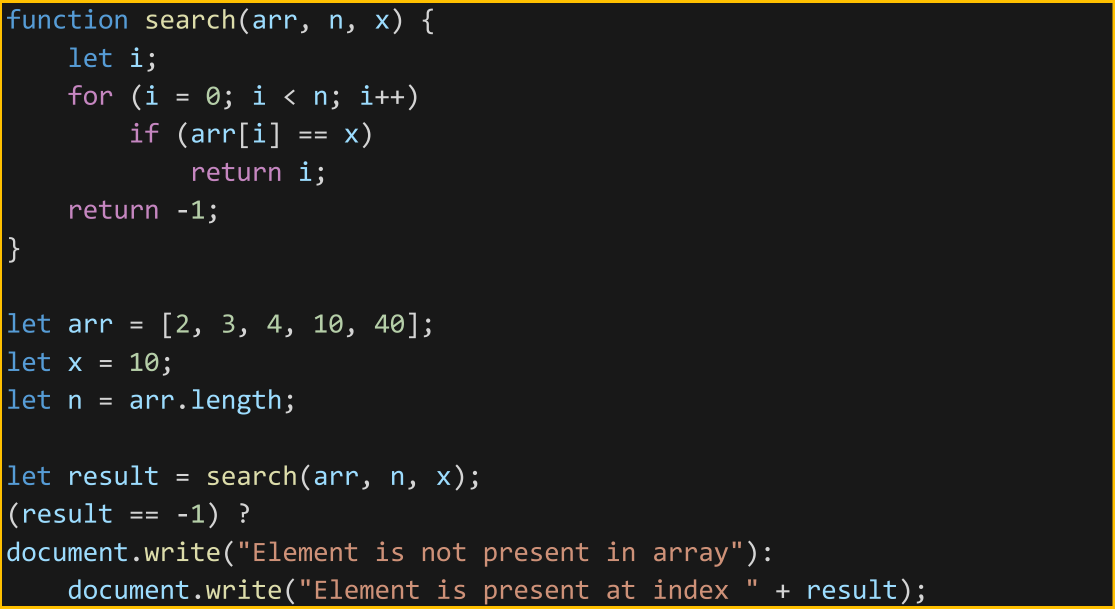
End is < Start
Un-Successful Search

Algorithm (Binary Search)

Here **A** is a sorted Linear Array with N elements and SKEY is a given item of information to search. The variables START, END and MID denote, respectively, the beginning, end and middle locations of a segments of element of **A**. This algorithm finds the location LOC of SKEY in **A** or sets LOC= NULL.

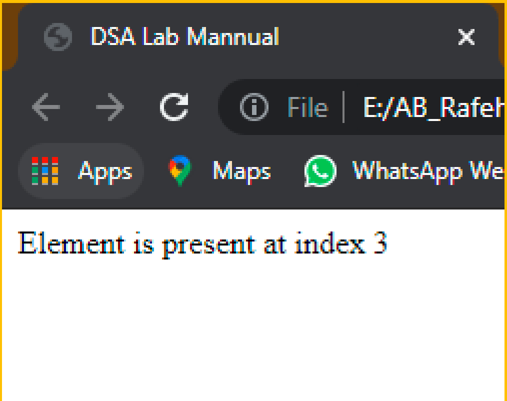
```
BinarySearch (A, SKEY)
1. [Initialize segment variables.]
   Set START:=0, END:=N-1 and MID=INT((START+END)/2).
2. Repeat Steps 3 and 4 while START ≤ END AND A[MID]≠SKEY.
3.         If SKEY< A[MID], then:
                   Set END:=MID-1.
           Else
                   Set START:=MID+1.
           [End of If Structure.]
4.         Set MID:=INT((START +END)/2).
   [End of Step 2 loop.]
5. If A[MID]= SKEY, then:
           Set LOC:= MID
   Else:
           Set LOC := NULL
   [End of If structure.]
6. Exit
```

LAB 2A

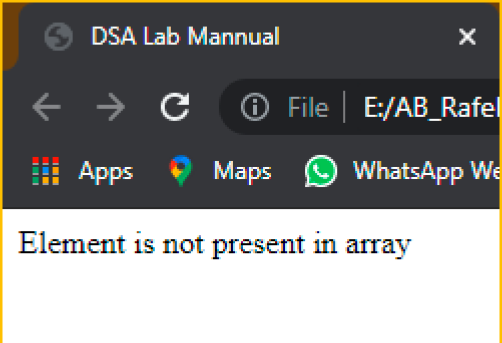


```
function search(arr, n, x) {
    let i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

let arr = [2, 3, 4, 10, 40];
let x = 10;
let n = arr.length;

let result = search(arr, n, x);
(result == -1) ?
document.write("Element is not present in array"):
    document.write("Element is present at index " + result);
```

Output:

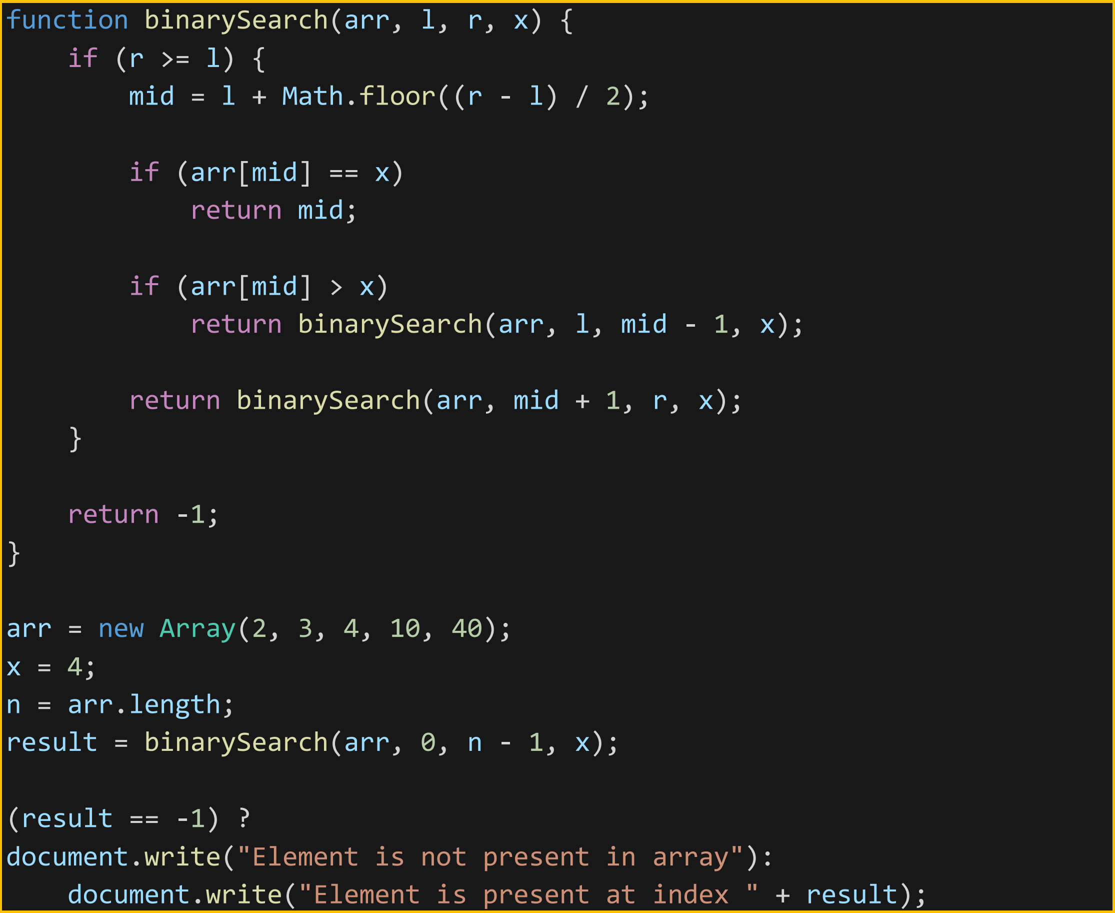


```
Element is present at index 3
```



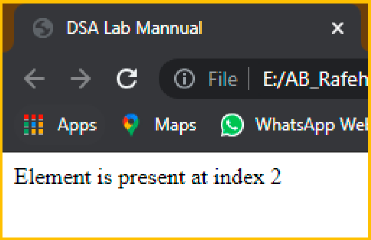
```
Element is not present in array
```

LAB 2B

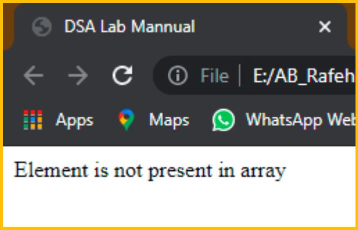


```
function binarySearch(arr, l, r, x) {
    if (r >= l) {
        mid = l + Math.floor((r - l) / 2);

        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        return binarySearch(arr, mid + 1, r, x);
    }

    return -1;
}

arr = new Array(2, 3, 4, 10, 40);
x = 4;
n = arr.length;
result = binarySearch(arr, 0, n - 1, x);

(result == -1) ?
document.write("Element is not present in array"):
    document.write("Element is present at index " + result);
```

Output:



```
Element is present at index 2
```



```
Element is not present in array
```

LAB No: 3

Objective

Perform Bubble Sort

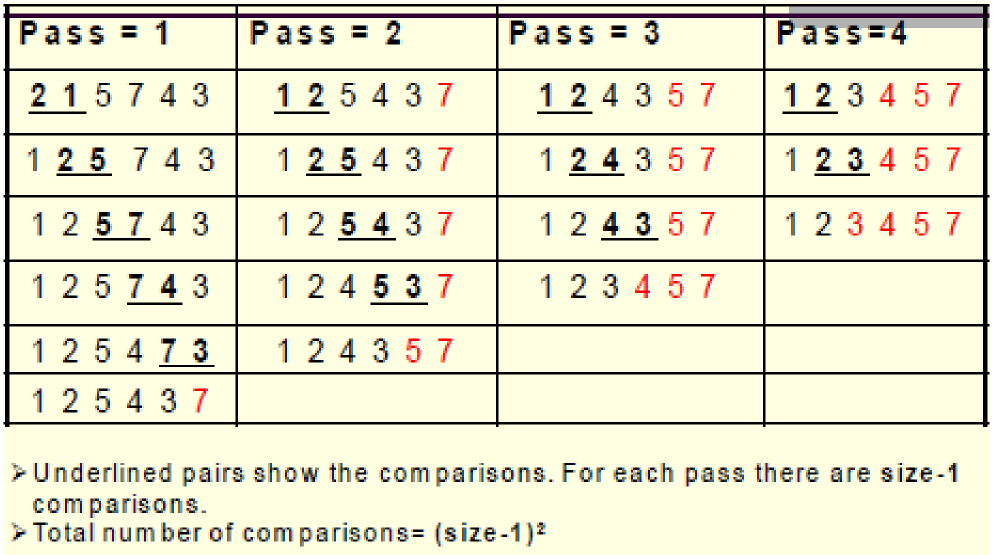
Theory

Bubble Sort:

The technique we use is called *"Bubble Sort"* because the bigger value gradually bubbles their way up to the top of array like air bubble rising in water, while the small values sink to the bottom of array.

It refers to the operation of rearranging to the elements of Array DATA so that they are in increasing order, i.e., so that DATA[1]<DATA[2]<DATA[3]<DATA[N]

This technique is to make several passes through the array. On each pass, successive pairs of elements are compared. If a pair is in increasing order (or the values are identical), we leave the values as they are. If a pair is in decreasing order, their values are swapped in the array.



Pass = 1	Pass = 2	Pass = 3	Pass=4
2 1 5 7 4 3	**1 2** 5 4 3 7	**1 2** 4 3 5 7	**1 2** 3 4 5 7
1 **2 5** 7 4 3	1 **2 5** 4 3 7	1 **2 4** 3 5 7	1 **2 3** 4 5 7
1 2 **5 7** 4 3	1 2 **5 4** 3 7	1 2 **4 3** 5 7	1 2 3 4 5 7
1 2 5 **7 4** 3	1 2 4 **5 3** 7	1 2 3 4 5 7	
1 2 5 4 **7 3**	1 2 4 3 5 7		
1 2 5 4 3 7			

- Underlined pairs show the comparisons. For each pass there are size-1 comparisons.
- Total number of comparisons= (size-1)^2

Algorithm (Bubble Sort) BUBBLE (DATA, N)

Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

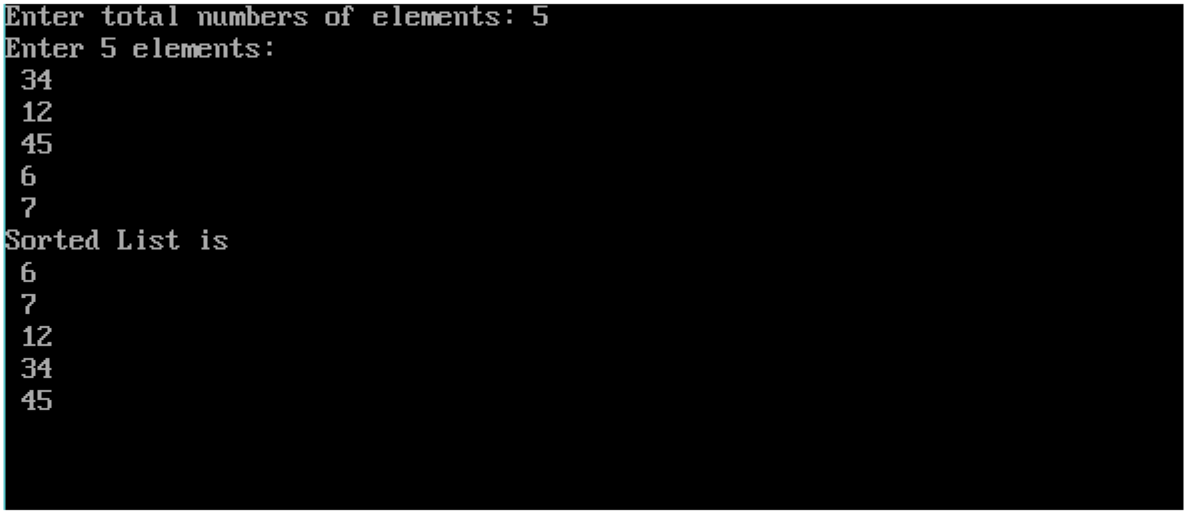
```
1.  Repeat step 2 and 3 for K=1 to N-1
2.  Set PTR :=1.      [Initialize pass pointer PTR.]
3.      Repeat While PTR ≤ N-K:        [Executes pass.]
            If DATA[PTR] > DATA[PTR+1], then
            Interchange DATA[PTR] and DATA[PTR+1]
            [End of If structure]
        [End of Step 3 inner loop.]
            Set PTR:= PTR+1.
    [End of Step 1 outer loop.]
4.  Exit.
```

Bubble Sort Program

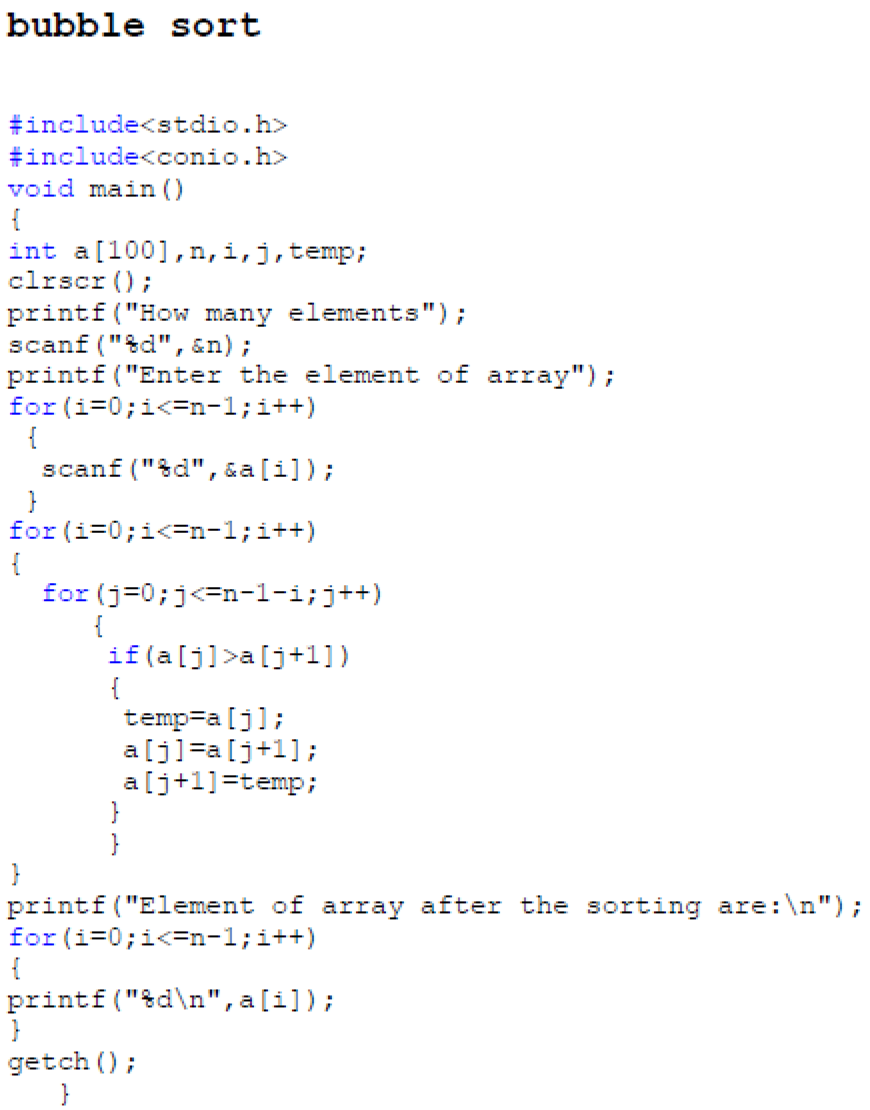
```
#include<stdio.h>
#include<conio.h>
void main(void)
{
   clrscr();
   int n,temp,i,j,a[20];

   printf("Enter total numbers of elements: ");
   scanf("%d",&n);

   printf("Enter %d elements: ",n);
   for(i=0;i<n;i++)
       scanf("%d",&a[i]);
   //Bubble sorting algorithm
   for(i=n-2;i>=0;i--){           // no. of passes
       for(j=0;j<=i;j++){         // no. of comparisons
            if(a[j]>a[j+1]){
                temp=a[j];
               a[j]=a[j+1];
               a[j+1]=temp;
            }
       }
```

```
    }


    printf("After sorting: ");

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

        printf(" %d",a[i]);

    getch();

}
```

Out Put



```
Enter total numbers of elements: 5
Enter 5 elements:
 34
 12
 45
 6
 7
Sorted List is
 6
 7
 12
 34
 45
```



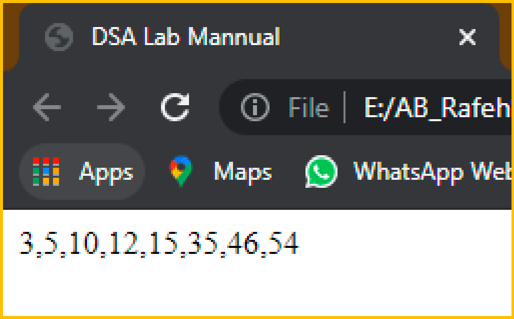
bubble sort

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],n,i,j,temp;
clrscr();
printf("How many elements");
scanf("%d",&n);
printf("Enter the element of array");
for(i=0;i<=n-1;i++)
 {
  scanf("%d",&a[i]);
 }
for(i=0;i<=n-1;i++)
{
  for(j=0;j<=n-1-i;j++)
      {
       if(a[j]>a[j+1])
       {
        temp=a[j];
        a[j]=a[j+1];
        a[j+1]=temp;
       }
       }
}
printf("Element of array after the sorting are:\n");
for(i=0;i<=n-1;i++)
{
printf("%d\n",a[i]);
}
getch();
   }
```

LAB 3

```
function bblSort(arr) {

    for (var i = 0; i < arr.length; i++) {

        for (var j = 0; j < (arr.length - i - 1); j++) {

            if (arr[j] > arr[j + 1]) {

                var temp = arr[j]
                arr[j] = arr[j + 1]
                arr[j + 1] = temp
            }
        }
    }
    document.write(arr);
}


// unsorted array
var arr = [12, 10, 5, 3, 15, 46, 35, 54];

bblSort(arr);
```

Output:



```
3,5,10,12,15,35,46,54
```

Lab 4

Objective: To understand the basic concepts of **Insertion** and **Deletion** in array.

Theory

Insert Operation on Array and Delete Operation on Array is part of Learning Data Structure Series. In the previous article of Array Data Structure we learnt about arrays and different types of array data structures.

Insert Operation on Array (Algorithm to Insert Element in un-ordered Array)
The term insert operation on array refers to adding element to an array at a particular location. We can insert element in array at any location. Algorithm to insert element in array requires the specific location at which the element is to be added.

Here DATA is an unsorted array stored in memory with N elements. This algorithm inserts a data element ITEM into the loc^{th} position in an array. DATA. The first four steps create space in DATA by moving downward the elements of DATA. These elements are used in reverese order i.e. first DATA[N], then DATA[N-1], DATA[N-2]…….and last DATA[LOC], otherwise data will be overwritten. We first set I=N and then, using I as a counter, decrease it each time the loop is executed until I reaches LOC. In the next step, Step 5, it inserts ITEM into the array in the space just created. And at last, the total number of elements N is increased by 1.

<u>Algorithm to Insert Element in un-ordered Array is as follows:</u>
DATA is unsorted linear array with N elements. LOC is the location where ITEM is to be inserted where $LOC \leq N$. This Algorithm inserts an element ITEM into the LOCth position in array DATA.

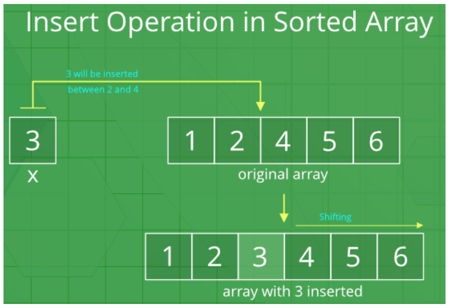
INSERT (DATA, N, I, ITEM, LOC)

