EX. NO: 1 FIBONACCI NUMBER USING RECURSION

Aim:

Write recursive programme which computes the n th Fibonacci number, for appropriate values of n. Analyze behavior of the programme Obtain the frequency count of the statement for various values of n.

Description: The **Fibonacci numbers** or **Fibonacci series** are the numbers in the following integer sequence: 0,1,1,2,3,5,8,13,21,…. .By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation $\mathbf{F_{n}= F_{n-1} + F_{n-2}}$

Algorithm:

```
Input: Read n value
Output: Prints the nth Fibonacci term
Step1: Start
Step2: Read n value for computing nth term in Fibonacci series
Step3: call  Fibonacci (n)
Step4: Print the nth term
Step5: End
```

```
 Fibonacci(n)
// Algorithm Fibonacci computes the nth Fibonacci number, for appropriate values of n.
Step1: If n = 0 then go to step2 else go to step3
Step2: return 0
Step3: If n = 1 then go to step4 else go to step5
Step4: return 1
Step5: return(Fibonacci (n-1) + Fibonacci (n-2))
```

Sample Input:

Fibonacci number upto 8 is 11

Observed Output:

```
                    Fibonacci Number Computation
        For which term you want to compute the Fibonacci number 9
        Fibonacci Number of 9 is 19
```

Viva questions:

1. What is Fibonacci series?
2. What is the Logic of Fibonacci series?
3. Give example of Fibonacci series:
4. What is recursion?
5. What are the advantages of the functions?
6. What is data structure?
7. What are the goals of Data Structure?
8. What does abstract Data Type Mean?

EX. NO: 2(A) FACTORIAL USING RECURSION AND NON-RECURSION

Aim:

Write recursive and non recursive C programme for calculation of Factorial of an integer

Description:

A recursive function defines values of the functions for some inputs in terms of the values of the same function for other inputs. Simply a recursion function can be defined as a function call to itself. The recursion concept was implemented using the data structure called STACKS.The **factorial** of a non-negative integer *n*, denoted by *n*! . It is the product of all positive integers less than or equal to *n*. For example, 5! =5*4*3*2*1=120

Algorithm:

/* factorial_recursive and NonRecursive*/

Input: integer n

Output: factorial of given number

```
1. Start.
2. Get the number nto which Factorial value is to be calculated.
3. Print Menu 1. Recursive function 2. Non-recursive function
4. If choice=1 then Call Factorial(n).
5. Else if choice=2 then Call NonRecFactorial(n)
6. Printf factorial Number
7. Stop

 Factorial(n) 1.Start
2. If n= 0 or 1 then fact=1
3. else fact= n* factorial(n-1)
4. return(fact)
```

```
NonRecFactorial(n)
1.Start
2. for i=1 to i<=n do
3. fact=fact*i;
4. return (fact)
```

Sample Input:

```
Factorial of 5 is 120
```

Observed Output:

Output1:

```
Factorial of recursion and non recursion
Enter the number 6
Enter the choice 1
Recursion Factorial of 6 is 720
```

Output 2:

```
Factorial of recursion and non recursion
Enter the number 7
Enter the choice 2
Recursion Factorial of 7 is 5040
```

EX. NO: 2(B) GCD USING RECURSION AND NON-RECURSION

Aim:

Write recursive and nonrecurive C program for calculation of GCD (n, m)

Description:

In mathematics, the greatest common divisor (GCD), also known as the greatest common factor (GCF), or highest common factor (HCF), of two or more non-zero integers, is the largest positive integer that divides the numbers without a remainder

For example, the GCD of 8 and 12 is 4.

Algorithm:

```
                         /* GCD recursive and Non-Recursive */
Input: integer a, b
Output: GCD of a, b

1. Start.
2. Get two numbersm, n for which GCD is to be calculated.
3. Print Menu 1. Recursive function 2. Non-recursive function
4. If choice=1 then Call  GCD(m,n).
5. Else if choice=2 then Call NonRec GCD(m,n).
6. Print GCD
7. Stop.

 GCD(int a, int b)
1.begin

2. if a = 0 then go to step3 else go to step4
3. return b
4. if b = 0 then go to step5 else go to step6
```

```
5. return a
6. g= gcd_rec (b, a mod b)
7. return g
```

Sample Input:

GCD of 16 and 4 is: 4

Observed Output:

```
Computing GCD of TWO numbers
Enter two numbers  36 54
The Recursion GCD of 36 and 54 is 18
The Non-Recursion GCD of 36 and 54 is 18
```