```
1. Set I:=N                              [Initialize counter]
2. Repeat the step 3 and 4 while I ≥ LOC
3.           Set DATA[I+1]:=DATA[I]      [Move Ith elements downward or forward]
4.           Set I=I-1                   [Decrease counter by 1]
   [End of while loop]
5. Set DATA[LOC]=ITEM                    [Insert Element]
6. Set N=N+1                             [Reset N]
7. Exit
```

<u>Delete Operation on Array (Algorithm to Delete Element from Array)</u>

This Algorithm deletes LOCth element from array DATA and assign it to a variable ITEM.
To delete element from array we should know the location from where we want to delete. Algorithm to Delete Element from Array can be executed at any location ie. at it beginning, in middle or at the end of the array.

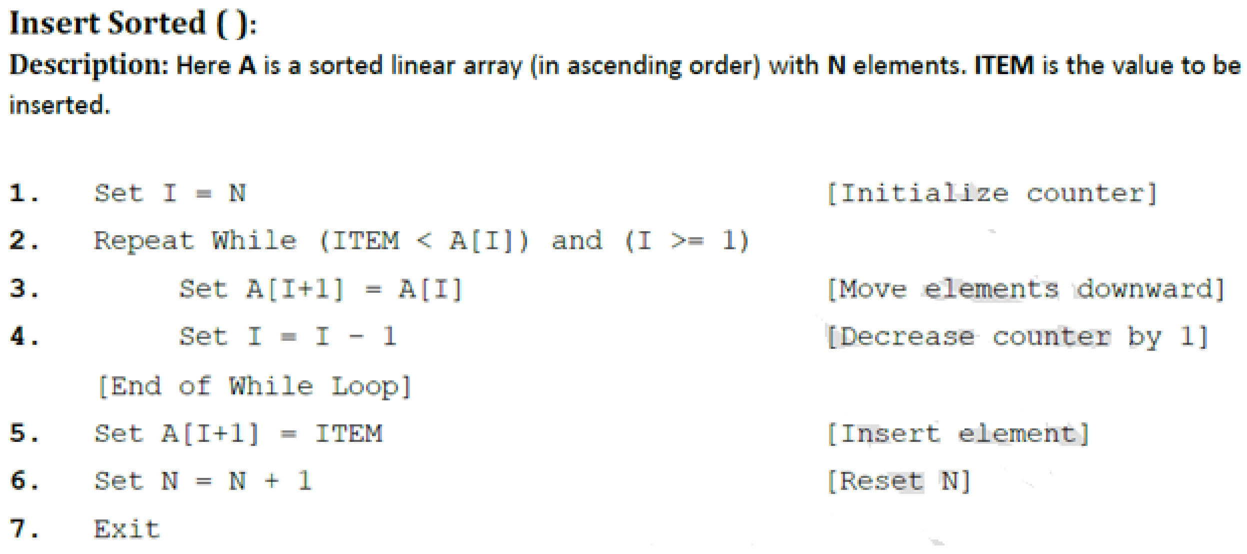
First, the Element to be deleted is assigned to ITEM from location DATA[LOC]. Then I is set to LOC from where ITEM is to be deleted and it iterated to total number of elements i.e. N. During this loop, the elemnts are moved upwards. And lastly, total number of elements is decreased by 1.



<u>Algorithm to Delete Element from Array is:</u>

DELETE (DATA, N, I, ITEM, LOC)

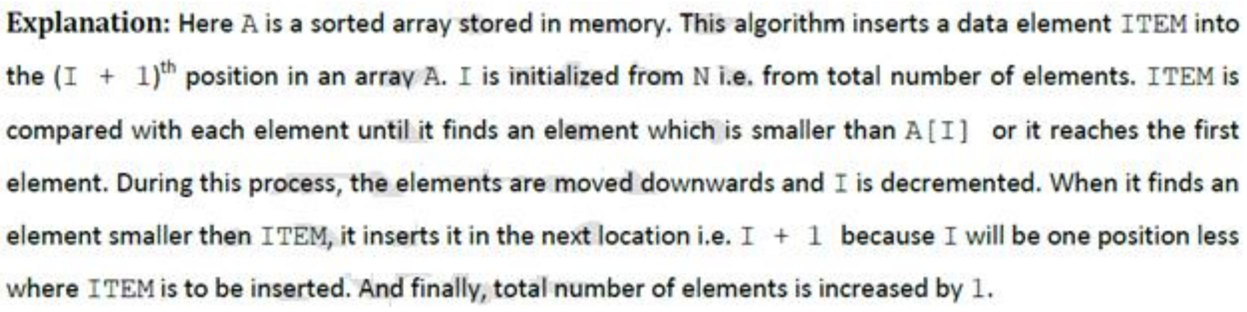
```
1.Set ITEM=DATA[LOC]                [Assign en Element to be deleted to ITEM]
2. Repeat for I=LOC to N-1
3.          Set DATA[I]:=DATA[I+1]   [Move the I+1st Element upward or backward]
    [End of For loop]
4. Set N:=N-1                        [Reset N]
5. Exit.
```



Insert Sorted ( ):

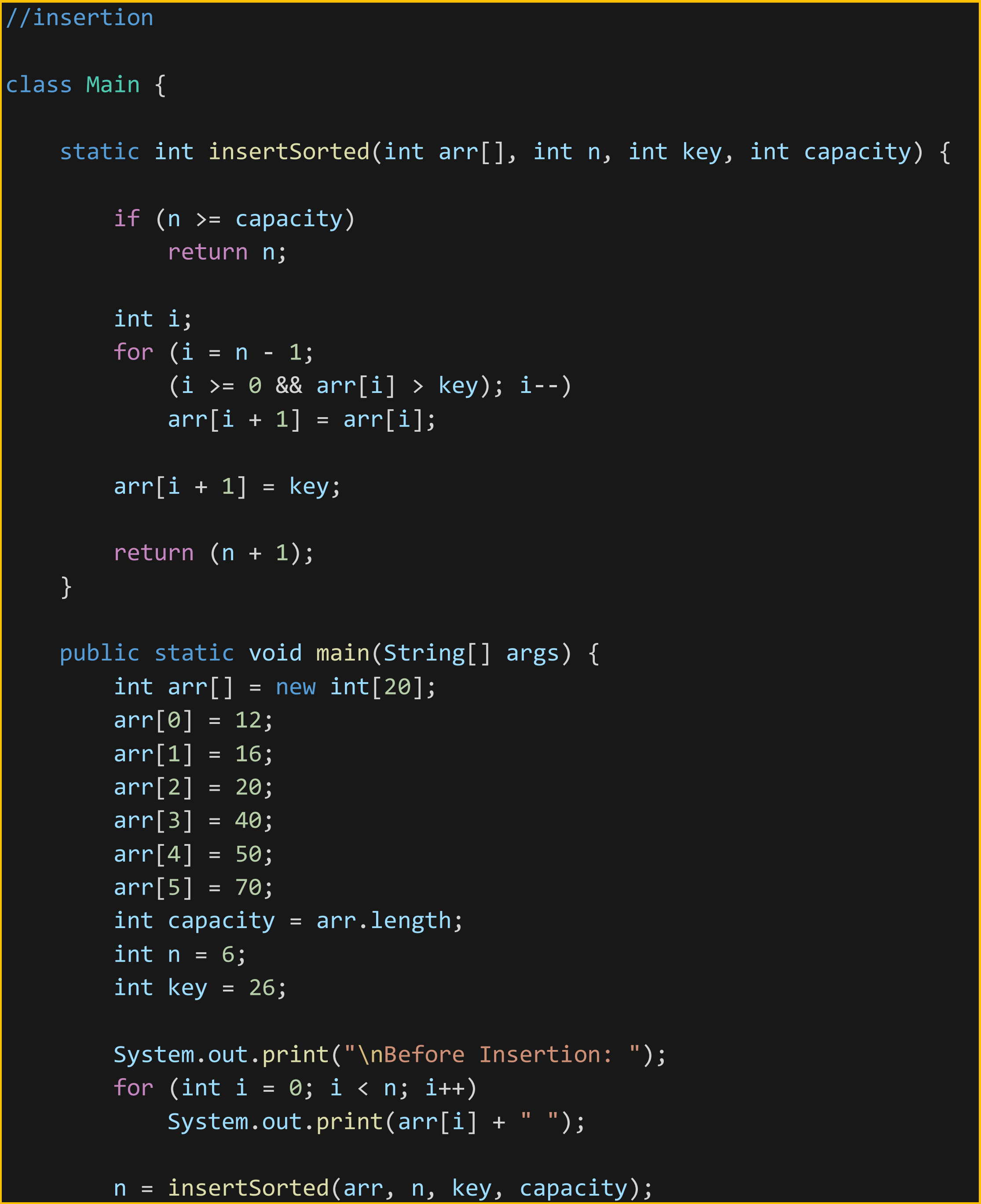
Description: Here **A** is a sorted linear array (in ascending order) with **N** elements. **ITEM** is the value to be inserted.

```
1.    Set I = N                                  [Initialize counter]
2.    Repeat While (ITEM < A[I]) and (I >= 1)
3.          Set A[I+1] = A[I]                    [Move elements downward]
4.          Set I = I - 1                        [Decrease counter by 1]
      [End of While Loop]
5.    Set A[I+1] = ITEM                          [Insert element]
6.    Set N = N + 1                              [Reset N]
7.    Exit
```

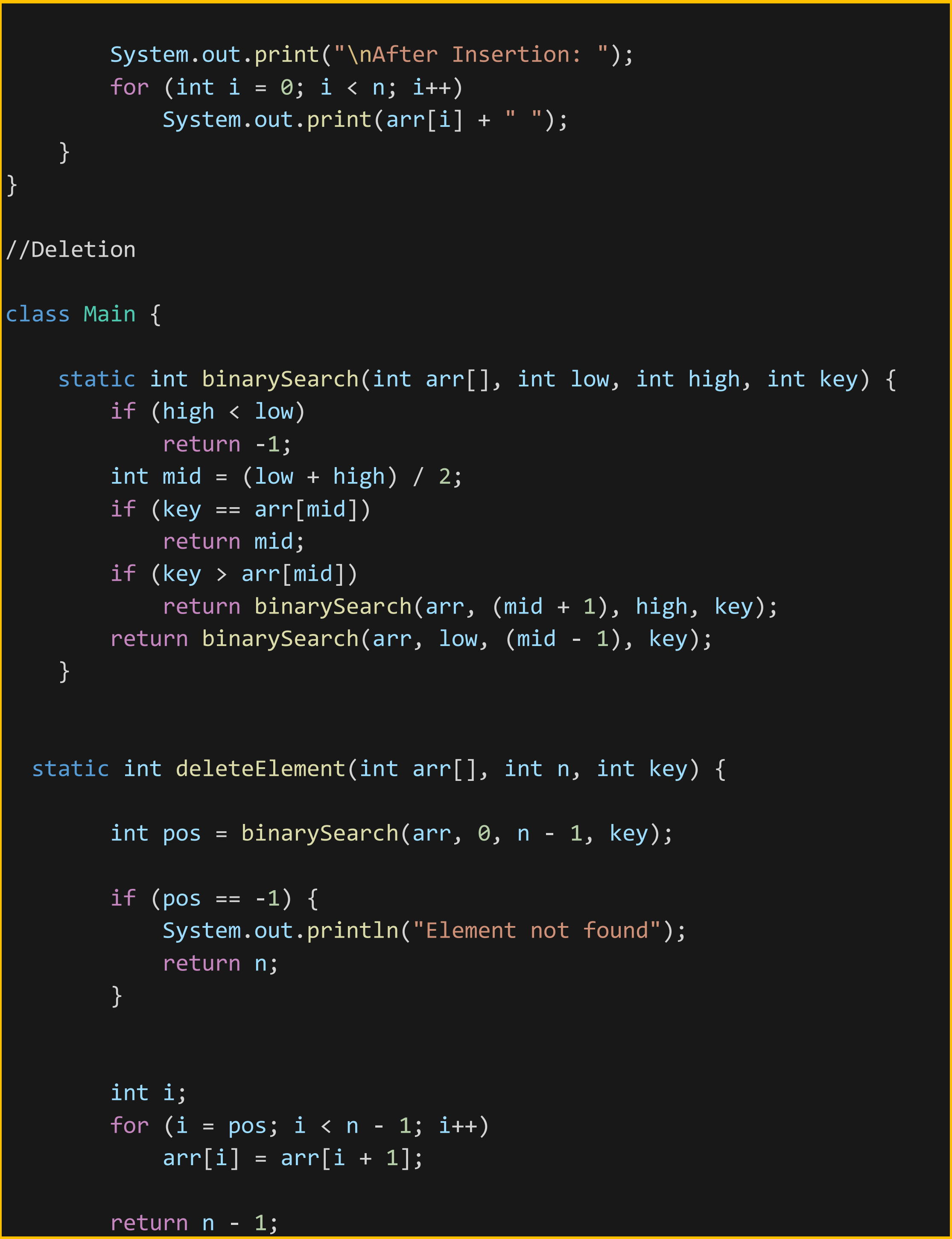


Explanation: Here A is a sorted array stored in memory. This algorithm inserts a data element ITEM into the (I + 1)th position in an array A. I is initialized from N i.e. from total number of elements. ITEM is compared with each element until it finds an element which is smaller than A[I] or it reaches the first element. During this process, the elements are moved downwards and I is decremented. When it finds an element smaller then ITEM, it inserts it in the next location i.e. I + 1 because I will be one position less where ITEM is to be inserted. And finally, total number of elements is increased by 1.

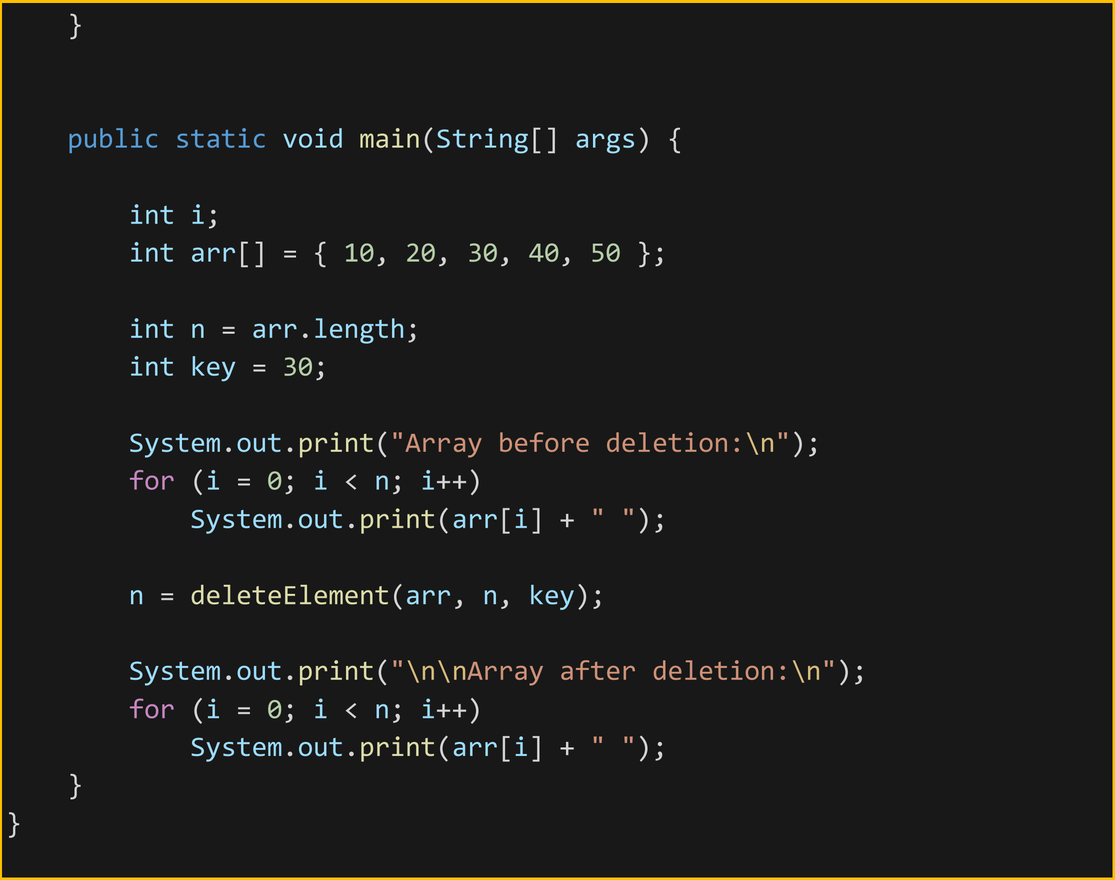
LAB 4



```
//insertion

class Main {

    static int insertSorted(int arr[], int n, int key, int capacity) {

        if (n >= capacity)
            return n;

        int i;
        for (i = n - 1;
            (i >= 0 && arr[i] > key); i--)
            arr[i + 1] = arr[i];

        arr[i + 1] = key;

        return (n + 1);
    }

    public static void main(String[] args) {
        int arr[] = new int[20];
        arr[0] = 12;
        arr[1] = 16;
        arr[2] = 20;
        arr[3] = 40;
        arr[4] = 50;
        arr[5] = 70;
        int capacity = arr.length;
        int n = 6;
        int key = 26;

        System.out.print("\nBefore Insertion: ");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");

        n = insertSorted(arr, n, key, capacity);
```

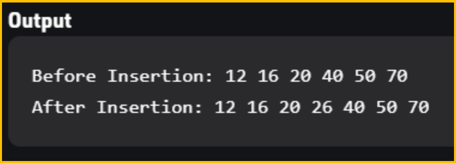


```java
        System.out.print("\nAfter Insertion: ");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }
}

//Deletion

class Main {

    static int binarySearch(int arr[], int low, int high, int key) {
        if (high < low)
            return -1;
        int mid = (low + high) / 2;
        if (key == arr[mid])
            return mid;
        if (key > arr[mid])
            return binarySearch(arr, (mid + 1), high, key);
        return binarySearch(arr, low, (mid - 1), key);
    }


  static int deleteElement(int arr[], int n, int key) {

        int pos = binarySearch(arr, 0, n - 1, key);

        if (pos == -1) {
            System.out.println("Element not found");
            return n;
        }


        int i;
        for (i = pos; i < n - 1; i++)
            arr[i] = arr[i + 1];

        return n - 1;
```

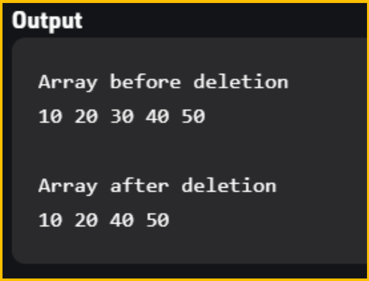


```
    }


    public static void main(String[] args) {

        int i;
        int arr[] = { 10, 20, 30, 40, 50 };

        int n = arr.length;
        int key = 30;

        System.out.print("Array before deletion:\n");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");

        n = deleteElement(arr, n, key);

        System.out.print("\n\nArray after deletion:\n");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }
}
```

Output:



```
Before Insertion: 12 16 20 40 50 70
After Insertion: 12 16 20 26 40 50 70
```



```
Array before deletion
10 20 30 40 50

Array after deletion
10 20 40 50
```

LAB No: 5A

Objective:

Sort an array using Selection Sort.

Selection Sort:

Selection sort performs sorting by repeatedly putting the largest element in the unprocessed portion of the array to the end of the unprocessed portion until the whole array is sorted.

Suppose an array A with N elements A[0], A[1], . . . A[N-1] Is in memory. The Selection sort algorithm for sorting ***A*** works as follows. First find the smallest element in the list and put it in the first position. Then find the second smallest element in the list and put it the second position. And so on.

Pass 1: Find the location LOC of the smallest in the list of N elements A[0], A[1], ... A[N-1], and then interchange A[LOC] and A[0]. Then: A[0] is sorted.

Pass 2: Find the location LOC of the smallest in the sublist of N-1 elements A[1], A[2], . . . A[N-1], and interchange A[LOC] and A[1]. Then: A[0], A[1] is sorted. Since A[0] ≤ A[1].

. . . .

. . . .

Pass N-1: Find the location LOC of the smallest A[N-2] and A[N-1], and then interchanged A[LOC] and A[N-1]. Then: A[0], A[1], A[2], ... A[N-1] is sorted.

Algorithm: MIN(A, K, N, LOC)

An Array A is in memory. This procedure finds the location LOC of the smallest element among A[K], A[K+1], ...A[N-1].

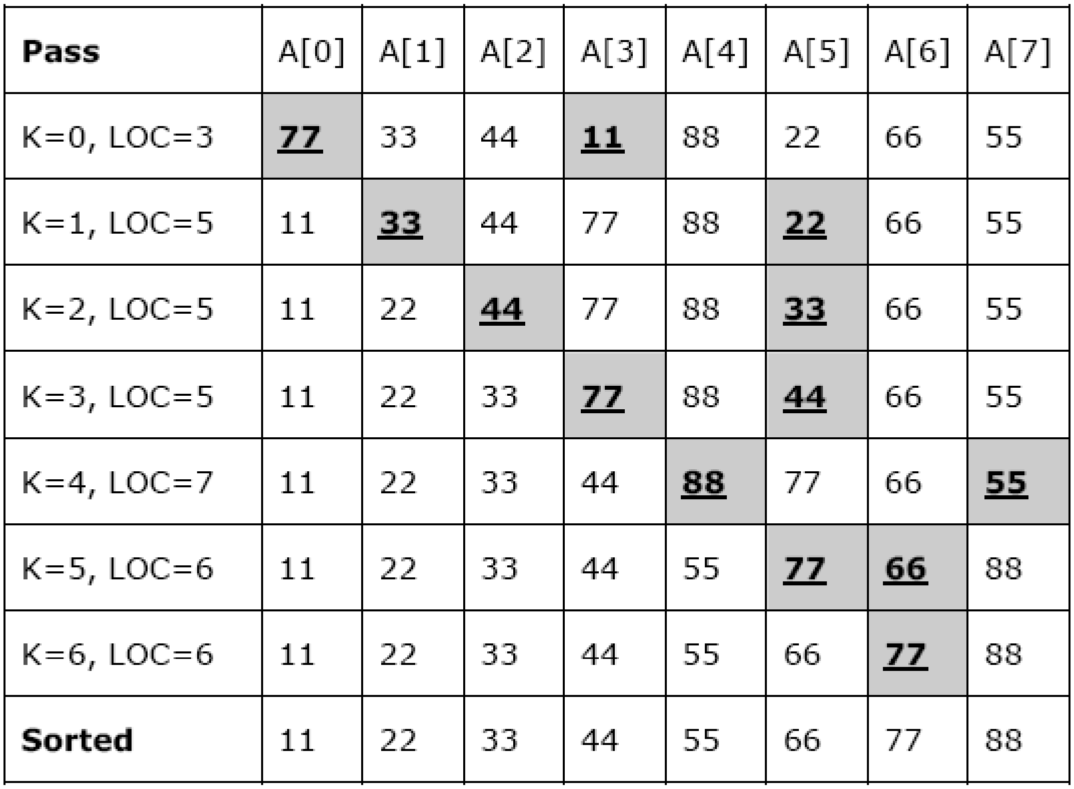
```
1. Set MIN := A[K] and LOC := K.            [Initializes pointers: min value=A[K], MIN index=K]
2. Repeat for J=K+1, K+2, . . . . ., N-1:       [Pass 1 to N-1: loop from next to MIN to till end]
        If MIN > A[J], then:          [compare every element (from next to MIN) till end with MIN]
        MIN := A[J] and LOC:=J.                [minimum value=A[J], minimum index=J
   [End of loop.]
3. Return LOC.
```

The selection sort algorithm can now be easily stated:

Algorithm: (Selection Sort) SELECTION (A, N)

This algorithm sorts the array A with N elements.