EX. NO: 2(C) TOWERS OF HANOI USING RECURSION

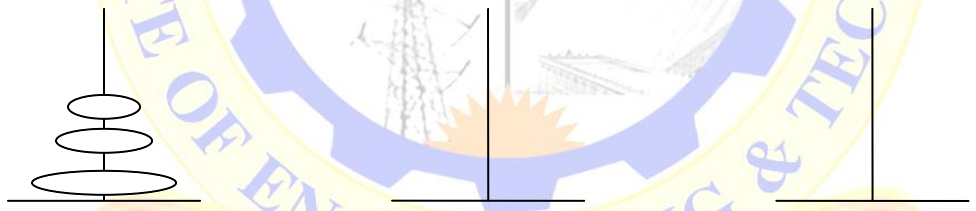
Aim:

Write recursive C programme for Towers of Hanoi: N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg.

Description:

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which should be transferred from soruce to destination using the intermediate rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time
- Each move consists of taking the upper disk from one of the rods and sliding into another rod, on the top of the other disks that may be already present on the rod.
- No disk may be placed on top of a smaller disk.



Source **Intermediate** **Destination**

Algorithm:

/* Towers of Hanoi recursive */

Input: integer n number of disks

Output:N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg

```
1.Start
2. read n value as the no. of disks
3. call  TOH(N, S, I, D).
4. Stop
```

```
TOH(N, S, I, D)
1.begin
2. if n = 1 then
3. Transfer disk from S to D and stop
4. else
5. transfer N-1 disks from peg S to peg I with peg D as the intermediate peg
6. Call  TOH(N-1, S, D, I)
7. Transfer disk from S to D
8. transfer N-1 disks from peg I to peg D with peg S as the intermediate peg
9. Call  TOH(N-1, I, S, D);
10.end
```

Sample Input:

```
ENTER NUMBER OF DISKS:3
Move diskA from S to D
Move diskB from S to I
Move diskA from D to I
Move diskC from S to D
Move diskA from I to S
Move diskB from I to D
Move diskA from S to D
```

Observed Output:

```
        TOWERS OF HANOI
Enter the number of disks :3
Move disk-1 from S to D
Move disk-2 from S to I
Move disk-1 from D to I
Move disk-3 from S to D
```

Move disk-1 from I to S

Move disk-2 from I to D

Move disk-1 from S to D

Viva questions:

1. What actions are performed when a function is called?
3. What actions are performed when a function returns?
4. What is the logic of factorial function?
5. Give an example of finding factorial?
6. What is tail recursion
6. What is the logic of GCD?
7. Give an example for GCD
8. Explain input parameters of towers of Hanoi algorithm
9. What is formulae for finding the no of moves required for n disks in towers of Hanoi problem
10. What is the data structures used to perform recursion?

EX. NO: 3(A) LINEAR SEARCH USING RECURSIVE & NON-RECURSIVE FUNCTIONS

Aim:

Write C programs that use both recursive and non recursive functions to perform Linear search for a Key value in a given list.

Description:

The linear search is most simple searching method. It does not expect the list to be sorted. The key which is to be searched is compared with each element of the list one by one. If a match exists, the search is terminated. If the end of list is reached it means that the search has failed and key has no matching in the list.

Algorithm:

```
1.Start
2. Read the array size _n‘
3. Read elements into array _L‘
4. Read the key to be searched in array _K‘
5. Print Menu 1. Recursive function 2. Non-recursive function
6. Read the choice
7. if choice=1 then call  pos = LINEAR_SEARCH_REC(L, n, K)
8. else if choice=2 then call pos = LINEAR_SEARCH_NONREC(L, n, K)
9. ifpos< 1 then print _ key not found‘
10. else print _key found in index position ‘, i
11. Stop

 LINEAR_SEARCH_NONREC(L, n, K) 1.i = 0
2. while (( i < n) and (K != L[i])) do
3. i = i + 1;
4. end while
5. if ( K = L[i]) then print (— KEY found‖) and return (i)
```

```
6. else print (— KEY not found‖)
7. Stop
```

LINEAR_SEARCH_REC(L, n, K)

```
1.if ( K = L[n-1]) then print (— KEY found‖) and return (n)
2.else if(n==0) then    return(-1)
3. elsereturn LINEAR_SEARCH_REC(L, n-1, K)
```