```
1. Repeat steps 2 and 3 for K=0 to N-2:          [Find LOC of the smallest element from A[0] to
                                                          A[N-2]]
2.     Call MIN(A, K, N, LOC).             [call function MIN to find minimum element in array A]
3.     [Interchange A[K] and A[LOC].]
       Set TEMP: = A[K], A[K] = A[LOC] and A[LOC] := TEMP.
   [End of step 1 loop.]
4. Exit.
```



Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
K=0, LOC=3	**77**	33	44	**11**	88	22	66	55
K=1, LOC=5	11	**33**	44	77	88	**22**	66	55
K=2, LOC=5	11	22	**44**	77	88	**33**	66	55
K=3, LOC=5	11	22	33	**77**	88	**44**	66	55
K=4, LOC=7	11	22	33	44	**88**	77	66	**55**
K=5, LOC=6	11	22	33	44	55	**77**	**66**	88
K=6, LOC=6	11	22	33	44	55	66	**77**	88
Sorted	11	22	33	44	55	66	77	88

LAB No: 5B

Objective:

Sort an array using Insertion Sort.

Insertion sort:

Insertion sort is an elementary sorting algorithm. Insertion sort is well suited for sorting small data sets or for the insertion of new elements into a sorted sequence.

Suppose an array ***A*** with ***N*** elements ***A***[0], ***A***[1], ... ***A***[N-1] is in memory. The insertion sort algorithm scan A from ***A***[0] to ***A***[N-1], inserting each element ***A***[K] into its proper position in the previously sorting sub array A[0], A[1], ...A[K- 1]. That is:

Pass 1: A[1] is inserted either before A[2] or after A[0] so that: A[0], A[1] is sorted.

Pass 2: A[2] is inserted into its proper place in A[0], A[1], so that A[0], A[1], A[2] are sorted.

Pass 3: A[3] is inserted into its proper place in A[0], A[1], A[2] so that: A[0], A[1], A[2], A[3] are sorted.

. . . .

Pass N-1: A[N-1] is inserted into its proper place in A[0], A[1], ... A[N-1] so that: A[0], A[1], ... A[N-1] are sorted.

This sorting algorithm is frequently used when ***N*** is small. There remains only the problem of deciding how to insert A[K] in its proper place in the sub array A[0], A[1], ... A[K-1]. This can be accomplished by comparing A[K] with A[K-1], comparing A[K] with A[K-2], comparing A[K] with A[K-3], and so on, until first meeting an element A[PTR] (where ***PTR*** start from k-1) such that A[PTR] ≤ A[K], then each of elements A[K-1], A[K-2], ... A[PTR+1] is moved forward one location, and A[K] is then inserted in the ***PTR*+1st** position in the array.

Following diagram illustrate Insertion Sort Algorithm.

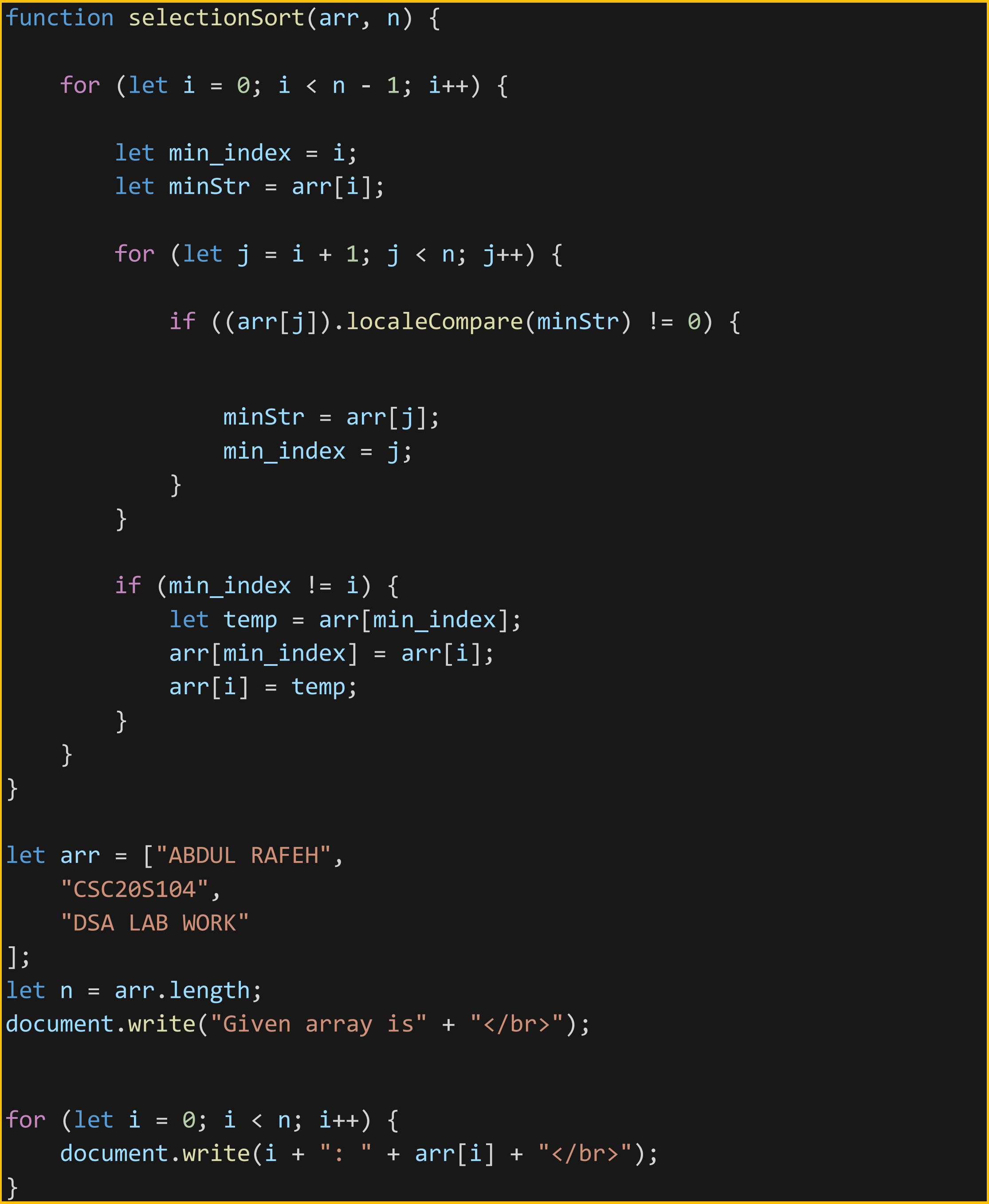
Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
	80	77	33	44	11	88	22	66	55
K=1	77	80	33	44	11	88	22	66	55
K=2	33	77	80	44	11	88	22	66	55
K=3	33	44	77	80	11	88	22	66	55
K=4	11	33	44	77	80	88	22	66	55
K=5	11	33	44	77	80	88	22	66	55
K=6	11	22	33	44	77	80	88	66	55
K=7	11	22	33	44	66	77	80	88	55
K=8	11	22	33	44	55	66	77	80	88
Sorted	11	22	33	44	55	66	77	80	88

The algorithm is simplified if there always is an element A[PTR] such that A[PTR] ≤ A[K]; otherwise we must constantly check to see if we are comparing A[K] with A[0].

Algorithm: (Insertion Sort) INSERTION (A, N)
This Algorithm sorts an array A with N elements.

```
1. Repeat steps 2 to 4 for K=1 to N-1:     [an outer loop which uses K as an Index from second
                                                element A[1] to A[N-1]to last element.]
2. Set TEMP := A[K] and PTR =K-1.
[an inner loop which is essentially controlled by the variable PTR to locate hole position for the element
to be inserted]
3. Repeat while PTR>= 0 and TEMP < A[PTR]     [check whether the adjacent element in left
side is greater or less than the current element]
      a) Set A[PTR+1] := A[PTR].     [Move all the left side elements one position forward, which
                                      are greater than key TEMP.]
      b) Set PTR := PTR -1.
   [End of step 3 loop.]
4. Set A[PTR+1] :=TEMP.     [Move (Insert) current element in its proper place (hole position).]
   [End of Step 1 loop.]
5  Return.
```

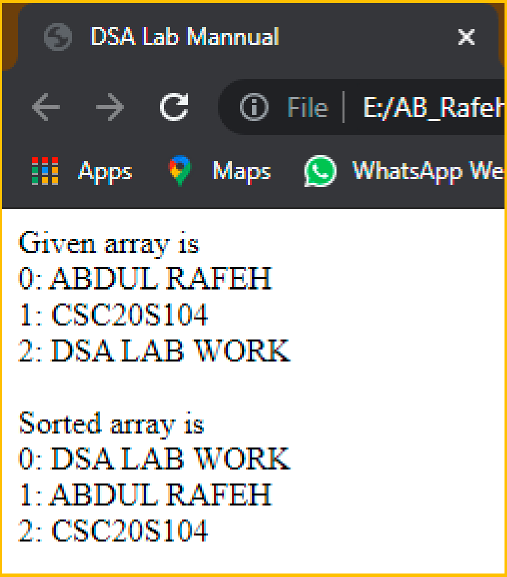
LAB 5A



```javascript
function selectionSort(arr, n) {

    for (let i = 0; i < n - 1; i++) {

        let min_index = i;
        let minStr = arr[i];

        for (let j = i + 1; j < n; j++) {

            if ((arr[j]).localeCompare(minStr) != 0) {


                minStr = arr[j];
                min_index = j;
            }
        }

        if (min_index != i) {
            let temp = arr[min_index];
            arr[min_index] = arr[i];
            arr[i] = temp;
        }
    }
}

let arr = ["ABDUL RAFEH",
    "CSC20S104",
    "DSA LAB WORK"
];
let n = arr.length;
document.write("Given array is" + "</br>");


for (let i = 0; i < n; i++) {
    document.write(i + ": " + arr[i] + "</br>");
}
```

```
document.write("</br>");

selectionSort(arr, n);

document.write("Sorted array is" + "</br>");


for (let i = 0; i < n; i++) {
    document.write(i + ": " + arr[i] + "</br>");
}
```

Output:

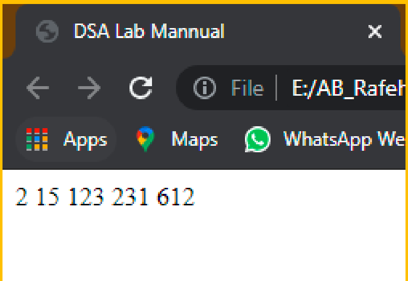


```
Given array is
0: ABDUL RAFEH
1: CSC20S104
2: DSA LAB WORK

Sorted array is
0: DSA LAB WORK
1: ABDUL RAFEH
2: CSC20S104
```

LAB 5B

```
function insertionSort(arr, n) {
    let i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

function printArray(arr, n) {
    let i;
    for (i = 0; i < n; i++)
        document.write(arr[i] + " ");
    document.write("<br>");
}

let arr = [612, 231, 123, 15, 2];
let n = arr.length;

insertionSort(arr, n);
printArray(arr, n);
```

Output:



```
2 15 123 231 612
```

LAB 6

Objective

Perform Matrix Addition, Subtraction, Multiplication and Transpose operations.

Theory

Matrices

A **matrix** is a collection of numbers arranged into a fixed number of rows and columns. Usually the numbers are real numbers. In general, matrices can contain complex numbers but we won't see those here. Here is an example of a matrix with three rows and three columns.

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

Vectors and Matrices are mathematical terms, which refer to collections of numbers which are analogous, respectively, to linear and two-dimensional arrays. That is,

(a) An n-element vector is V is list of n numbers usually given in the form
$V = (V_1, V_2, \ldots\ldots, V_n)$

(b) An m x n matrix A is an array of m . n numbers arranged in m rows and n columns as follows:

$$A = \begin{bmatrix} A_{11}\ A_{12} \ldots\ldots A_{1n} \\ A_{21}\ A_{22} \ldots\ldots A_{2n} \\ A_{31}\ A_{32} \ldots\ldots A_{3n} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ A_{m1}\ A_{m2} \ldots\ldots A_{mn} \end{bmatrix}$$

A ***matrix*** is a rectangular array of numbers. For example,

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

is a 2 × 3 matrix $A = (a_{ij})$, where for $i = 1, 2$ and $j = 1, 2, 3$, the element of the matrix in row i and column j is a_{ij}. We use uppercase letters to denote matrices and corresponding subscripted lowercase letters to denote their elements. The set of all $m \times n$ matrices with real-valued entries is denoted $\mathbf{R}^{m \times n}$. In general, the set of $m \times n$ matrices with entries drawn from a set S is denoted $S^{m \times n}$.

The ***transpose*** of a matrix A is the matrix A^T obtained by exchanging the rows and columns of A. For the matrix A ,

$$A^{\mathrm{T}} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Matrix Multiplication Rule: We do not need to have two matrices of the same size to multiply them. We can multiply a (2x2) matrix with a (2x1) matrix (which gave a (2x1) matrix). In fact, the general rule says that in order to perform the multiplication *AB*, where *A* is a (mxn) matrix and *B* a (kxl) matrix, then we must have *n*=*k*. The result will be a (mxl) matrix.

Matrix Multiplication Algorithm

Matmula (A,B,C,M,P,N)

Let A be an M X P matrix array, and let B be a P X N matrix array. This algorithm stores the product of A and B in an MXN matrix array C.

```
1. Repeat step 2 to 4 for I=1 to M:
2.     Repeat step 3 AND 4 for J=1 to N:
3.         Set C[I,J]:=0.                  [Initializes C[I,J]. ]
4.         Repeat for K=1 to P:            [loop grants the no. of columns of second matrix]
               C[I,J]:=C[I,J]+A[I,K]*B[K,J]   [no. of rows of second matrix =no. of
                                                columns of first matrix]
           [End of inner loop.]
       [End of step 2 middle loop.]
   [End of step 1 outer loop.]
5. Exit.
```

Matrix Addition Algorithm

MatAdd(A,B,C,M,N)

```
1 Repeat for I=1; to M
2     Repeat for J=1 to N
          C[I][J]= A[I][J]+ B[I][J]
      [End of inner loop.]
  [End of outer loop.]
3 Exit.
```

Transpose of Matrix Algorithm

Transpose (A,M,N)

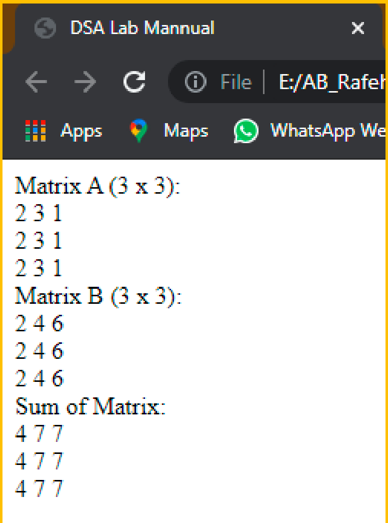
```
1 Repeat for I=1; to M
2     Repeat for J=1 to N
          C[J][I]= A[I][J]
      [End of inner loop.]
  [End of outer loop.]
3 Exit
```

LAB 6

Addition Matrix:

```
var i, j;
var arr1 = [
    [2, 3, 1],
    [2, 3, 1],
    [2, 3, 1]
];
var arr2 = [
    [2, 4, 6],
    [2, 4, 6],
    [2, 4, 6]
];
var arr3 = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0],
];

document.write("Matrix A (3 x 3):<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
        document.write(arr1[i][j] + " ");
    document.write("<br>");
}

document.write("Matrix B (3 x 3):<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
        document.write(arr2[i][j] + " ");
    document.write("<br>");
}

document.write("Sum of Matrix:<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
        arr3[i][j] = arr1[i][j] + arr2[i][j];
        document.write(arr3[i][j] + " ");
    }
    document.write("<br>");
}
```

Output:

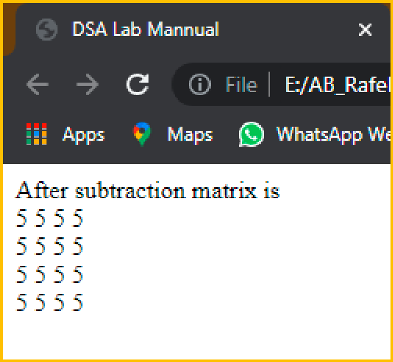


```
Matrix A (3 x 3):
2 3 1
2 3 1
2 3 1
Matrix B (3 x 3):
2 4 6
2 4 6
2 4 6
Sum of Matrix:
4 7 7
4 7 7
4 7 7
```

Subtraction Matrix:

```
var N = 4;

function subtract(A, B, C) {
    var i, j;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            C[i][j] = A[i][j] - B[i][j];
}


var A = [
    [6, 6, 6, 6],
    [7, 7, 7, 7],
    [8, 8, 8, 8],
    [9, 9, 9, 9]

];
var B = [
    [1, 1, 1, 1],
    [2, 2, 2, 2],
    [3, 3, 3, 3],
    [4, 4, 4, 4]
];
var C = Array.from(Array(N), () => Array(N));
var i, j;
subtract(A, B, C);
document.write("After subtraction matrix is " + "<br>");
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++)
        document.write(C[i][j] + " ");
    document.write("<br>");
}
```

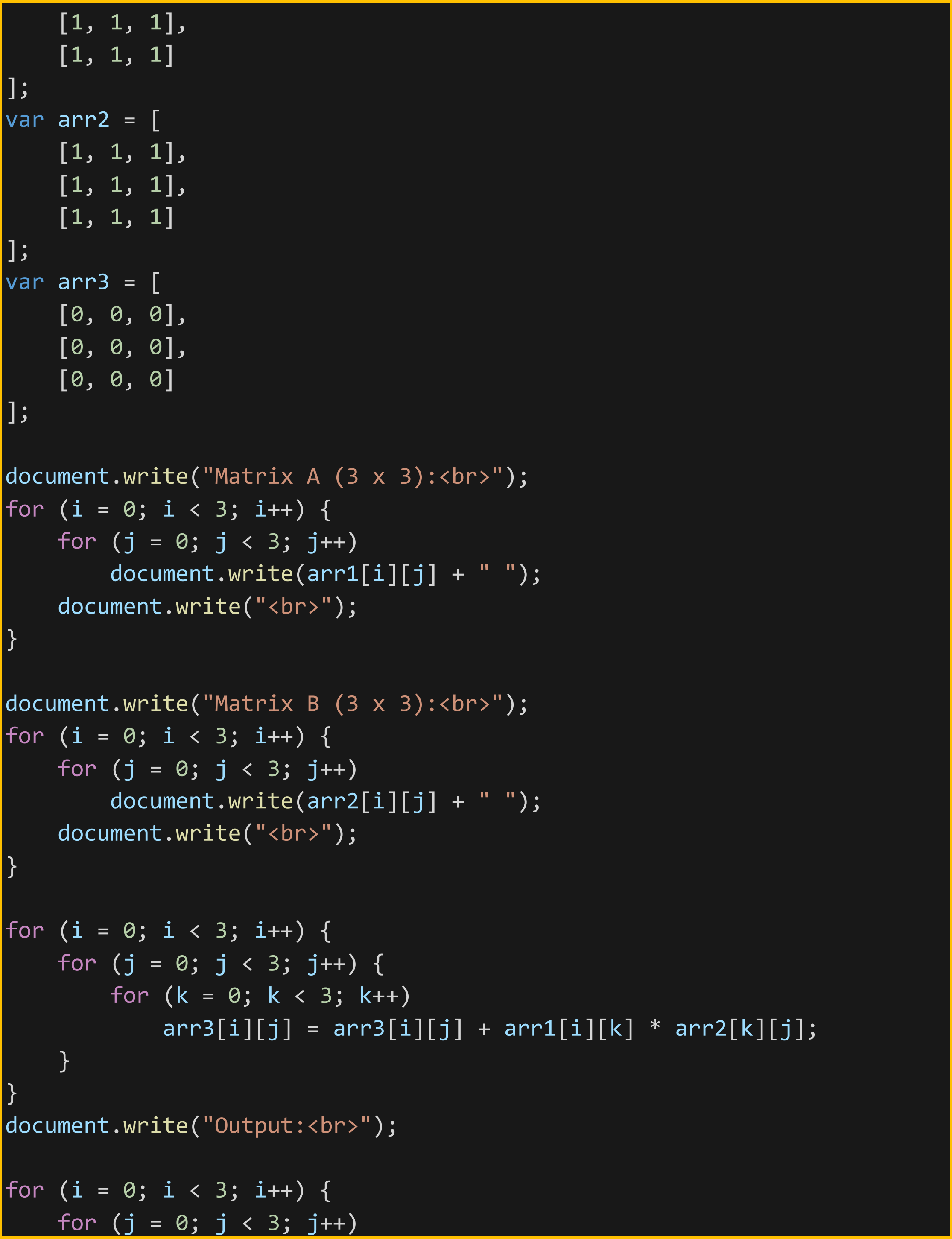
Output:



```
After subtraction matrix is
5 5 5 5
5 5 5 5
5 5 5 5
5 5 5 5
```

Multiplication Matrix:

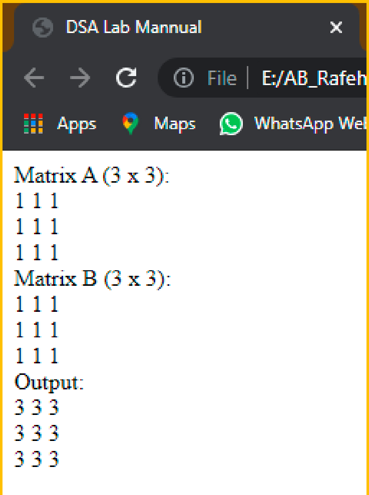
```
var i, j, k;
var arr1 = [
    [1, 1, 1],
```



```
    [1, 1, 1],
    [1, 1, 1]
];
var arr2 = [
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
];
var arr3 = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
];

document.write("Matrix A (3 x 3):<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
        document.write(arr1[i][j] + " ");
    document.write("<br>");
}

document.write("Matrix B (3 x 3):<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
        document.write(arr2[i][j] + " ");
    document.write("<br>");
}

for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
        for (k = 0; k < 3; k++)
            arr3[i][j] = arr3[i][j] + arr1[i][k] * arr2[k][j];
    }
}
document.write("Output:<br>");

for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
```

```
        document.write(arr3[i][j] + " ");
    document.write("<br>");
}
```

Output:

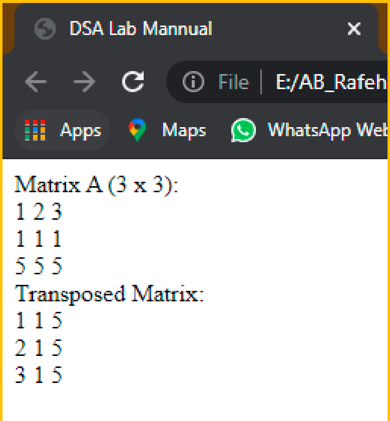


```
Matrix A (3 x 3):
1 1 1
1 1 1
1 1 1
Matrix B (3 x 3):
1 1 1
1 1 1
1 1 1
Output:
3 3 3
3 3 3
3 3 3
```

Transpose Matrix:

```
var i, j, n;
var arr = [
    [1, 2, 3],
    [1, 1, 1],
    [5, 5, 5]
];

document.write("Matrix A (3 x 3):<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
        document.write(arr[i][j] + " ");
    document.write("<br>");
}

for (i = 0; i < 3; i++) {
    for (j = i + 1; j < 3; j++) {
        n = arr[i][j];
        arr[i][j] = arr[j][i];
        arr[j][i] = n;
    }
}

document.write("Transposed Matrix:<br>");
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++)
        document.write(arr[i][j] + " ");
    document.write("<br>");
}
```

Output:



```
Matrix A (3 x 3):
1 2 3
1 1 1
5 5 5
Transposed Matrix:
1 1 5
2 1 5
3 1 5
```

LAB No: 7

Objective

Write program to create a Link List and perform different functionality related to it as stated below:

Operations on Link List

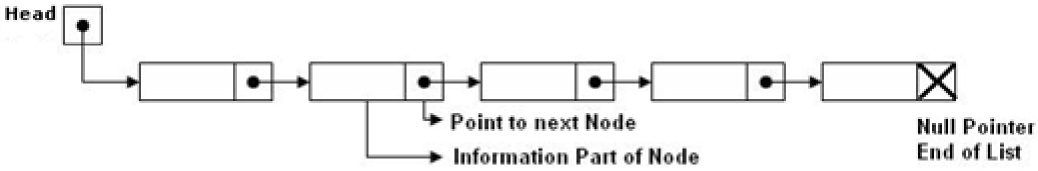
a) Create a Link List
b) Traversing Link List
c) Searching in Link List
d) Display a Link List

Theory

A linked list or one way list is a linear collection of data elements, called ***nodes***, where the linear order is given by means of ***"pointers"***. Each node is divided into two parts.

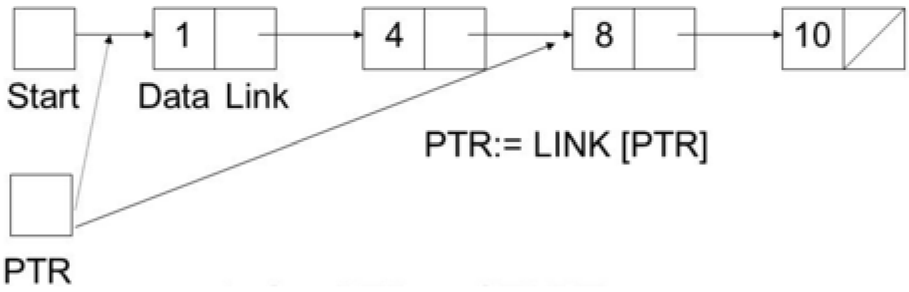
- The first part contains the information of the element / node.
- The second part called the link field contains the address of the next node in the list.

Example:



Traversing a Link List:

Initialize PTR or START. Then process INFO[PTR], the information at the first node. Update PTR by the assignment PTR :=LINK[PTR], so that PTR points to the second node. Then process INFO[PTR], the information at the second node. Again update PTR by the assignment PTR :=LINK[PTR], and then process INFO[PTR], the information at the third node and so on. Continue until PTR=NULL,which signals the end of the list.

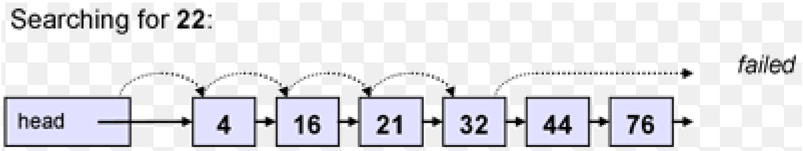


Algorithm: (Traversing a Link List) Let LIST be a link list in memory. This Algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR:=START.[Initializes pointer PTR.]
2. Repeat step 3 and 4 while PTR≠NULL.

```
3.        Apply PROCESS to INFO[PTR].
4.        Set PTR=LINK[PTR]. [PTR now points to the next node.]
      [End of Step 2 loop.]
5.    Exit
```

Searching a Link List:



List is Unsorted:
Suppose the data in LIST are not necessarily sorted. Then one searches for ITEM in LIST by traversing through the list using a pointer variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST. Before we update the pointer PTR by PTR:=LINKPTR], we require two tests. First we have to check to see whether we have reached the end of the LIST i.e. PTR=NULL. If not, then we check to see whether INFO[PTR]=ITEM.
The two tests cannot be performed at the same time, since INFO[PTR] is not defined when PTR=NULL. Accordingly we use the First test to control the execution of a loop, and we let the second test take place inside the loop.

Algorithm: SEARCH (INFO, LINK,START, ITEM, LOC)

LIST is a linked list in the memory. This algorithm finds the location LOC of the node where ITEM first appear in LIST, or sets LOC=NULL.

```
1. Set PTR:=START.
2. Repeat Step 3 while PTR≠NULL:
3.        If ITEM = INFO[PTR], then:
               Set LOC:=PTR, and Exit.  [Search is successful.]
          Else:
               Set PTR:=LINK[PTR].      [PTR now points to the next node]
          [End of If structure.]
   [End of Step 2 loop.]
4. Set LOC:=NULL.                       [Search is unsuccessful.]
5. Exit.
```

List is Sorted:
Suppose the data in LIST are sorted. Again we search for ITEM in LIST by traversing the list using a pointer variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST. Now, however, we can stop once ITEM exceeds INFO[PTR]. (List sorted in descending order)

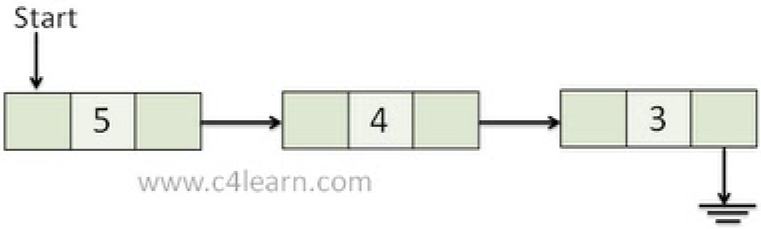
Algorithm: SRCHSL (INFO, LINK,START, ITEM, LOC)
LIST is sorted list in memory. This algorithm finds the location LOC of the node where ITEM first appear in LIST, or sets LOC=NULL.

```
1. Set PTR:=START.
2. Repeat Step 3 while PTR≠NULL:
3.        If ITEM < INFO[PTR], then:
               Set PTR:=LINK[PTR].         [PTR now points to the next node]
          Else If ITEM=INFO[PTR],then
              Set LOC:=PTR, and Exit.      [Search is successful.]
          Else:
              LOC:=NULL , and Exit.        [ITEM now exceeds INFO[PTR].]
          [End of If structure.]
   [End of Step 2 loop.]
4. Set LOC:=NULL.
5. Exit.
```

Assignment:
Write a program to create, display and search an element in a Single Link List.

Singly linked list node structure

Before doing Singly Linked Operations we need to define structure for Linked List. Structure is collection of different data types in a single unit.

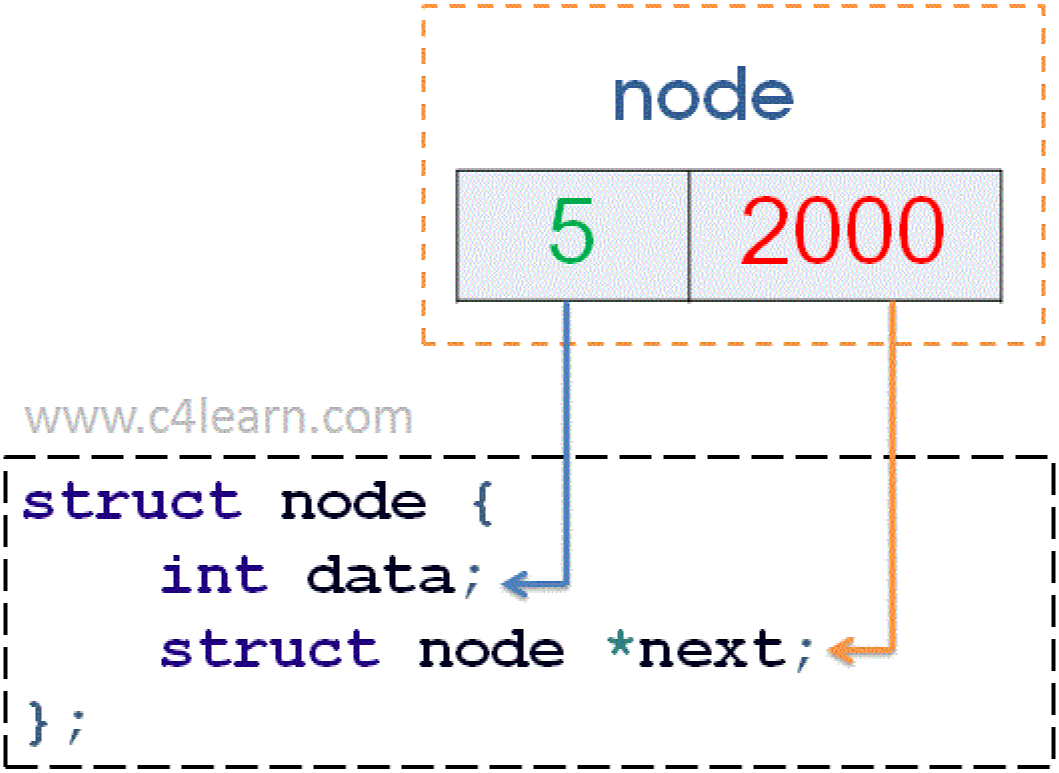


Node Structure for Singly Linked List :

```
struct node {
      int data;
      struct node *next;
}*start = NULL;
```

In the above node structure we have defined two fields in structure –

No	Field	Significance
1	data	It is Integer Part for Storing data inside Linked List Node
2	next	It is pointer field which stores the address of another structure (i.e node)



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

Explanation of Node Structure :

1. We have declared structure of type **“NODE”**, i.e we have created a Single Linked List Node.
2. A Node in general language is a Structure having two value containers i.e **[Square box having two Partitions]**
3. One value container stores **actual data and another stores address of the another structure** i.e (Square box having two partitions)
4. We have declared a structure and also created 1 very first structure called **“Start”.**
5. Very first node **“Start”** contain 1 field for storing data and another field for address of another structure
6. As this is very first node or Structure, we have specified its next field with **“NULL”** value.

NULL means “NOTHING” , if next node is unavailable then initialize variable name to NULL.

Step 1 : Include Alloc.h Header File

1. We don’t know, how many nodes user is going to create once he execute the program.
2. In this case we are going to allocate memory using Dynamic Memory Allocation functions such as Alloc & Malloc.
3. Dynamic memory allocation functions are included in alloc.h

```
#include<alloc.h>
```

Step 2 : Define Node Structure

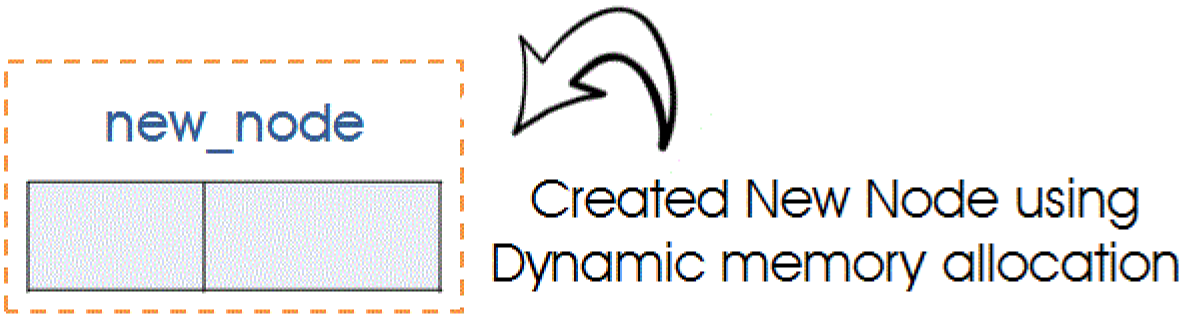
We are now defining the new global node which can be accessible through any of the function.

```
struct node {
  int data;
  struct node *next;
}*start=NULL;
```

Step 3 : Create Node using Dynamic Memory Allocation

Now we are creating one node dynamically using malloc function.We don’t have prior knowledge about number of nodes , so we are calling malloc function to create node at run time.

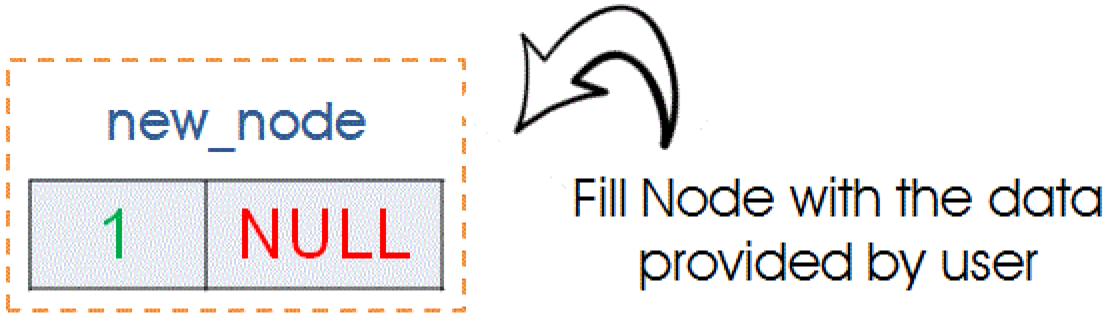
```
new_node=(struct node *)malloc(sizeof(struct node));
```



Step 5 : Fill Information in newly Created Node

Now we are accepting value from the user using scanf. Accepted Integer value is stored in the data field.
[box]**Tip #1** Whenever we create new node , Make its Next Field as NULL.[/box]

```
printf("nEnter the data : ");
scanf("%d",&new_node->data);
new_node->next=NULL;
```



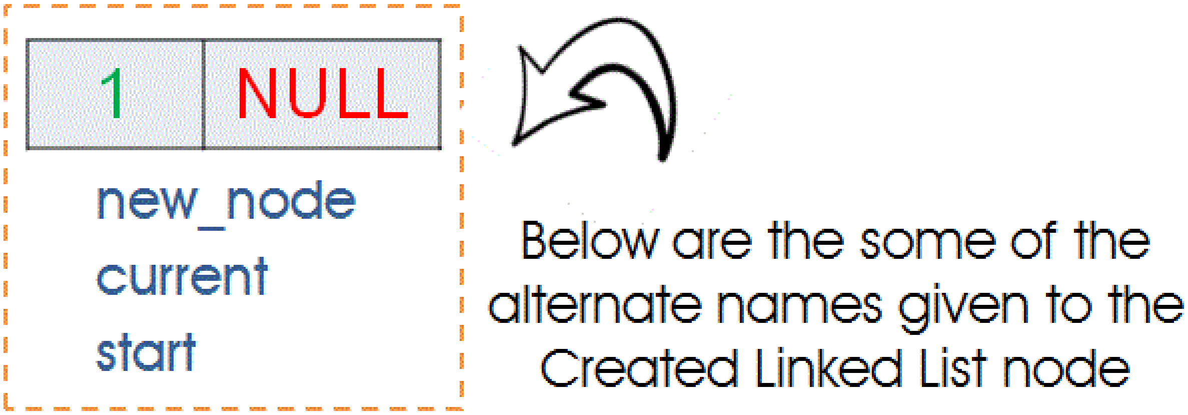
Step 6 : Creating Very First Node

If node created in the above step is very first node then we need to assign it as Starting node. If start is equal to null then we can identify node as first node –

```
start == NULL
```

First node has 3 names : new_node,current,start

```
if(start == NULL) {     {
   start = new_node;
   curr = new_node;
}
```


1
NULL
new_node
current
start
Below are the some of the alternate names given to the Created Linked List node

Step 7 : Creating Second or nth node

1. Lets assume we have 1 node already created i.e we have first node. First node can be referred as “new_node”,”curr”,”start”.
2. Now we have called create() function again

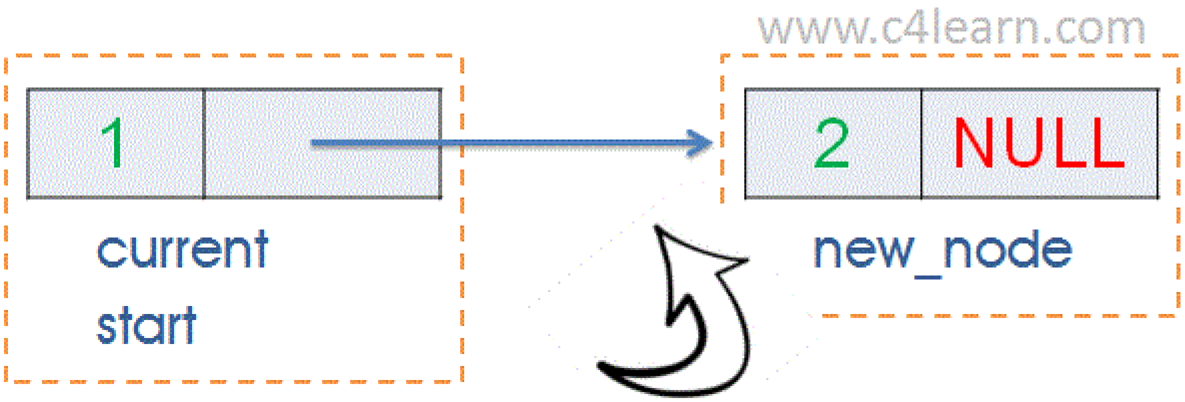
Now we already have starting node so control will be in the else block –

```
else
    {
    current->next = new_node;
    current = new_node;
    }
```

Inside Else following things will happen –

In the else block we are making link between new_node and current node.

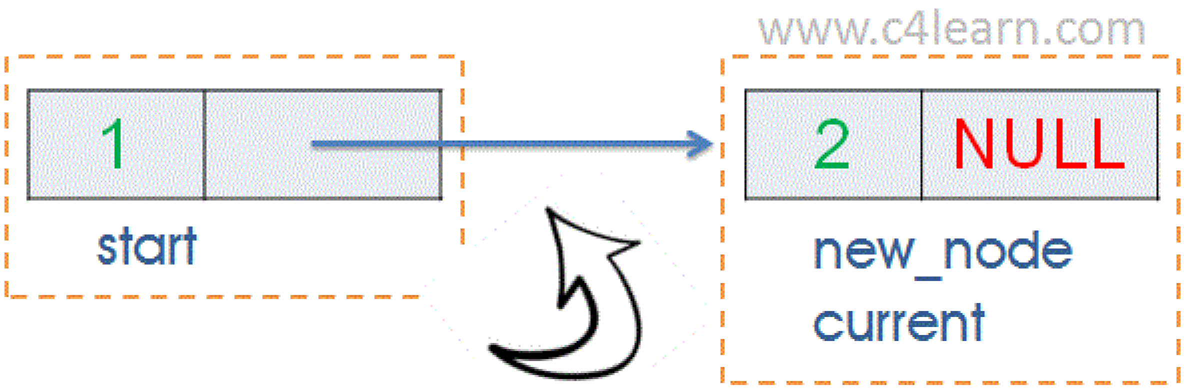
```
current->next = new_node;
```



Making Link between Current and new_node

Now move current pointer to next node –

```
current = new_node;
```



Now current pointer is pointing to new node i.e 2nd Node

Traversing through Singly Linked List (SLL) :

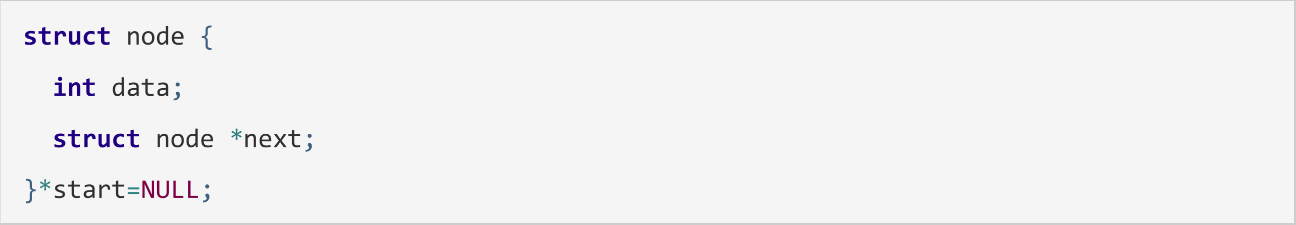
In the previous chapter we have learnt about the Creation of Singly Linked List. In this chapter, we will see how to traverse through Singly Linked List using C Programming.

1. Introduction :

Consider the Singly Linked list node structure. Traversing linked list means visiting each and every node of the Singly linked list. Following steps are involved while traversing the singly linked list –

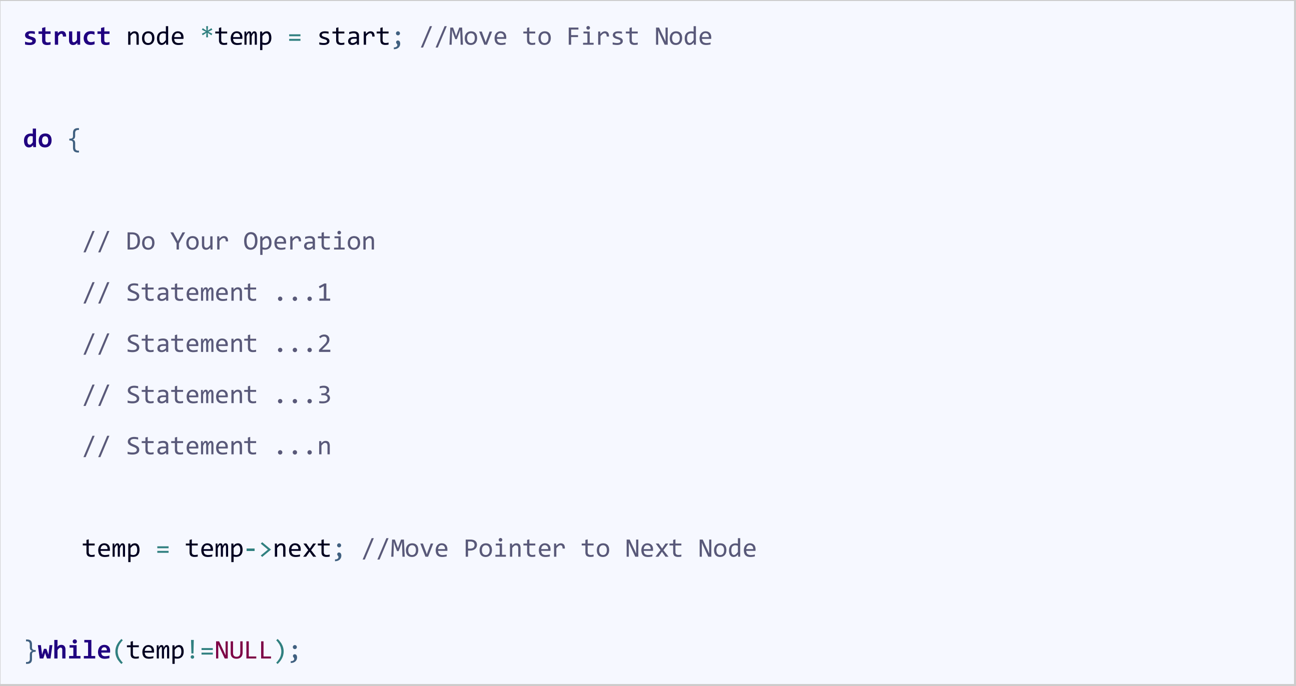
1. Firstly move to the first node
2. Fetch the data from the node and perform the operations such as arithmetic operation or any operation depending on data type.
3. After performing operation, advance pointer to next node and perform all above steps on Visited node.

2. Node Structure and Program Declaration :



```
struct node {
  int data;
  struct node *next;
}*start=NULL;
```

and the function definition for traversing linked list is –



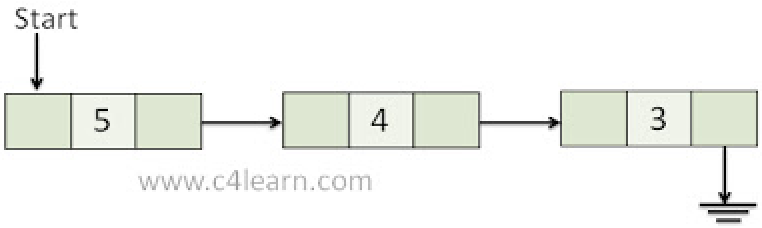
```
struct node *temp = start; //Move to First Node

do {

    // Do Your Operation
    // Statement ...1
    // Statement ...2
    // Statement ...3
    // Statement ...n

    temp = temp->next; //Move Pointer to Next Node

}while(temp!=NULL);
```

3. Explanation of Traversing Linked List :

We know that, In order to traverse linked list, We need to visit first node and then visiting nodes one by one until the last node is reached.