Sample Input:

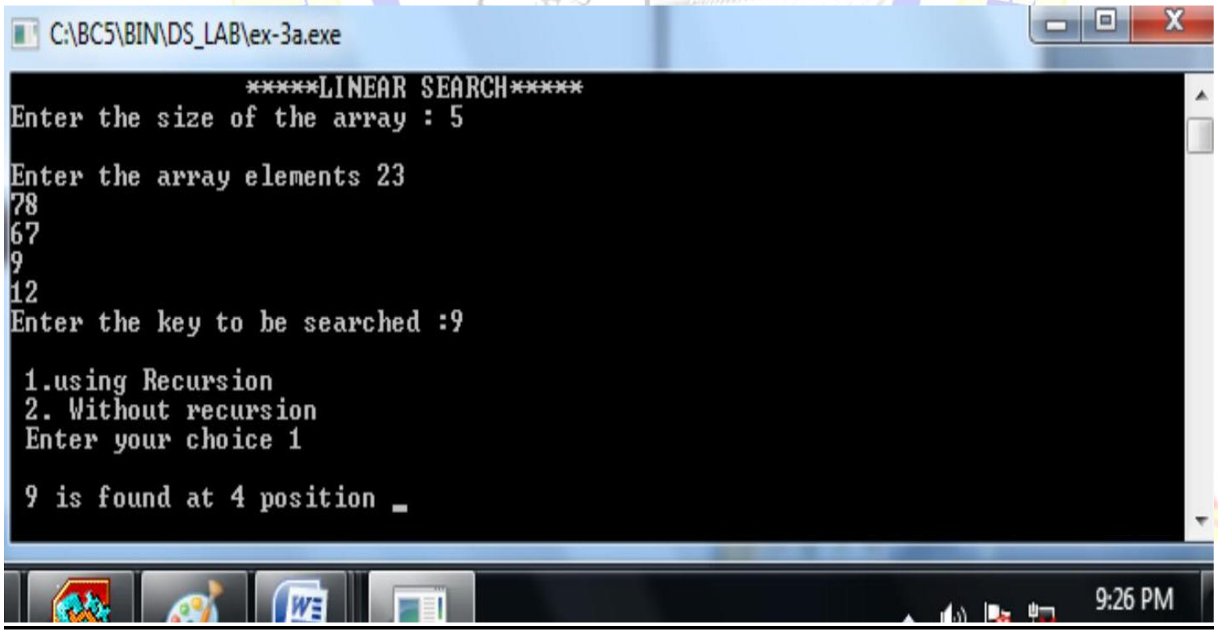
Enter the size of array: 5

Enter the elements: 65   78   2   1   9

Key : 2

2 is found

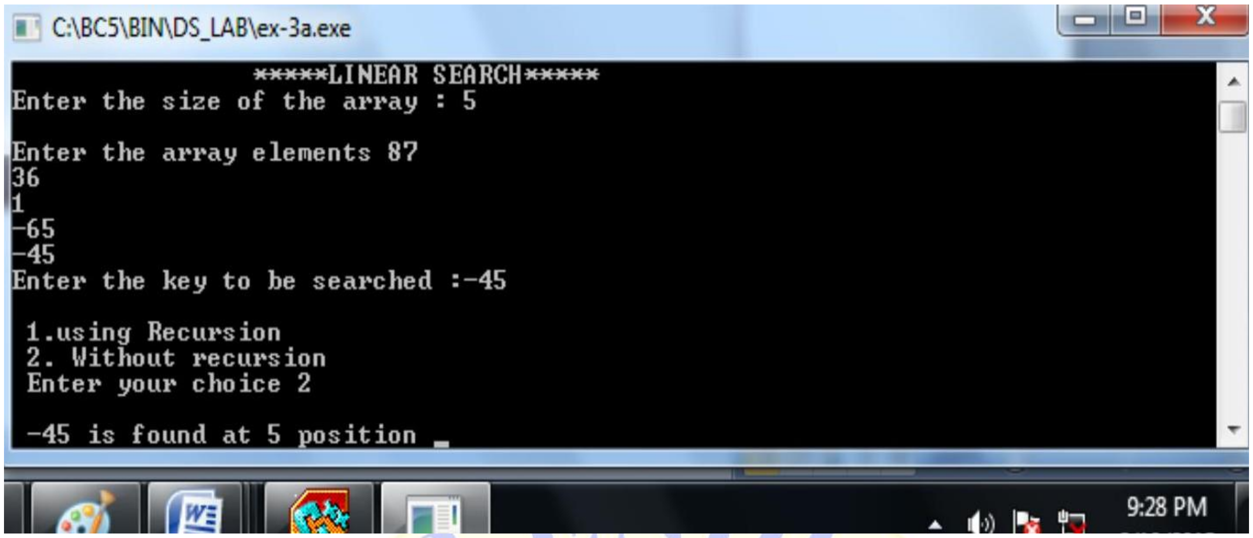
Observed Output:



C:\BC5\BIN\DS_LAB\ex-3a.exe

```
                *****LINEAR SEARCH*****
Enter the size of the array : 5

Enter the array elements 23
78
67
9
12
Enter the key to be searched :9

 1.using Recursion
 2. Without recursion
 Enter your choice 1

 9 is found at 4 position _
```

9:26 PM



```
                *****LINEAR SEARCH*****
Enter the size of the array : 5

Enter the array elements 87
36
1
-65
-45
Enter the key to be searched :-45

 1.using Recursion
 2. Without recursion
 Enter your choice 2

 -45 is found at 5 position _
```

EX. NO: 3(B) BINARY SEARCH USING RECURSIVE & NON- RECURSIVE FUNCTIONS

Aim:

Write C programs that use both recursive and non recursive functions to perform Binary search for a Key value in a given list.

Description:

Binary search is a vast improvement over the sequential (Linear) search. For binary search to work, the item in the list must be in a sorted order. The approach employed in the binary search is divide and conquer. In binary search the list is divided into two half's based on the MID value, and the key is compared with the mid element. If it is successful then it returns the mid value, if the key value is not found at mid then the search will proceed in any one of the half's based on whether the key element is greater or lesser than the mid element. If the list to be sorted for a specific item is not sorted, binary search fails. The Mid value can be calculated by the formula:

MID= (HIGH+LOW)/2

Algorithm:

```
1.Start
2. Read the array size _n‘
3. Read elements into array _L‘
4. Read the key to be searched in array _K‘
5. low=0, high = N – 1
6. Print Menu 1. Recursive function 2. Non-recursive function
7. Read the choice
8. if choice=1 then call to the function pos = BINARY_SEARCH_REC(L, low, high, K)
9. else if choice=2 then pos = BINARY_SEARCH_NONREC(L, low, high, K)
10. ifpos< 1 then print _ key not found‘
11. else print _key found in index position ‘,i
12. Stop
```

```
 BINARY_SEARCH_NONREC(L, low, high, K)
1. begin
```

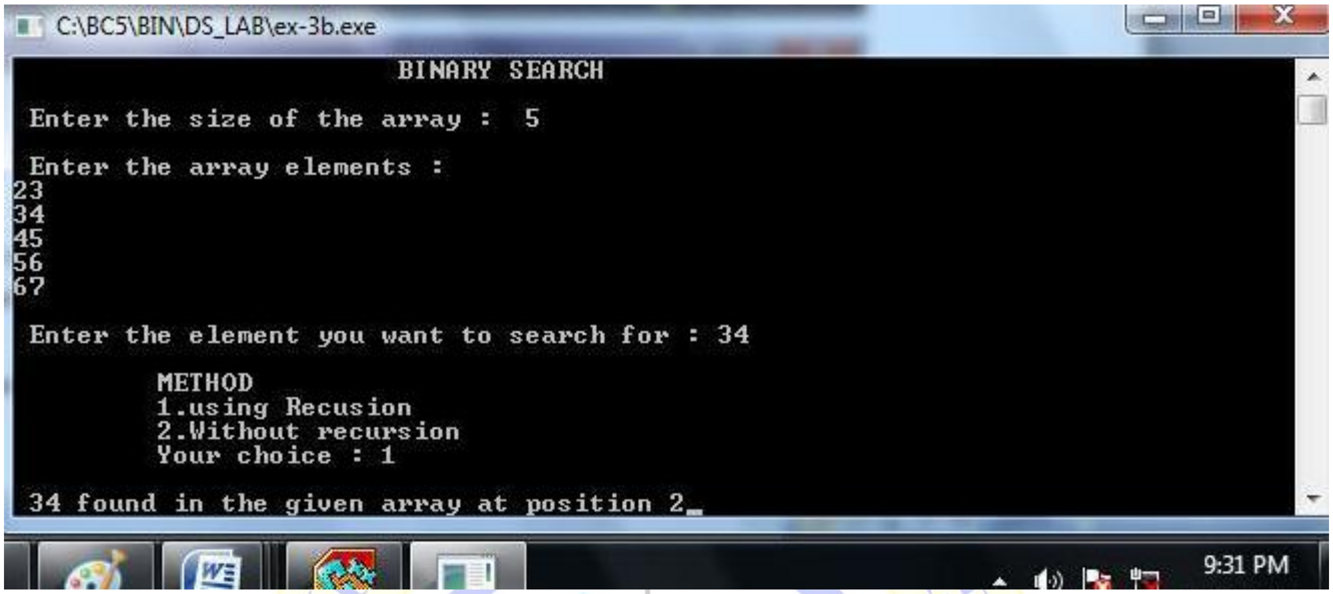
```
2. repeat steps 3 to 10 until low <= high
3. mid=(low+high)/2
4. if(K = L[mid])then go to step5 else go to step7
5. loc=mid
6. return loc
7. else if(K <L[mid]) then go to step8 else go to step9
8. high= mid-1 and go to step 10
9. low=mid+1
10. return -1
```

```
 BINARY_SEARCH_REC(L, low, high, K)
1.begin
2. if( low > high) then go to step3else go to step4
3. return -1;
4. mid=(low+high)/2;
5. if (K < L[mid]) then go to step6 else go to step7
6. return BINARY_SEARCH_REC (L, low, mid-1, K)
7. if(K > L[mid]) then go to step8 else go to step10
8. return BINARY_SEARCH_REC (L, mid+1, high, K)
9.if (K = L[mid]) then return mid
10. end
```