```
temp = start; //Move to First Node
do {
    // Do Your Operation
    // Statement ...1
    // Statement ...2
    // Statement ...3
    // Statement ...n
    temp = temp->next; //Move Pointer to Next Node

}while(temp!=NULL);
```

Explanation :



Initially

```
temp = start;
```

1. Store Address of Starting node into “**temp**” , Print data stored in the node “**temp**“.

   ## Refer : Different Syntax and Terms in Linked List

2. Once Data stored in the temp is printed, move pointer to the next location so that “temp” will contain address of 2nd node.

[box]

Special Note :

In Singly Linked List Program , do not change start index un-necessarily because we must have something that can store address of **Head node**

Display Singly Linked List from First to Last

In the last chapter we have learn't about traversing the linked list. In this chapter we will be printing the content of the linked list from start to last.

Steps for Displaying the Linked List :

1. In order to write the program for display, We must create a linked list using create(). Then and then only we can traverse through the linked list.
2. Traversal Starts from **Very First node**. We cannot modify the address stored inside global variable "**start**" thus we have to declare one temporary variable - "**temp**" of type node.
3. In order to traverse from start to end you should **assign Address of Starting node** in **Pointer variable** i.e temp

```
struct node *temp;   //Declare temp
temp = start;        //Assign Starting Address to temp
```

Now we are checking the value of pointer (i.e temp). If the temp is NULL then we can say that last node is reached.

```
while(temp!=NULL)
    {
    printf("%d",temp->data);
    temp=temp->next;
    }
```

See below dry run to understand the complete code and consider the linked list shown in the above figure –

Control Position	Printed Value	temp points to
Before Going into Loop	-	Starting Node
Iteration 1	1	Node with data = 3
Iteration 2	3	Node with data = 5
Iteration 3	5	Node with data = 7
Iteration 4	7	NULL
Iteration 5(False Condition)	-	-

Some Terminology Of Linked List :

Consider the following node structure –

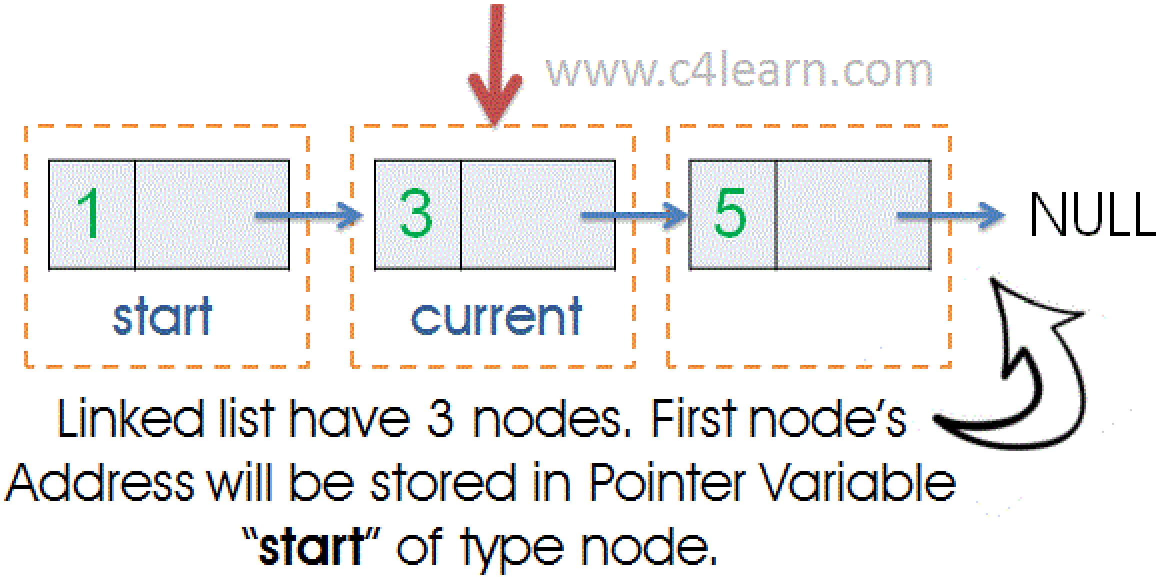
```
struct node {
      int data;
      struct node *next;
}*start = NULL;
struct node *new_node,*current;
```

Summary of Different Linked List Terms :

Declaration Term	Explanation
struct node *new_node,*current;	Declaring Variable of Type Node.

`*start = NULL;`	Declared a global variable of type node. “start” is used to refer the starting node of the linked list.
`start->next`	Access the 2nd Node of the linked list. This term contain the address of 2nd node in the linked list. If 2nd node is not present then it will return NULL.
`start->data`	It will access “data” field of starting node.
`start->next->data`	It will access the “**data**” field of **2nd node**.
`current = start->next`	Variable “**current**” will contain the address of 2nd node of the linked list.
`start = start->next;`	After the execution of this statement, 2nd node of the linked list will be called as “**starting node**” of the linked list.

Explanation of Terms :


www.c4learn.com
1
3
5
NULL
start
current
Linked list have 3 nodes. First node’s
Address will be stored in Pointer Variable
“start” of type node.

Consider the above linked list , we have initial conditions –

1. “start” node is pointing to First Node [data = 1]

2. "current" node is pointing to Second Node [data = 3]

with respect to the above linked list , data will be –

Term	**Explanation**
`current = start->next`	"**current**" will point to node having data = 3
`int num = start->data`	"**num**" will contain integer value "**1**".
`int num = start->next->data`	"**num**" will contain integer value "**3**".
`temp = current->next`	"**temp**" will point to node having data = 5
`int num = current->data`	"**num**" will contain integer value "**3**".
`int num = current->next->data`	"**num**" will contain integer value "**5**".

Complete code: create, display and search

Global Declaration (structure and header files)

```
int search(int);

#include<alloc.h>

struct node

{

  int data;

  struct node *next;

}*start=NULL;
```

Create Singly Linked List

```
void createlinklist()
{
char ch;
 do
 {
  struct node *new_node,*current;

  new_node=(struct node *)malloc(sizeof(struct node));

  printf("/nEnter the data : ");
  scanf("%d",&new_node->data);
  new_node->next=NULL;

  if(start==NULL)
  {
  start=new_node;
  current=new_node;
  }
  else
  {
  current->next=new_node;
  current=new_node;
  }

 printf("Do you want to create another : ");
 ch=getche();
 }while(ch!='n');
}
```

Display Singly Linked List from First to Last

```
void display()
{
struct node *new_node;
 printf("The Linked List : n");
 new_node=start;
 while(new_node!=NULL)
   {
   printf("%d--->",new_node->data);
   new_node=new_node->next;
   }
  printf("NULL");
}
```

Main Method to call all functions here.

```
void main()
{
 int num,item;
clrscr();
 createlinklist();
```

```
  display();
 printf("Enter element you want to search : ");
 scanf("%d",&item);
 search(item)==1?printf("\n Present"):printf("\n Not present");
getch();
}
```

Searching particular element from link list.

This Function only tells that whether data is present or not

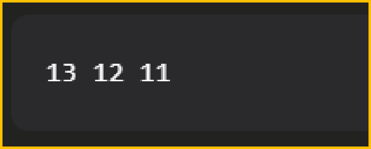
```
int search(int num)
{
int flag = 0;
struct node *temp;

temp = start;
  while(temp!=NULL)
  {
    if(temp->data == num)
       return(1); //Found

    temp = temp->next;
  }

if(flag == 0)
    return(0); // Not found
}
```

LAB 7

Create & traversing linked list:

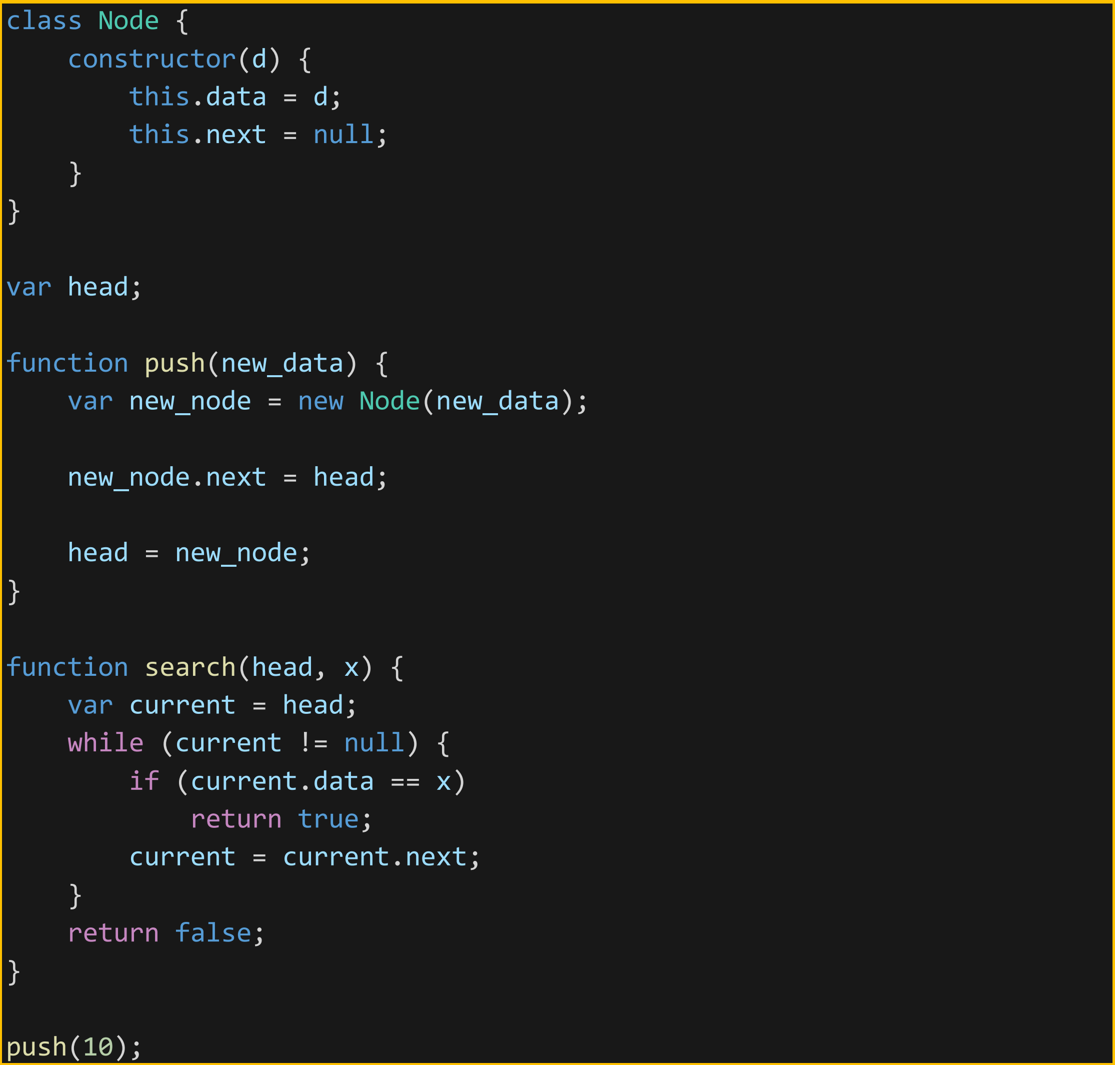
```
class LinkedList {
    Node head;


    static class Node {
        int data;
        Node next;
        Node(int d) {
            this.data = d;
            next = null;
        }

        public void printList() {
            Node n = head;
            while (n != null) {
                System.out.print(n.data + " ");
                n = n.next;
            }
        }

        public static void main(String[] args) {
            LinkedList llist = new LinkedList();

            llist.head = new Node(13);
            Node second = new Node(12);
            Node third = new Node(11);

            llist.head.next = second;
            second.next = third;

            llist.printList();
        }
    }
```

Output:



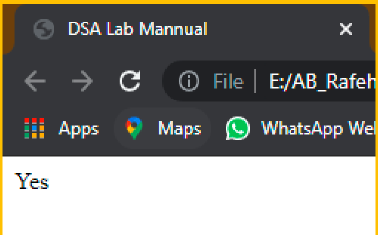
```
13 12 11
```

Searching & Display linked list:

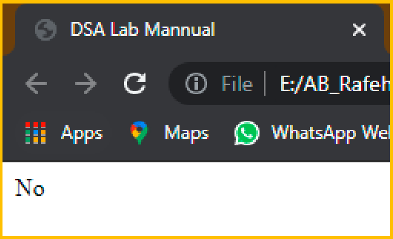


```
class Node {
    constructor(d) {
        this.data = d;
        this.next = null;
    }
}

var head;

function push(new_data) {
    var new_node = new Node(new_data);

    new_node.next = head;

    head = new_node;
}

function search(head, x) {
    var current = head;
    while (current != null) {
        if (current.data == x)
            return true;
        current = current.next;
    }
    return false;
}

push(10);
push(30);
push(11);
push(21);
push(14);

if (search(head, 21)) //if put 24 so you will get “No”
    document.write("Yes");
else
    document.write("No");
```

Output:



```
Yes
```



```
No
```

LAB 8

Insert node at First Position : Singly Linked List

Insert node at Start/First Position in Singly Linked List

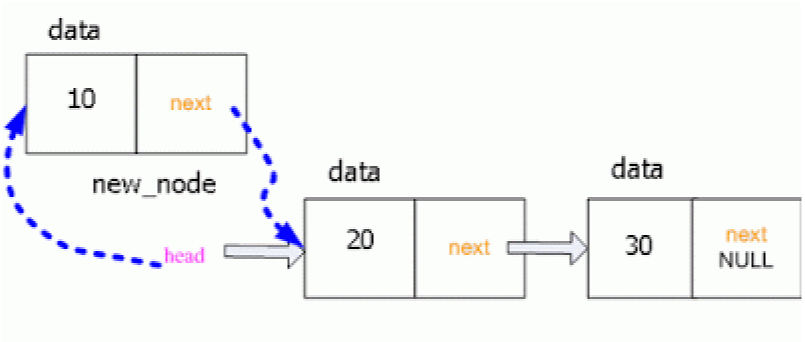
Inserting node at start in the SLL (Steps):

1. **Create** New Node
2. Fill Data into "**Data Field**"
3. Make it's "**Pointer**" or "**Next Field**" as **NULL**
4. Attach This newly **Created node to Start**
5. Make newnode as **Starting node**

```
void insert_at_beg()
{
struct node *new_node,*current;

new_node=(struct node *)malloc(sizeof(struct node));

 if(new_node == NULL)
    printf("nFailed to Allocate Memory");

 printf("nEnter the data : ");
 scanf("%d",&new_node->data);
 new_node->next=NULL;

   if(start==NULL)
   {
   start=new_node;
   current=new_node;
   }
   else
```

```
    {
    new_node->next=start;
    start=new_node;
    }
}
```

Diagram :



Attention :

1. If starting node is not available then **"Start = NULL"** then following part is executed

```
if(start==NULL)
        {
        start=new_node;
        current=new_node;
        }
```

2. If we have previously created First or starting node then **"else part"** will be executed to insert node at start

```
else
        {
        new_node->next=start;
        start=new_node;
        }
```

Insert node at Last Position : Singly Linked List

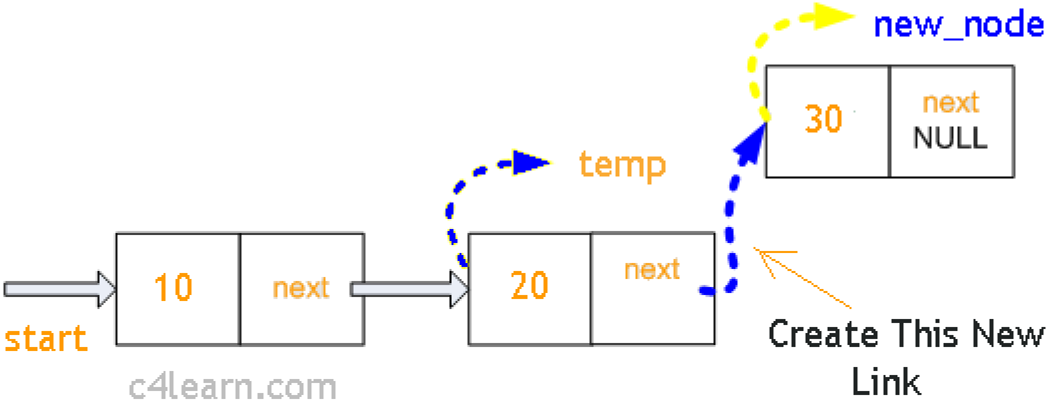
Insert node at Last / End Position in Singly Linked List

Inserting node at start in the SLL (Steps):

1. **Create** New Node
2. Fill Data into “**Data Field**“
3. Make it’s “**Pointer**” or “**Next Field**” as **NULL**
4. Node is to be inserted at Last Position so we need to traverse **SLL upto Last Node**.
5. Make link between **last node and newnode**

```
void insert_at_end()
{
struct node *new_node,*current;

new_node=(struct node *)malloc(sizeof(struct node));

if(new_node == NULL)
   printf("nFailed to Allocate Memory");

 printf("nEnter the data : ");
 scanf("%d",&new_node->data);
 new_node->next=NULL;

 if(start==NULL)
 {
   start=new_node;
   current=new_node;
 }
 else
 {
   temp = start;
     while(temp->next!=NULL)
     {
     temp = temp->next;
     }
   temp->next = new_node;
 }
}
```

Diagram :



Attention :

1. If starting node is not available then **“Start = NULL”** then following part is executed

```
if(start==NULL)
       {
       start=new_node;
       current=new_node;
       }
```

2. If we have previously created First or starting node then **“else part”** will be executed to insert node at start
3. Traverse Upto Last Node., So that **temp** can keep track of Last node

```
else
       {
       temp = start;
   while(temp->next!=NULL)
  {
  temp = temp->next;
  }
```

4. Make **Link between Newly Created node and Last node** (temp)

```
temp->next = new_node;
```

To pass Node Variable to Function Write it as –

```
void insert_at_end(struct node *temp)
```

Insert node at middle position : Singly Linked List

Linked-List : Insert Node at Middle Position in Singly Linked List

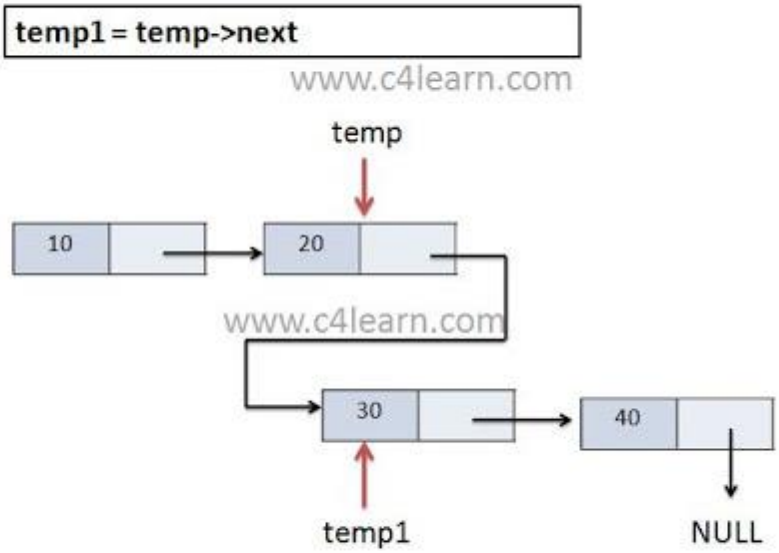
```
void insert_mid()
{
    int pos,i;
    struct node *new_node,*current,*temp,*temp1;

    new_node=(struct node *)malloc(sizeof(struct node));

    printf("nEnter the data : ");
    scanf("%d",&new_node->data);

    new_node->next=NULL;
    st :
    printf("nEnter the position : ");
    scanf("%d",&pos);

    if(pos>=(length()+1))
       {
       printf("nError : pos > length ");
       goto st;
       }

    if(start==NULL)
       {
       start=new_node;
       current=new_node;
       }
    else
       {
       temp = start;
             for(i=1;i< pos-1;i++)
             {
             temp = temp->next;
             }
       temp1=temp->next;
       temp->next = new_node;
       new_node->next=temp1;
       }
```

```
}
```

Explanation :

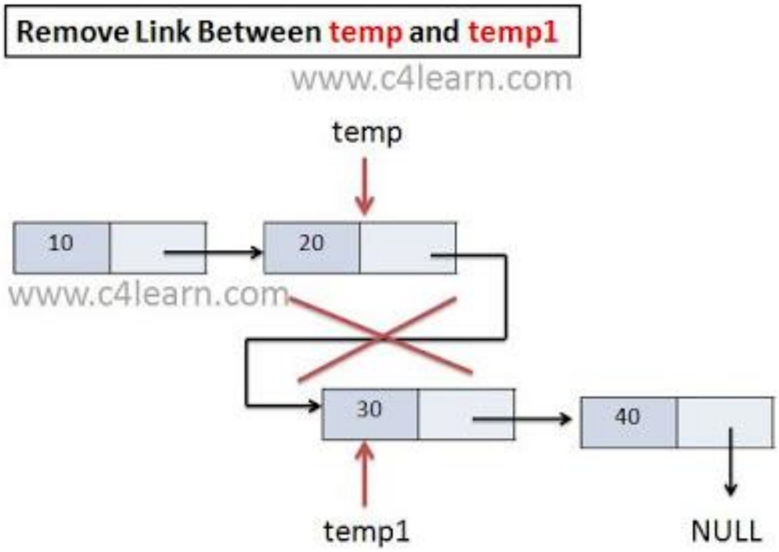
Step 1 : Get Current Position Of “temp” and “temp1” Pointer.