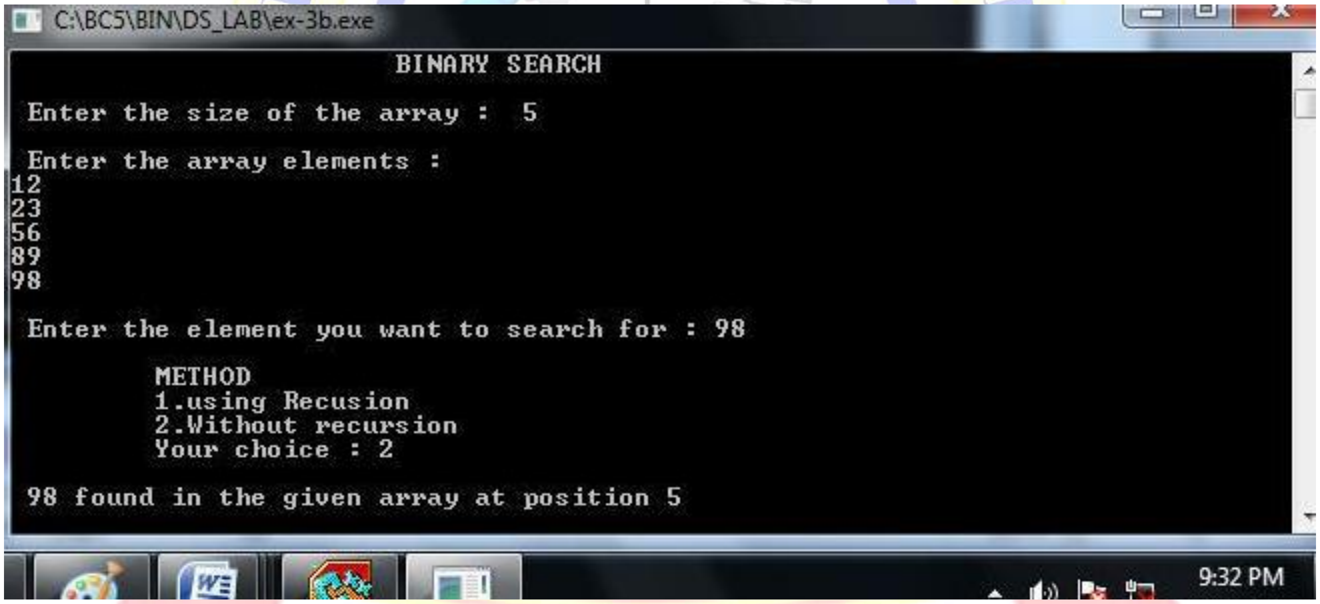
Sample Input:

```
Enter the size of the array:  4

Elements are: 23, 26, 29, 40

Enter Key: 26

Key is found at 1 position
```

Observed Output:



```
C:\BC5\BIN\DS_LAB\ex-3b.exe
                         BINARY SEARCH

 Enter the size of the array :  5

 Enter the array elements :
23
34
45
56
67

 Enter the element you want to search for : 34

         METHOD
         1.using Recusion
         2.Without recursion
         Your choice : 1

 34 found in the given array at position 2_
```



```
C:\BC5\BIN\DS_LAB\ex-3b.exe
                         BINARY SEARCH

 Enter the size of the array :  5

 Enter the array elements :
12
23
56
89
98

 Enter the element you want to search for : 98

         METHOD
         1.using Recusion
         2.Without recursion
         Your choice : 2

 98 found in the given array at position 5
```

EX. NO: 3(C) FIBONACCI SEARCH USING RECURSIVE & NON-RECURSIVE FUNCTIONS

Aim:

Write C programs that use both recursive and non recursive functions to perform Fibonacci search for a Key value in a given list.

Description:

A possible improvement in binary search is not to use the middle element at each step, but to guess more precisely where the key being sought falls within the current interval of interest. This improved version is called **Fibonacci search**. Instead of splitting the array in the middle, this implementation splits the array corresponding to the **Fibonacci numbers**, which are defined in the following manner:

$F_0 = 0, F_1 = 1$

$F_n = F_{n-1}+F_{n-2}$ for n>=2.

Algorithm:

```
1.Start
2. Read the array size _n‘
3. Read elements into array _L‘
4. Read the key to be searched in array _K‘
5. low=0, high = N – 1
6. Print Menu 1. Recursive function 2. Non-recursive function
7. Read the choice
8. if choice=1 then call to the function pos = FIBONACCI_SEARCH _REC(L, n, K)
9. else if choice=2 then pos = FIBONACCI_SEARCH _NONREC(L, n, K)
10. if pos< 1 then print _ key not found‘
11. else print _key found in index position ‘,i
```

12. Stop

FIBONACCI_SEARCH _NONREC(L, n, K)

```
/*L[1:n] is a linear ordered (non-decreasing) list of data elements. n is such that k+1>(n+1).Also
$F_k$+m=(n+1). K is the key to be searched in the list. */
Obtain the largest Fibonacci number $F_k$ closest to n+1;
p=$F_k$-1;
q=$F_k$-2; r=$F_k$-3;
m=(n+1)-(p+q);
if (k>L[p]) then
p=p+m;
found =false;
while ( (p=!0) and (not found )) do
case: k=L[p]:
{
            print (―key found‖); /* key found */
             found =true;
}
Case:k<L[p]:
                        if (r=0) then p=0
                        else
                        {
                                  p=p-r; t=q; q=r; r=t-r;
                        }
Case: k> L[p]:
                        if (q=1) then p=0
                        else
                        {
                                  p=p+r; q=q-r; r=r-q
                        }
```

```
        end case
end while
if(found=false) then print (—key not found‖);
end FIBONACCI_SEARCH_NONREC.
```

FIBONACCI_SEARCH_REC (L,n,k)