```
temp = start;
                for(i=1;i< pos-1;i++)
  {
  temp = temp->next;
  }
```



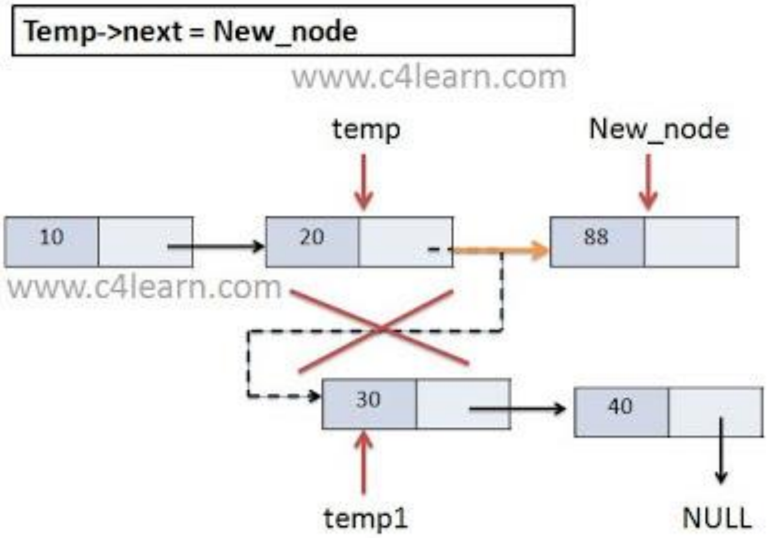
Step 2 :

```
temp1=temp->next;
```



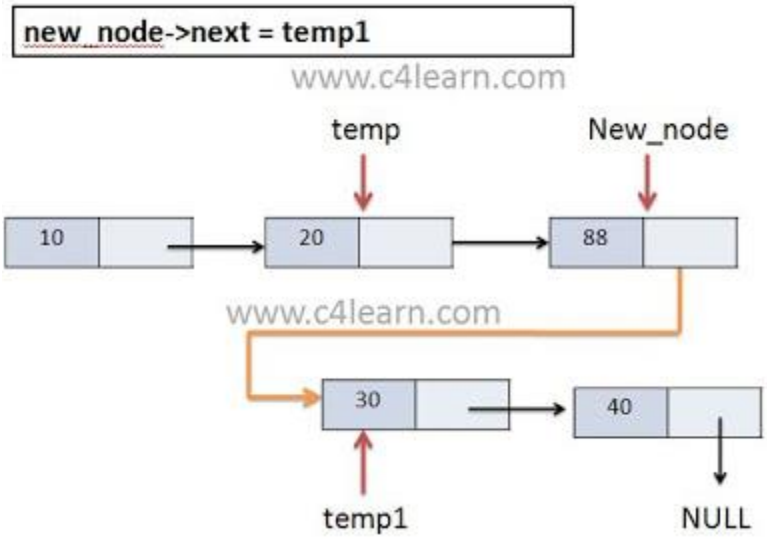
Step 3 :

```
temp->next = new_node;
```



Step 4 :

```
new_node->next = temp1
```



Delete Node from First Position : Singly Linked List

Delete a node in a singly linked list just like a person breaks away from a human chain then the two persons (between whom the person was) needs to join hand together to continue the chain. We use free keyword to delete a node from linked list, first finding the node by searching in the linked list and then calling free() on the pointer that containing its address.

How to delete a node from linked list:

If you wish to delete head (first) node from linked list, you need to assign 'head' node to a 'temp' node and changeling the head node with next node then calling free(temp) to delete the node. If you wish to delete tail (last) node from linked list, you need to assign 'tail' node to a 'temp' node and changing the tail node with previous node then calling free(temp) to delete the node. If you wish to delete node is any node other than the head and tail node then you need to changing the 'next' pointer of the previous node with the address of the node that is just after the deleted node.

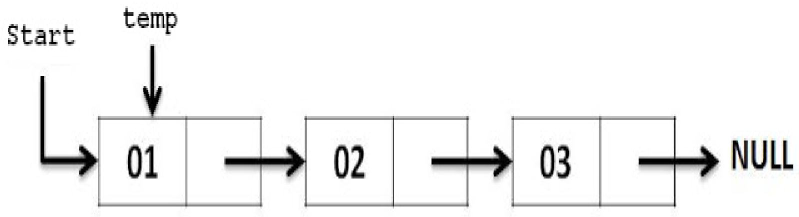
Delete First Node from Singly Linked List Program :

```
void del_beg()
{
struct node *temp;

temp = start;
start = start->next;

free(temp);
printf("nThe Element deleted Successfully ");
}
```

Attention :

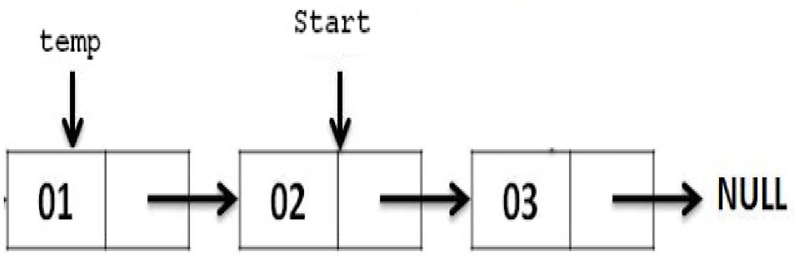
Step 1 : Store Current Start in Another Temporary Pointer

```
temp = start;
```



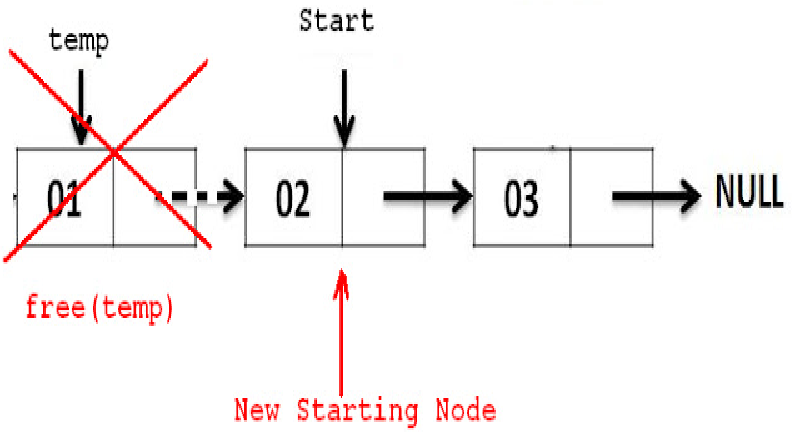
Step 2 : Move Start Pointer **One position Ahead**

```
start = start->next;
```



Step 3 : Delete temp i.e Previous Starting Node as we have Updated Version of Start Pointer

```
free(temp);
```



Delete a node in a singly linked list just like a person breaks away from a human chain then the two persons (between whom the person was) needs to join hand together to continue the chain. We use free keyword to delete a node from linked list, first finding the node by searching in the linked list and then calling free() on the pointer that containing its address.

How to delete a node from linked list:

If you wish to delete head (first) node from linked list, you need to assign 'head' node to a 'temp' node and changing the head node with next node then calling free(temp) to delete the node. If you wish to delete tail (last) node from linked list, you need to assign 'tail' node to a 'temp' node and changing the tail node with previous node then calling free(temp) to delete the node. If you wish to delete node is any node other than the head and tail node then you need to change the 'next' pointer of the previous node with the address of the node that is just after the deleted node.

```
//Code for create, search, insert and display in singly link list
#include<stdio.h>
#include<stdlib.h>
#define Esc 27
struct node
```

```
{
    int data;
    struct node *next;
}*head=NULL,*last;
void create_node();
int search();
void delete_node();
void display();
//-----------------------------------------------
int main()
{
    struct node *temp;
    int i;
    char ch;
    while(1)
    {
        printf("\n\n          <@> Main Menu <@>\n");
        printf("          ******************\n");
        printf("\n\t1.Create Node\n\t2.Seacrh Node\n\t3.Delete node\n\t4.Display.\n\tPress ESC to 
Exit\n");
        printf("\n Please choose, what kind of work you want to do?");
        ch=getch();
        switch(ch)
        {
        case '1':
            create_node();
            break;
        case '2':
            search();
            break;
        case '3':
            delete_node();
            break;
        case '4':
            display();
            break;
        case Esc:
            exit(1);
        default:
            printf("\n\n Invalid Input/ choice.");
        }
    }
    return 0;
}
//-----------------------------------------------
void create_node()
{
    struct node *new_node;
    char ch;
    do
    {
        new_node=(struct node *)malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d",&new_node->data);
        new_node->next=NULL;
        if(head==NULL)
        {
            head=new_node;
            last=new_node;
        }
        else
        {
            last->next=new_node;
            last=new_node;
        }
        printf("\n To create again press any key, if not press Space.\n");
        ch=getch();
    }
    while(ch!=32);
}
//-----------------------------------------
int search()
{
    int svalue,i=0;
    struct node *temp;
    printf("\n Search: ");
    scanf("%d",&svalue);
    temp=head;
    while(temp!=NULL)
    {
        i++;
        if(svalue==temp->data)
        {
            printf("\n Searched value %d found in the location of %d. ",svalue,i);
            return i;
        }
        temp=temp->next;
    }
    printf("\n Not found %d in Linked List.",svalue);
    return 0;
```

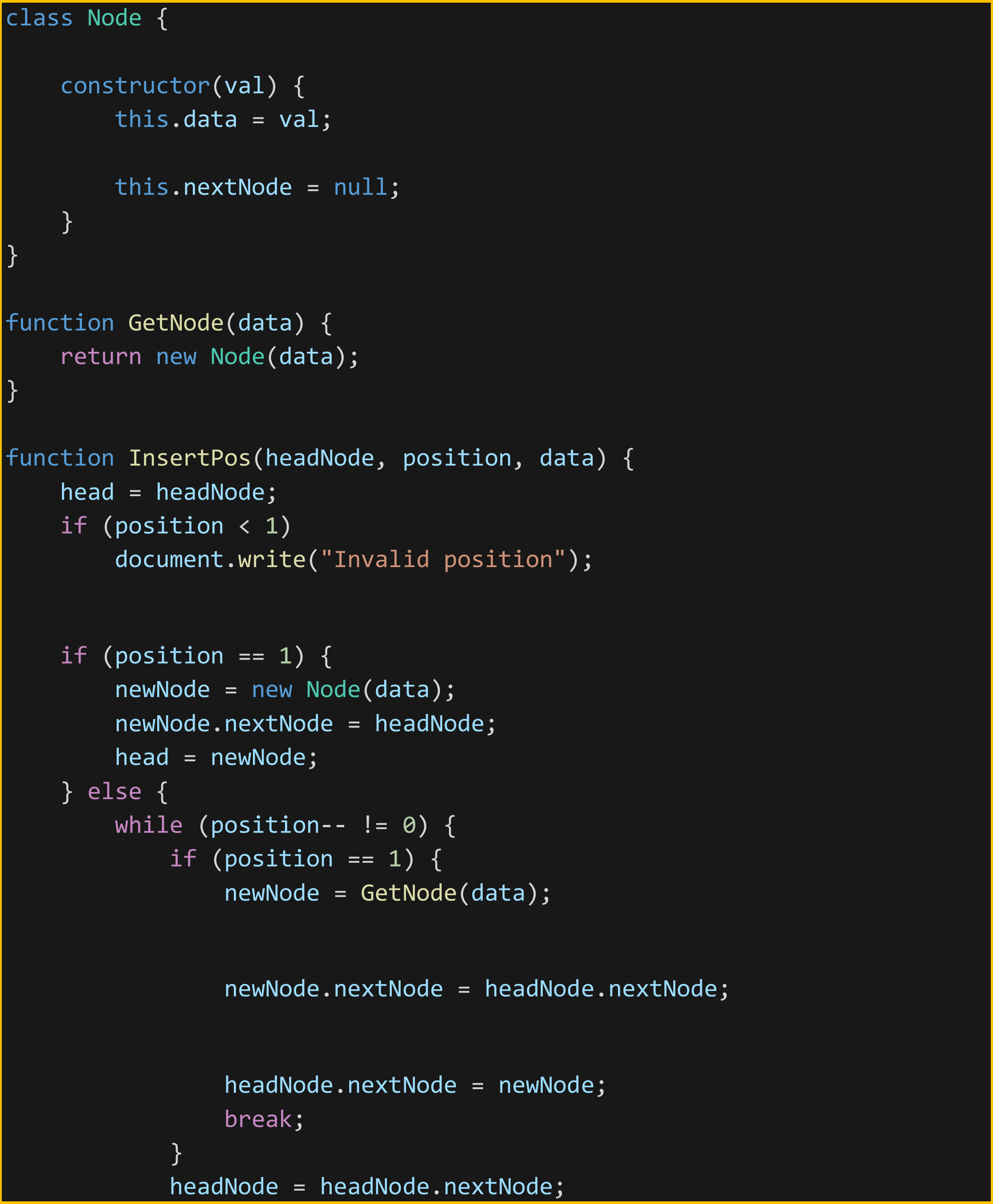
```
}
//-------------------------------------------
void delete_node()
{
    struct node *temp=head,*prev;
    int i,position=search();
    if(position==0)
        printf("\n ERROR!!!(Node not found)");
    else if(position==1)
    {
        head=temp->next;
        free(temp);
        printf("\n Head node successfully deleted.");
    }
    else
    {
        for(i=1; i<=position; i++)
        {
            if(i==position)
            {
                if(temp==last)
                {
                    prev->next=NULL;
                    last=prev;
                    free(temp);
                    printf("\n Tail Node succesfully deleted.");
                }
                else
                {
                    prev->next=temp->next;
                    free(temp);
                    printf("\n Node succesfully deleted.");
                }
            }
            prev=temp;
            temp=temp->next;
        }
    }
}
//--------------------------------------------
void display()
{
    struct node *temp;
    if(head==NULL)
        printf("\n\n Linked List is empty.");
    else
    {
        printf("\n\t Linked List:\n\t-------------\n\n");
        temp=head;
        while(temp!=NULL)
        {
            printf("\t=>\t%d\n",temp->data);
            temp=temp->next;
        }
    }
}
```

```
// A complete working C program to delete a node in a linked list
// at a given position
#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct node
{
    int data;
    struct node *next;
};

/* Given a reference (pointer to pointer) to the head of a list
   and an int, inserts a new node on the front of the list. */
void push(struct node** head_ref, int new_data)
{
    struct node* new_node = (struct node*) malloc(sizeof(struct node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}

/* Given a reference (pointer to pointer) to the head of a list
   and a position, deletes the node at the given position */
void deleteNode(struct node **head_ref, int position)
{
   // If linked list is empty
   if (*head_ref == NULL)
      return;

   // Store head node
   struct node* temp = *head_ref;

    // If head needs to be removed
    if (position == 0)
    {
        *head_ref = temp->next;   // Change head
        free(temp);               // free old head
        return;
    }

    // Find previous node of the node to be deleted
    for (int i=0; temp!=NULL && i<position-1; i++)
         temp = temp->next;

    // If position is more than number of ndoes
    if (temp == NULL || temp->next == NULL)
         return;

    // Node temp->next is the node to be deleted
    // Store pointer to the next of node to be deleted
    struct node *next = temp->next->next;

    // Unlink the node from linked list
    free(temp->next);  // Free memory

    temp->next = next;  // Unlink the deleted node from list
}

// This function prints contents of linked list starting from
// the given node
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf(" %d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    push(&head, 7);
    push(&head, 1);
    push(&head, 3);
    push(&head, 2);
    push(&head, 8);

    puts("Created Linked List: ");
    printList(head);
    deleteNode(&head, 4);
    puts("\nLinked List after Deletion at position 4: ");
    printList(head);
    return 0;
}
```

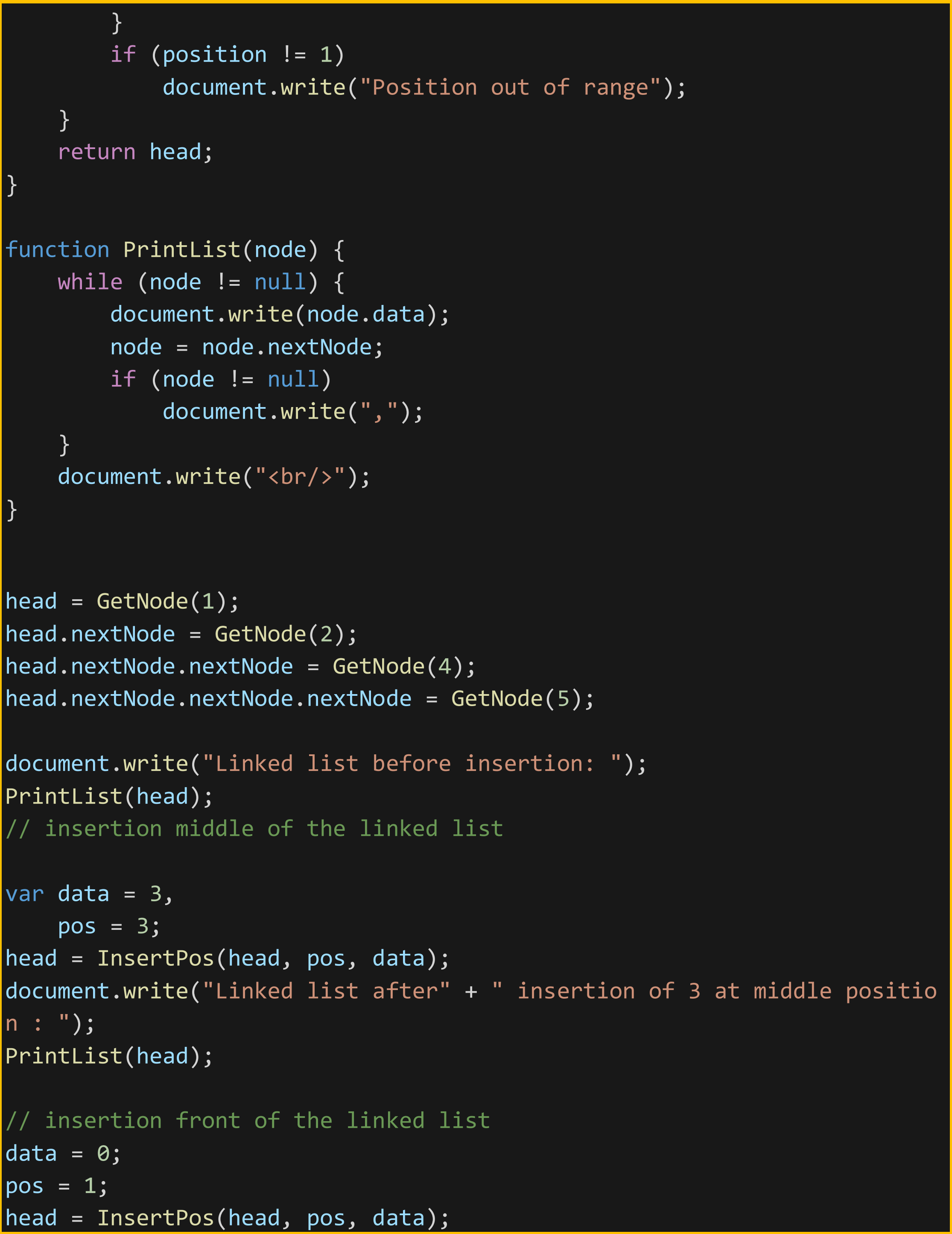
Output:

```
Created Linked List:
 8  2  3  1  7
Linked List after Deletion at position 4:
 8  2  3  1
```

LAB 8



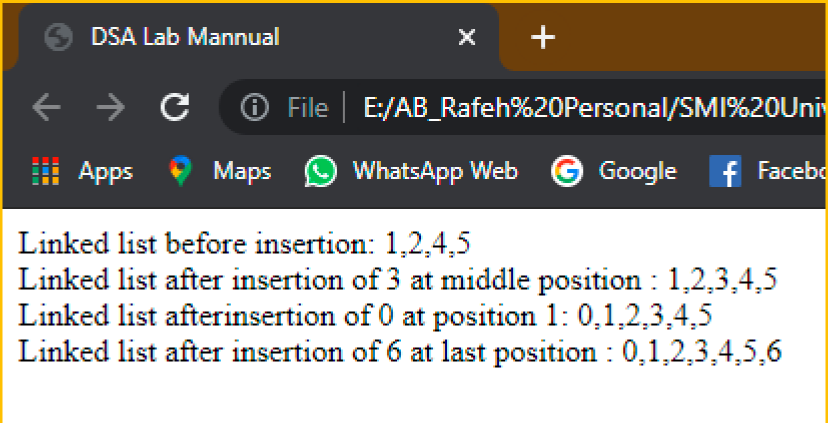
```
class Node {

    constructor(val) {
        this.data = val;

        this.nextNode = null;
    }
}

function GetNode(data) {
    return new Node(data);
}

function InsertPos(headNode, position, data) {
    head = headNode;
    if (position < 1)
        document.write("Invalid position");


    if (position == 1) {
        newNode = new Node(data);
        newNode.nextNode = headNode;
        head = newNode;
    } else {
        while (position-- != 0) {
            if (position == 1) {
                newNode = GetNode(data);


                newNode.nextNode = headNode.nextNode;


                headNode.nextNode = newNode;
                break;
            }
            headNode = headNode.nextNode;
```



```
        }
        if (position != 1)
            document.write("Position out of range");
    }
    return head;
}

function PrintList(node) {
    while (node != null) {
        document.write(node.data);
        node = node.nextNode;
        if (node != null)
            document.write(",");
    }
    document.write("<br/>");
}


head = GetNode(1);
head.nextNode = GetNode(2);
head.nextNode.nextNode = GetNode(4);
head.nextNode.nextNode.nextNode = GetNode(5);

document.write("Linked list before insertion: ");
PrintList(head);
// insertion middle of the linked list

var data = 3,
    pos = 3;
head = InsertPos(head, pos, data);
document.write("Linked list after" + " insertion of 3 at middle positio
n : ");
PrintList(head);

// insertion front of the linked list
data = 0;
pos = 1;
head = InsertPos(head, pos, data);
```

```
document.write("Linked list after" + "insertion of 0 at position 1: ");
PrintList(head);

// insertion at end of the linked list
data = 6;
pos = 7;
head = InsertPos(head, pos, data);
document.write("Linked list after" + " insertion of 6 at last position
: ");
PrintList(head);
```

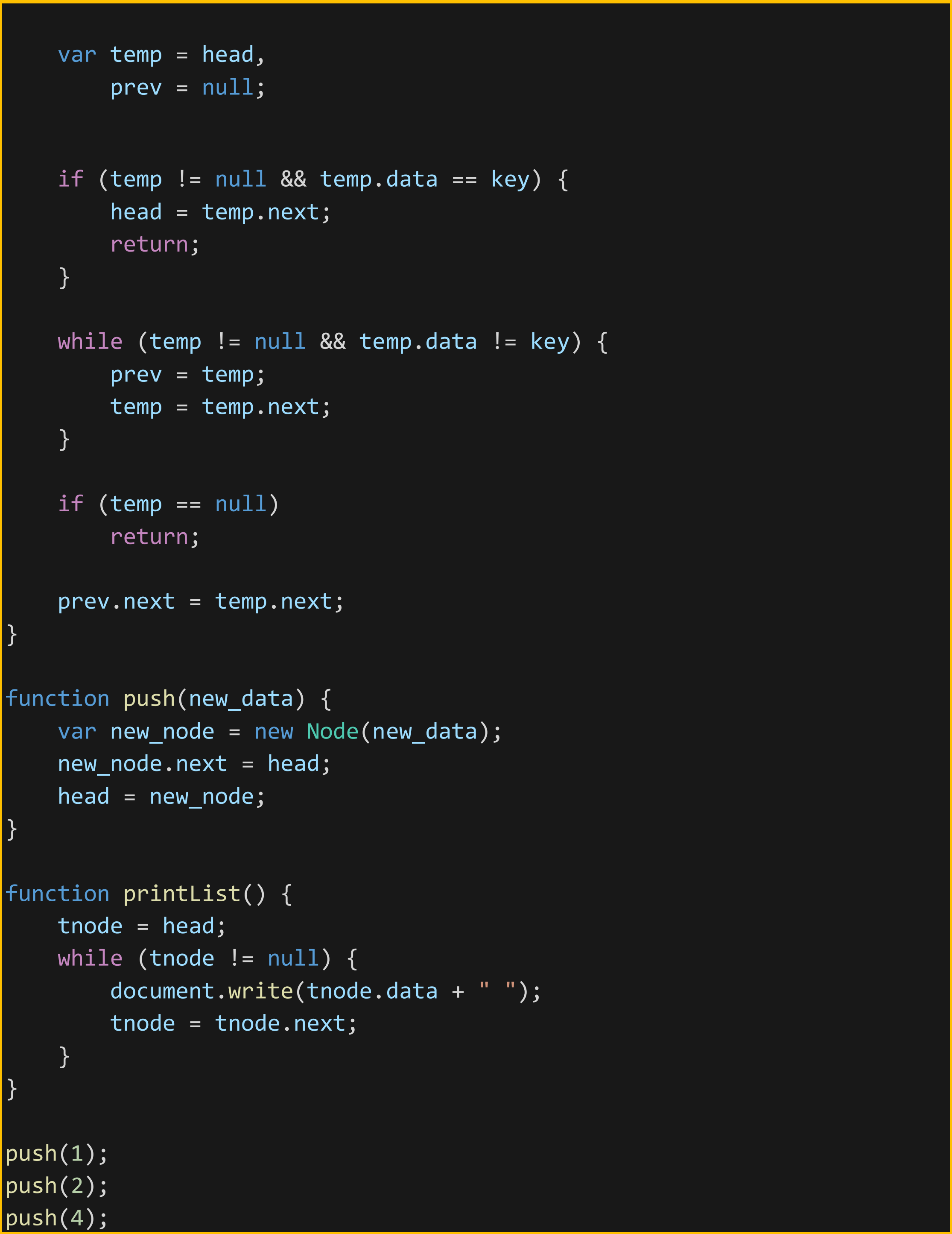
Output:



```
Linked list before insertion: 1,2,4,5
Linked list after insertion of 3 at middle position : 1,2,3,4,5
Linked list afterinsertion of 0 at position 1: 0,1,2,3,4,5
Linked list after insertion of 6 at last position : 0,1,2,3,4,5,6
```

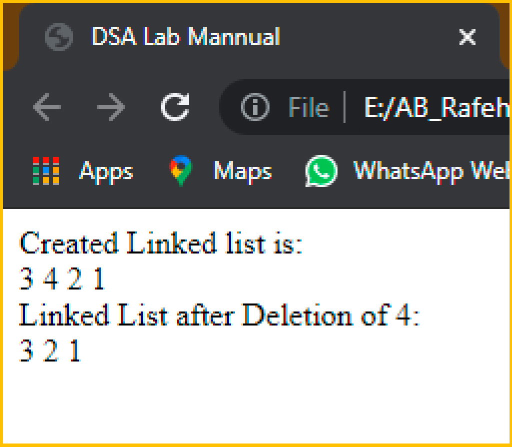
Deletion node:

```
var head;

class Node {
    constructor(val) {
        this.data = val;
        this.next = null;
    }
}


function deleteNode(key) {
```



```
    var temp = head,
        prev = null;


    if (temp != null && temp.data == key) {
        head = temp.next;
        return;
    }

    while (temp != null && temp.data != key) {
        prev = temp;
        temp = temp.next;
    }

    if (temp == null)
        return;

    prev.next = temp.next;
}

function push(new_data) {
    var new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
}

function printList() {
    tnode = head;
    while (tnode != null) {
        document.write(tnode.data + " ");
        tnode = tnode.next;
    }
}

push(1);
push(2);
push(4);
push(3);

document.write("Created Linked list is:<br/>");
printList();

deleteNode(4);

document.write("<br/>Linked List after Deletion of 4:<br/>");
printList();
```

Output:



```
Created Linked list is:
3 4 2 1
Linked List after Deletion of 4:
3 2 1
```

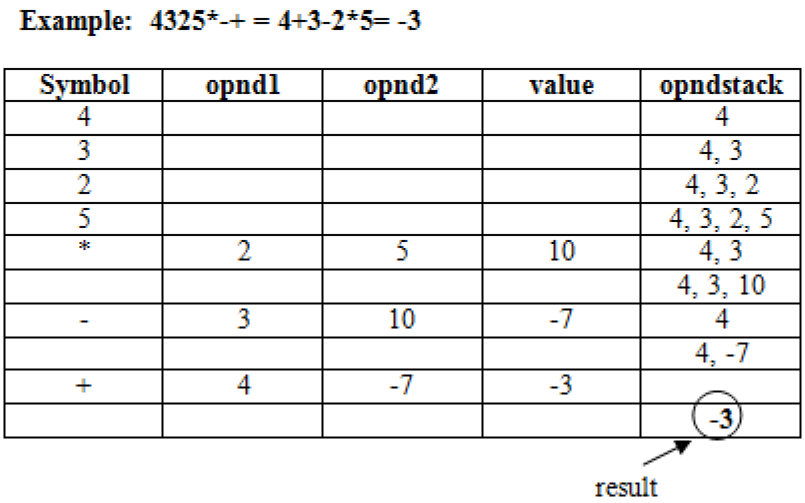
LAB 9

Objective

Evaluate Postfix Expression

Theory

Postfix notation is a notation for writing arithmetic expressions in which the operands appear after their operators. There are no precedence rules to learn, and parentheses are never needed. Because of this simplicity, some popular hand-held calculators use postfix notation to avoid the complications of the multiple parentheses required in non-trivial infix expressions. You are to write a computer program that simulates how these postfix calculators evaluate expressions.



Example: 4325*-+ = 4+3-2*5= -3

Symbol	opnd1	opnd2	value	opndstack
4				4
3				4, 3
2				4, 3, 2
5				4, 3, 2, 5
*	2	5	10	4, 3
				4, 3, 10
-	3	10	-7	4
				4, -7
+	4	-7	-3	
				-3

result

Algorithm of Evaluation of Postfix Expression

```
P→ postfix expression
1. Add a right parenthesis ")" at the end of P. [This act as sentinel.]
2. Scan P from left to right and repeat steps 3 and 4 for each
   element of P until the sentinel ")" is encountered,
3.        If an operand is encountered, put it on stack
4.        If an operator is encountered, then:
                  (a) Remove the top two elements of stack, where A is the top
                      element and  B is  the next-to-top  element.
                  (b) Evaluate B [operator] A.
                  (c) Place the result of (b) on stack.
          [End of If structure]
     [End of Step 2 loop.]
5. Set value equal to the top element on stack.
6. Exit.
```

Data Structure program to evaluate a postfix expression by using stack.

```
#define SIZE 50            /* Size of Stack */
#include <ctype.h>
int s[SIZE];
int top=-1;       /* Global declarations */

push(int elem)
{                       /* Function for PUSH operation */
 s[++top]=elem;
}

int pop()
{                      /* Function for POP operation */
 return(s[top--]);
}
void main()
{                         /* Main Program */
 char pofx[50],ch;
 int i=0,op1,op2;
 printf("\n\nRead the Postfix Expression ? ");
 scanf("%s",pofx);
 while( (ch=pofx[i++]) != '\0')
 {
  if(isdigit(ch)) push(ch-'0'); /* Push the operand */
  else
  {        /* Operator,pop two  operands */
   op2=pop();
   op1=pop();
   switch(ch)
   {
   case '+':push(op1+op2);break;
   case '-':push(op1-op2);break;
   case '*':push(op1*op2);break;
   case '/':push(op1/op2);break;
   }
  }
 }
 printf("\n Given Postfix Expn: %s\n",pofx);
 printf("\n Result after Evaluation: %d\n",s[top]);
getch();
}
```

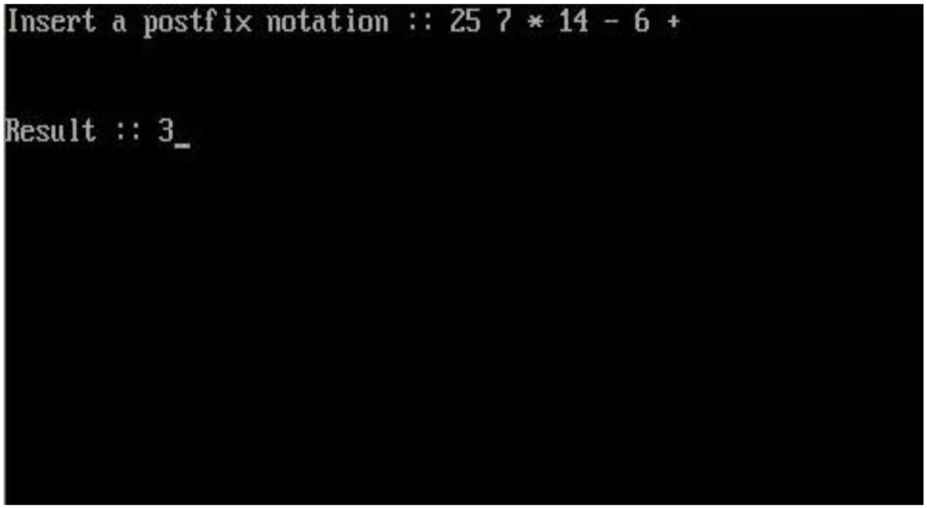
Postfix expression: 45+63/*23^+

Program:

```
#include<stdio.h>
#include<conio.h> float
stack[10]; int top=-1;
void push(char);
float pop();
float eval(char [],float[]);
void main()
{
int i=0;
char suffix[20];
float value[20],result;
clrscr();
printf("Enter a valid postfix expression\t");
gets(suffix);
while (suffix[i]!='\0')
{
if(isalpha(suffix[i]))
{
fflush(stdin);
printf("\nEnter the value of %c",suffix[i]);
scanf("%f",&value[i]);
}
i++;
}
result=eval(suffix,value);
printf("The result of %s=%f",suffix,result);
getch();
}
float eval(char suffix[],float data[])
{
int i=0;
float op1,op2,res; char ch;
while(suffix[i]!='\0')
{ ch=suffix[i];
if(isalpha(suffix[i]))
{
push(data[i]);
}
else
{ op2=pop();
op1=pop();
switch(ch)
{
case '+' : push(op1+op2);
break;
case '-' : push(op1-op2);
break;
case '*' : push(op1+op2);
break;
case '/' :push(op1/op2);
break;
case '^' : push(pow(op1,op2));
break;
}
}
i++;
} res=pop();
return(res);
}
void push(char num)
{ top=top+1;
stack[top]=num;
}
float pop()
{
float num;
num=stack[top];
top=top-1; return(num);
}
```

Alternate code

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 50
int stack[MAX];
char post[MAX];
int top=-1;
void pushstack(int tmp);
void calculator(char c);
void main()
{
 int i;
 clrscr();
 printf("Insert a postfix notation :: ");
 gets(post);
 for(i=0;i<strlen(post);i++)
 {
  if(post[i]>='0' && post[i]<='9')
  {
   pushstack(i);
  }
  if(post[i]=='+' || post[i]=='-' || post[i]=='*' ||
  post[i]=='/' || post[i]=='^')
  {
   calculator(post[i]);
  }
 }
 printf("\n\nResult :: %d",stack[top]);
 getch();
}
void pushstack(int tmp)
{
 top++;
 stack[top]=(int)(post[tmp]-48);
}
void calculator(char c)
{
 int a,b,ans;
 a=stack[top];
 stack[top]='\0';
 top--;
 b=stack[top];
 stack[top]='\0';
 top--;
 switch(c)
 {
  case '+':
  ans=b+a;
  break;
  case '-':
  ans=b-a;
  break;
  case '*':
  ans=b*a;
  break;
  case '/':
  ans=b/a;
  break;
  case '^':
  ans=b^a;
  break;
  default:
  ans=0;
 }
 top++;
 stack[top]=ans;
}
```



```
Insert a postfix notation :: 25 7 * 14 - 6 +


Result :: 3_
```

Alternate code

Program:

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>



struct node {

    double element;
    struct node *next;
} *head;


void push(int c);      // function to push a node onto the stack
int pop();             // function to pop the top node of stack
void traceStack();      // function to //print the stack values


int main() {
    int i = 0, j = 0;   // indexes to keep track of current position for input output strings
    char *exp = (char *)malloc(sizeof(char)*100);
    double res = 0;
    char tmp;
    head = NULL;


    printf("Enter the postfix expression: ");
    scanf("%s", exp);


    while( (tmp=exp[i++]) != '\0') {    // repeat till the last null terminator
        // if the char is operand, pust it into the stack
        if(tmp >= '0' && tmp <= '9') {
            int no = tmp - '0';
            push(no);
            continue;
        }

        if(tmp == '+') {
            int no1 = pop();
            int no2 = pop();
            push(no1 + no2);
        } else if (tmp == '-') {
            int no1 = pop();
            int no2 = pop();
            push(no1 - no2);
        } else if (tmp == '*') {
            int no1 = pop();
            int no2 = pop();
            push(no1 * no2);
        } else if (tmp == '/') {
            int no1 = pop();
            int no2 = pop();
            push(no1 / no2);
        }
    }

    printf("Result of the evalution is %d", pop());
    return 0;
}


void push(int c) {
    if(head == NULL) {
        head = malloc(sizeof(struct node));
        head->element = c;
        head->next = NULL;
    } else {
        struct node *tNode;
        tNode = malloc(sizeof(struct node));
        tNode->element = c;
        tNode->next = head;
        head = tNode;
    }
}
```

```
int pop() {

    struct node *tNode;

    tNode = head;

    head = head->next;

    return tNode->element;

}
```

Sample Output:

```
Enter the postfix expression: 65*3+

Result of the evalution is 33
```

Alternate code

```
#define SIZE 50            /* Size of Stack */
#include <ctype.h>
int s[SIZE];
int top=-1;       /* Global declarations */

push(int elem)
{                        /* Function for PUSH operation */
 s[++top]=elem;
}

int pop()
{                       /* Function for POP operation */
 return(s[top--]);
}

main()
{                          /* Main Program */
 char pofx[50],ch;
 int i=0,op1,op2;
 printf("\n\nRead the Postfix Expression ? ");
 scanf("%s",pofx);
 while( (ch=pofx[i++]) != '\0')
 {
  if(isdigit(ch)) push(ch-'0'); /* Push the operand */
  else
  {         /* Operator,pop two  operands */
   op2=pop();
   op1=pop();
   switch(ch)
   {
   case '+':push(op1+op2);break;
   case '-':push(op1-op2);break;
   case '*':push(op1*op2);break;
   case '/':push(op1/op2);break;
   }
  }
 }
 printf("\n Given Postfix Expn: %s\n",pofx);
 printf("\n Result after Evaluation: %d\n",s[top]);
}
```

Alternate code

```
#include <stdio.h>
#include <stdlib.h>

struct stackNode {
    int data;
    struct stackNode *nextPtr;
};
typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;

int evaluatePostfixExpression( char *expr );
int calculate( int op1, int op2, char operator );
void push( StackNodePtr *topPtr, int value );
int pop( StackNodePtr *topPtr );
int isEmpty( StackNodePtr topPtr );
void printStack( StackNodePtr topPtr );

char postfix[50];
int answer;



void main()
{
    printf("Print an postfix expression\n");
    scanf("%s", postfix);
    evaluatePostfixExpression(postfix);
    printf("The value of the expression is: %i",answer);
}

int evaluatePostfixExpression( char *expr )
    //Evaluate the postfix expression.
{
    StackNode node;
    StackNodePtr ptrnode;
    ptrnode = &node;

    int x;
    int y;
    int z;

    strcat(ptrnode,'\0');/*Append the null character ('\0') to the end of the postfix
expression.
    When the null character is encountered, no further processing is necessary.*/

    int i=0;
    for(i; postfix[i]!='\0'; i++){      //While '\0' has not been encountered, read
the expression from left to right.
        if(isdigit(postfix[i])){           //If the current character is a digit,Push
its integer value onto the stack
            push(&ptrnode, postfix[i]); //(the integer value of a digit character is
its value in the computer’s character
            printStack(ptrnode);                           //set minus the value of
'0' in the computer’s character set).
        }
        else if(postfix[i]=='+'||postfix[i]=='-
'||postfix[i]=='*'||postfix[i]=='/'||postfix[i]=='^'){                   //Otherwise,
if the current character is an operator, Pop the two top elements of the stack into
            x=pop(&ptrnode);    //variables x and y. Calculate y operator x.
            printStack(ptrnode);
            y=pop(&ptrnode);
            printStack(ptrnode);
            z=calculate(x,y,postfix[i]);
            push(&ptrnode, z);          /*Push the result of the calculation onto the
stack.*/
            printStack(ptrnode);
        }
        if (postfix[i]=='\0'){  //When the null character is encountered in the
expression, pop the top value of the
            answer = pop(&ptrnode);//stack. This is the result of the postfix
expression.
            printStack(ptrnode);
        }
    }
}

int calculate( int op1, int op2, char operator )
    //Evaluate the expression op1 operator op2.
{
    if (operator=='+')
        return op1+op2;

    else if (operator=='-')
        return op1-op2;

    else if (operator=='*')
        return op1*op2;
    else if (operator=='/')
        return op1/op2;

    else if (operator=='^')
        return op1^op2;



    else{
    return printf("calculation error");
    }
}

void push( StackNodePtr *topPtr, int value )
    //Push a value on the stack.
{
    StackNodePtr temp; /* to create a new node */
    temp =  malloc(sizeof(value));//need Malloc because it will not remember it
    temp->data = value;
    temp->nextPtr = NULL; /* the new node points to NULL */


    if(isEmpty(*topPtr)==0) {
    temp->nextPtr = *topPtr;
    }

}

int pop( StackNodePtr *topPtr )
    //Pop a value off the stack.
{
    char Data ; /* to be used to store the data */
    StackNodePtr tmp; /* to be used for handling the node*/
                        /* that is going to be deleted */
    tmp = *topPtr; /* tmp has the address of the node */
                        /* that is going to be deleted */
    Data = tmp->data;
    *topPtr = tmp->nextPtr;

    free(tmp);
    return Data;
}

int isEmpty( StackNodePtr topPtr )
    //Determine if the stack is empty.
{
    if(topPtr->nextPtr==NULL)
        return 1;
    else
        return 0;
}

void printStack( StackNodePtr topPtr )
```

```
    //Print the stack.
{
    while ((topPtr->nextPtr)!=NULL){
        printf("%C",topPtr->data);
    }
    if ((topPtr->nextPtr)==NULL)
        printf("NULL");

    printStack(topPtr->nextPtr);
}
```

Alternate code

```
// C program to evaluate value of a postfix expression
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

// Stack type
struct Stack
{
    int top;
    unsigned capacity;
    int* array;
};

// Stack Operations
struct Stack* createStack( unsigned capacity )
{
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct
Stack));

    if (!stack) return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (int*) malloc(stack->capacity * sizeof(int));

    if (!stack->array) return NULL;

    return stack;
}

int isEmpty(struct Stack* stack)
{
    return stack->top == -1 ;
}

char peek(struct Stack* stack)
{
    return stack->array[stack->top];
}
```

```
char pop(struct Stack* stack)
{
    if (!isEmpty(stack))
        return stack->array[stack->top--] ;
    return '$';
}

void push(struct Stack* stack, char op)
{
    stack->array[++stack->top] = op;
}



// The main function that returns value of a given postfix
expression
int evaluatePostfix(char* exp)
{
    // Create a stack of capacity equal to expression size
    struct Stack* stack = createStack(strlen(exp));
    int i;

    // See if stack was created successfully
    if (!stack) return -1;

    // Scan all characters one by one
    for (i = 0; exp[i]; ++i)
    {
        // If the scanned character is an operand or number,
        // push it to the stack.
        if (isdigit(exp[i]))
            push(stack, exp[i] - '0');

        //  If the scanned character is an operator, pop two
        // elements from stack apply the operator
        else
        {
            int val1 = pop(stack);
            int val2 = pop(stack);
            switch (exp[i])
            {
             case '+': push(stack, val2 + val1); break;
             case '-': push(stack, val2 - val1); break;
             case '*': push(stack, val2 * val1); break;
             case '/': push(stack, val2/val1);   break;
            }
        }
    }
    return pop(stack);
}

// Driver program to test above functions
int main()
{
    char exp[] = "231*+9-";
    printf ("Value of %s is %d", exp, evaluatePostfix(exp));
    return 0;
}
```

Output:

```
Value of 231*+9- is -4
```

Program :

- **This Program Accepts Operators : +,-,/,*,(,)**
- **Sample Infix Expression : (a+b)*c/(a+b*c)**

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#define MAX 100

typedef struct stack
{
 int data[MAX];
 int top;
}stack;

int priority(char);
void init(stack *);
int empty(stack *);
int full(stack *);
char pop(stack *);
void push(stack *,char);
char top(stack *);

void main()
{
stack s;
char x;
int token;
init(&s);
clrscr();
printf("nEnter infix expression:");
  while((token=getchar())!='n')
  {
    if(isalnum(token))
       printf("%c",token);
    else
```

```
       if(token == '(')
           push(&s,'(');
       else
       {
         if(token == ')')
             while((x=pop(&s))!='(')
             printf("%c",x);
         else
         {
         while(priority(token)< =priority(top(&s)) && !empty(&s))
             {
             x=pop(&s);
             printf("%c",x);
             }
         push(&s,token);
         }
       }
  }
  while(!empty(&s))
    {
    x=pop(&s);
    printf("%c",x);
    }
getch();
}
//---------------------------------------------
int priority(char x)
{
   if(x == '(')
 return(0);
   if(x == '+' || x == '-')
 return(1);
   if(x == '*' || x == '/' || x == '%')
 return(2);
   return(3);
}
//---------------------------------------------
void init(stack *s)
{
   s->top=-1;
}
//---------------------------------------------
int empty(stack *s)
{
    if(s->top==-1)
 return(1);
    else
 return(0);
}
//----------------------------------------------
int full(stack *s)
{
    if(s->top==MAX-1)
 return(1);
    else
 return(0);
}
//----------------------------------------------
void push(stack *s,char x)
{
  s->top=s->top+1;
  s->data[s->top]=x;
}
//----------------------------------------------
char pop(stack *s)
{
   int x;
   x=s->data[s->top];
   s->top=s->top-1;
   return(x);
}
//----------------------------------------------
char top(stack * s)
{
   return(s->data[s->top]);
}
//----------------------------------------------
```

Explanation :

```
while((token=getchar())!='n')
```

- Accepts Expression Character by Character **Till Entered Character is 'n'**
- After Accepting Single Character do all actions **inside while loop.**

```
if(isalnum(token))
 printf("%c",token);
```

- isalnum(token)   checks whether entered Character is Alphabetic or Numeric .
- If Entered Character is alpha-numeric then Directly Display it – for more help [Algo] + [Example]

- If Entered Character is Opening Bracket then **Push Element Onto Stack**

```
if(token == '(')
     push(&s,'(');
```

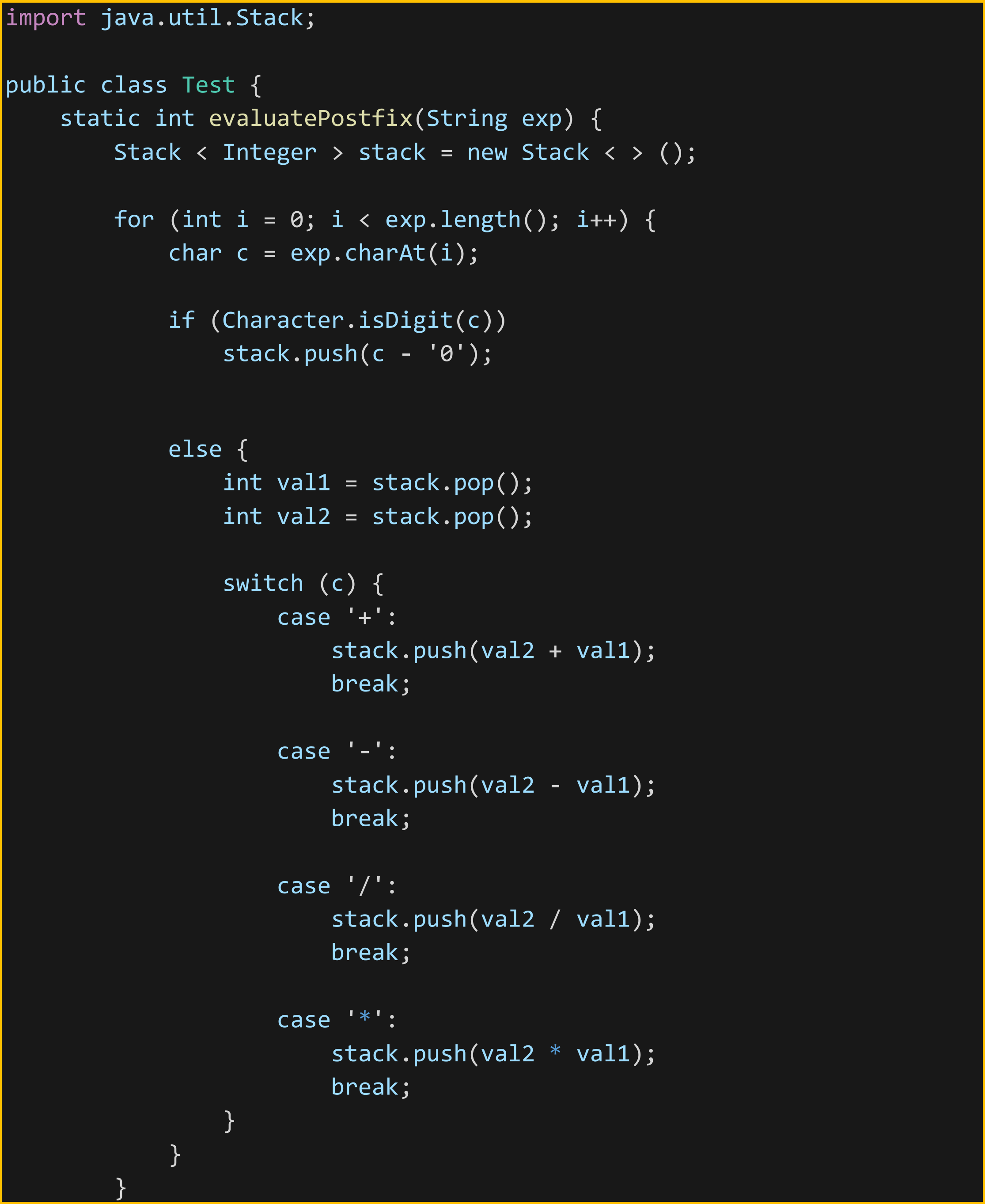
- If Entered Character is ‘Closing Bracket’ then **Pop All Elements till Equivalent Opening Bracket is poped.**

```
if(token == ')')
     while((x=pop(&s))!='(')
  printf("%c",x);
```

- If Entered Character is Operator then – do

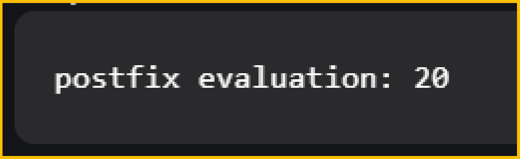
```
else
   {
   while(priority(token)< =priority(top(&s)) && !empty(&s))
     {
     x=pop(&s);
     printf("%c",x);
     }
   push(&s,token);
   }
```

LAB 9



```
import java.util.Stack;

public class Test {
    static int evaluatePostfix(String exp) {
        Stack < Integer > stack = new Stack < > ();

        for (int i = 0; i < exp.length(); i++) {
            char c = exp.charAt(i);

            if (Character.isDigit(c))
                stack.push(c - '0');


            else {
                int val1 = stack.pop();
                int val2 = stack.pop();

                switch (c) {
                    case '+':
                        stack.push(val2 + val1);
                        break;

                    case '-':
                        stack.push(val2 - val1);
                        break;

                    case '/':
                        stack.push(val2 / val1);
                        break;

                    case '*':
                        stack.push(val2 * val1);
                        break;
                }
            }
        }
        return stack.pop();
    }

    public static void main(String[] args) {
        String exp = "254*+2-";
        System.out.println("postfix evaluation: " + evaluatePostfix(exp
));
    }
}
```

Output:



```
postfix evaluation: 20
```

LAB No: 10

Objective

Convert Infix to Postfix Expression

Theory

In **Infix Notation**, Operators are placed between operands.
In **Postfix Notation**, Operators are placed after operands.

Example: A + B * C

symbol	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC *	+
	ABC * +	

Algorithm

```
    Q→arithmetic expression
    P→ postfix expression
1.  Push "("onto Stack, and add ")" to the end of Q.
2.  Scan Q from left to right and repeat  Step 3 to 6 for each element of Q until the stack is
    empty:
3.      If an operand is encountered, add it to P.
4.      If a left parenthesis is encountered, push it onto stack.
5.      If an operator is encountered, then:
            (a) Repeatedly pop from the stack and add to P each operator (on the top of stack)
                which has the same precedence as or higher precedence than [operator].
            (b) Add [operator] to stack.
        [End of If structure.]
6.      If a right parenthesis is encountered, then:
            (a) Repeatedly pop from the stack and add to P each operator (on the top of stack)
                until a left parenthesis is encountered.
            (b) Remove the left parenthesis. [Do not add the left parenthesis to P.]
        [End of If structure.]
    [End of Step2 loop.]
7.  Exit.
```

PROGRAM:-

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
char stack[100];
int top=0;
char exp[100];
struct table
{
  char s[2];
  int isp;
  int icp;
}pr[7];
int isp(char c)
{
int i;
 for(i=0;i<=6;i++)
 if(pr[i].s[0]==c)
  return(pr[i].isp);
  return 0;
}
int icp(char c)
{
int i;
 for(i=0;i<=6;i++)
 if(pr[i].s[0]==c)
  return(pr[i].icp);
  return 0;
}
void main()
{
 int i;
 clrscr();
strcpy(pr[0].s,"^");
pr[0].isp=3;
pr[0].icp=4;

strcpy(pr[1].s,"*");
pr[1].isp=2;
pr[1].icp=2;

strcpy(pr[2].s,"/");
pr[2].isp=2;
pr[2].icp=2;
strcpy(pr[3].s,"+");
pr[3].isp=1;
pr[3].icp=1;

strcpy(pr[4].s,"-");
```

```
pr[4].isp=1;
pr[4].icp=1;

strcpy(pr[5].s,"(");
pr[5].isp=0;
pr[5].icp=4;

strcpy(pr[6].s,"=");
pr[6].isp=-1;
pr[6].icp=0;

 clrscr();
 stack[top]='=';
 printf("enter the infix expression");
 gets(exp);
 i=0;
 printf("the postfix expression is ")
 while(i<strlen(exp))
 {
 if(isalpha(exp[i])==0)
 {

    if(exp[i]==')')
       {
             while(stack[top]!='(')
                          {
                          printf("%c",stack[top]);
                          top--;
                          }
                          top--;
       }
       else
              {
                while(isp(stack[top])>=icp(exp[i]))
                {
                printf("%c",stack[top]);
                top--;
                }
                top++;
                stack[top]=exp[i];
              }
  }
  else
   printf("%c",exp[i]);
 i++;
 }
 while(top>0)
 {
 printf("%c",stack[top]);
 top--;
 }
 getch();
}
```

OUTPUT:-

```
enter the infix expression a*(s+d/f)+c
the postfix expression is asdf/+*c+
```

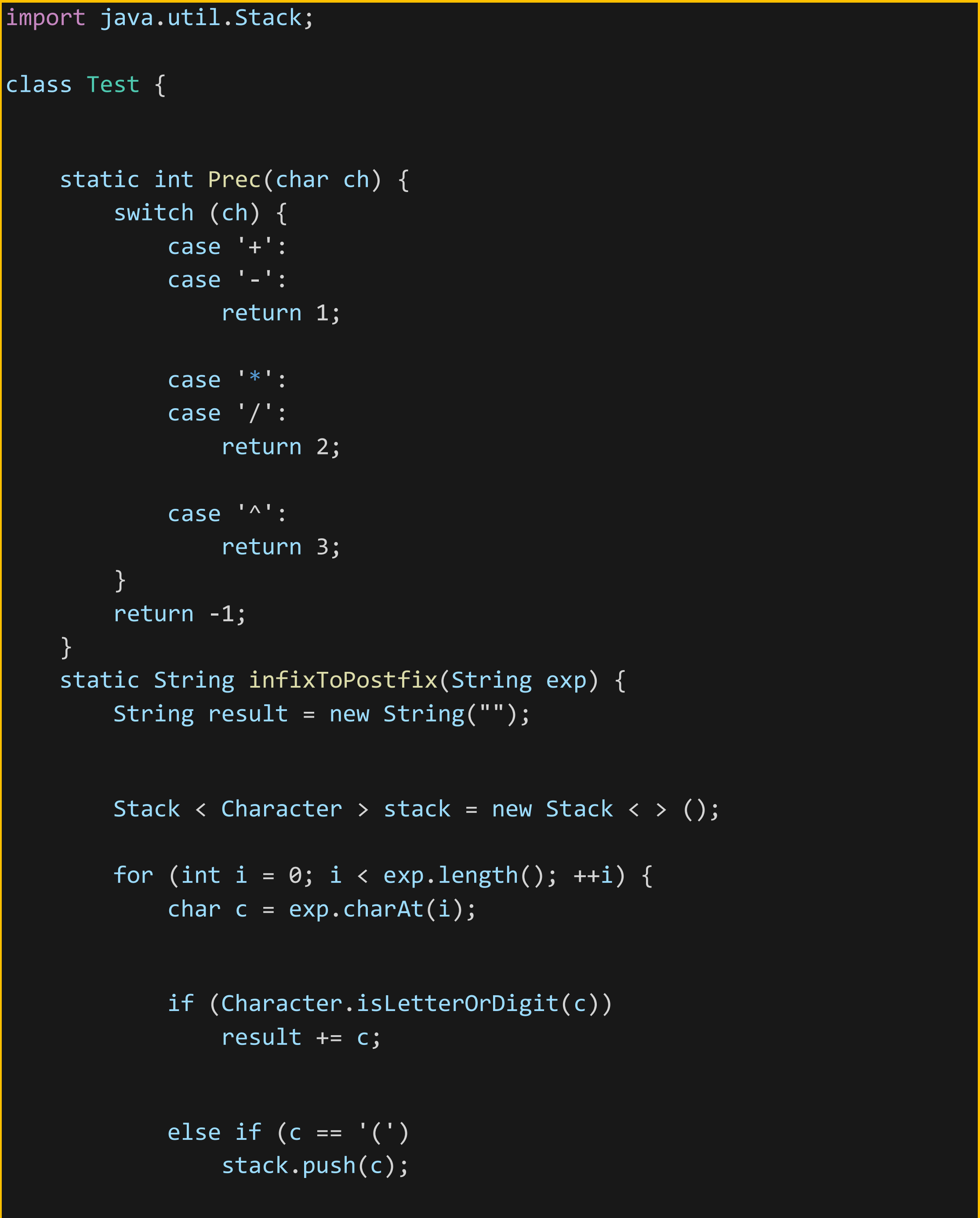
TASK CODE:

```
#define SIZE 50

#include <ctype.h>
char s[SIZE];
int top=-1;

void push(char elem)
{
    s[++top]=elem;
}

char pop()
{
    return(s[top--]);
}

int pr(char elem)
{
    switch(elem)
    {
    case '#':
      return 0;

      case '(':
      return 1;

      case '+':
      case '-':
      return 2;

      case '*':
      case '/':
```

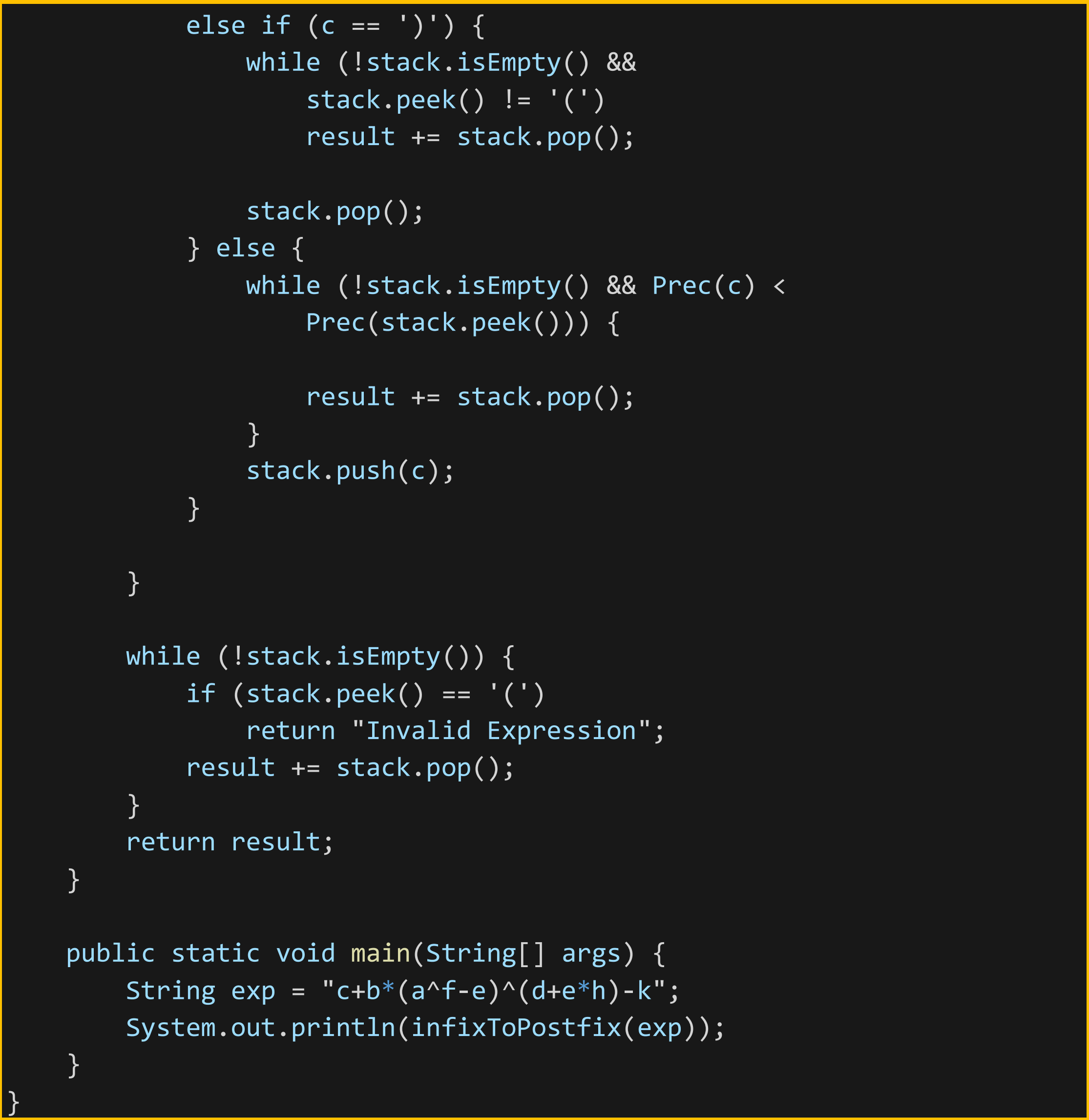
```
     return 3;
    }
     Return 0;
}


void main()
{
    char infx[50],pofx[50],ch,elem;
    int i=0,k=0;
     clrscr();
     printf("\n Enter Infix Expression:");
     scanf("%s",infx);
     push('#');

     while( (ch=infx[i++]) != '\0')
    {
        if( ch == '(')
           push(ch);
        else
            if(isalnum(ch))
        pofx[k++]=ch;
            else
                if( ch == ')')
                {
                    while( s[top] != '(')
                        pofx[k++]=pop();
                    elem=pop();
                }
                else
                {       /* Operator */
                    while( pr(s[top]) >= pr(ch) )
                        pofx[k++]=pop();
                    push(ch);
                }
    }
    while( s[top] != '#')
    pofx[k++]=pop();
    pofx[k]='\0';
```

```
        printf("\n\nGiven Infix Expression is: %s\nPostfix Expression
     is: %s\n",infx,pofx);
    getch();
}
```

LAB 10

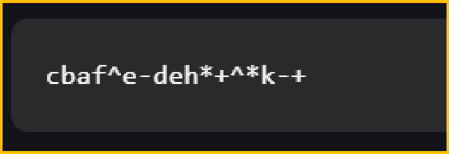


```
import java.util.Stack;

class Test {


    static int Prec(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;

            case '*':
            case '/':
                return 2;

            case '^':
                return 3;
        }
        return -1;
    }
    static String infixToPostfix(String exp) {
        String result = new String("");


        Stack < Character > stack = new Stack < > ();

        for (int i = 0; i < exp.length(); ++i) {
            char c = exp.charAt(i);


            if (Character.isLetterOrDigit(c))
                result += c;


            else if (c == '(')
                stack.push(c);
```



```
            else if (c == ')') {
                while (!stack.isEmpty() &&
                    stack.peek() != '(')
                    result += stack.pop();

                stack.pop();
            } else {
                while (!stack.isEmpty() && Prec(c) <
                    Prec(stack.peek())) {

                    result += stack.pop();
                }
                stack.push(c);
            }

        }

        while (!stack.isEmpty()) {
            if (stack.peek() == '(')
                return "Invalid Expression";
            result += stack.pop();
        }
        return result;
    }

    public static void main(String[] args) {
        String exp = "c+b*(a^f-e)^(d+e*h)-k";
        System.out.println(infixToPostfix(exp));
    }
}
```

Output:



```
cbaf^e-deh*+^*k-+
```