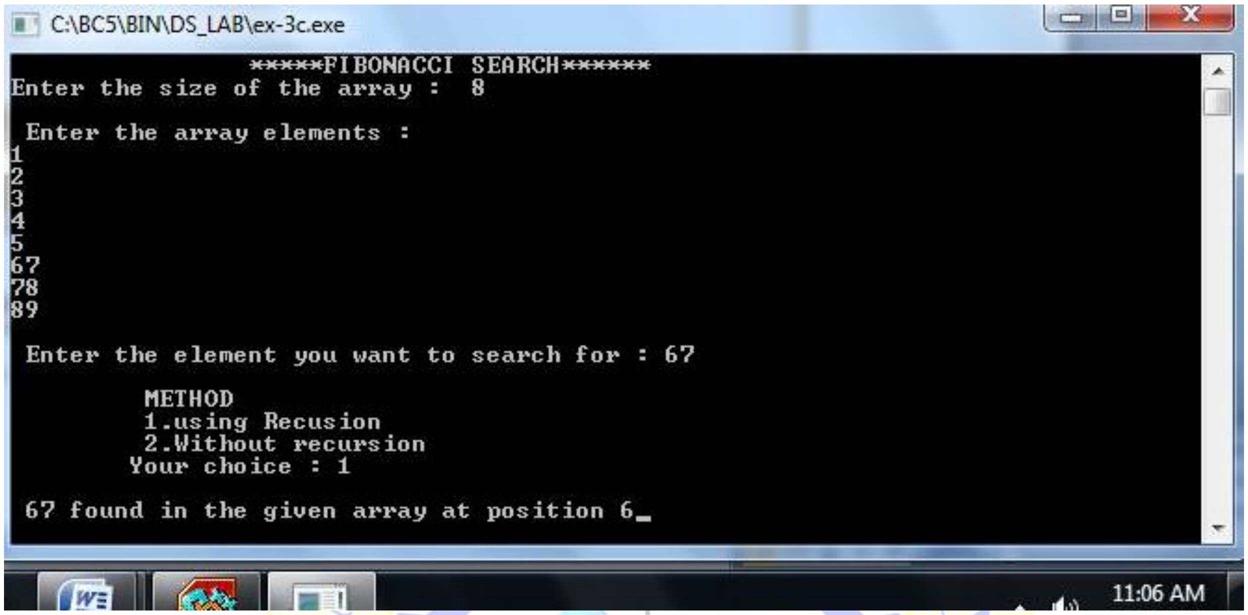
/*L[1:n] is a linear ordered (non-decreasing) list of data elements. n is such that k+1>(n+1).Also F_k+m=(n+1). K is the key to be searched in the list. */

```
Obtain the largest Fibonacci number $F_k$ closest to n+1;
p=$F_k$-1;
q=$F_k$-2; r=$F_k$-3;
m=(n+1)-(p+q);
if (k>L[p]) then p=p+m;
call  pos=Fibsearch(L,key,p,q,r);
returnpos;
end FIBONACCI_SEARCH_REC.
```

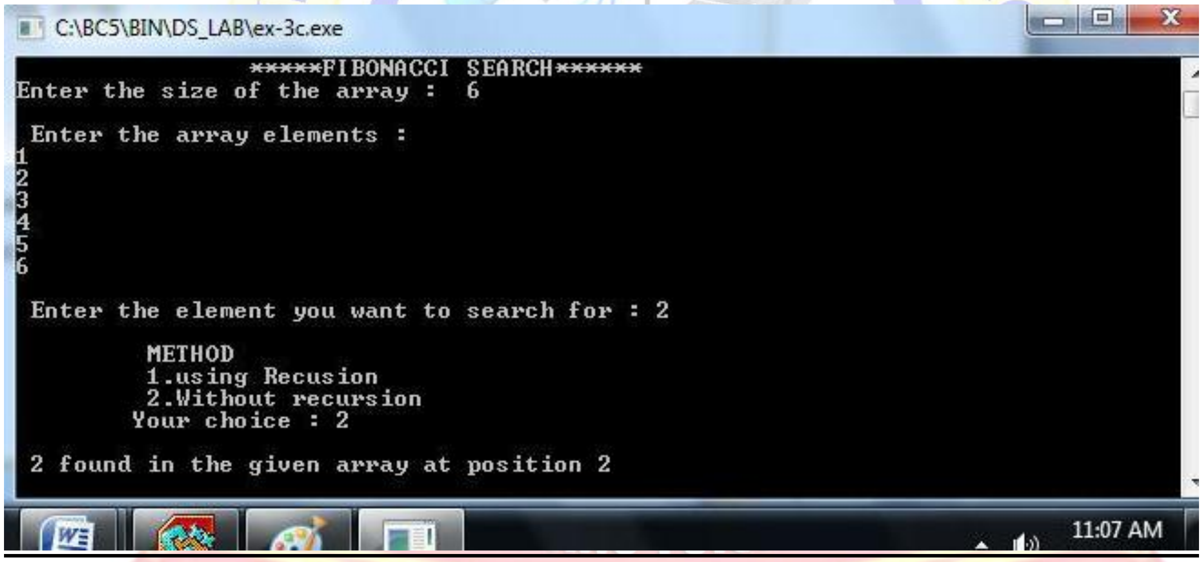
Sample Input:

```
Enter Size of Array: 4

Enter the elements: 3 12 16 45

Enter the element want to search: 45

45 found at position 4
```

Observed Output:



```
C:\BC5\BIN\DS_LAB\ex-3c.exe
                *****FIBONACCI SEARCH******
Enter the size of the array :  8

 Enter the array elements :
1
2
3
4
5
67
78
89

 Enter the element you want to search for : 67

         METHOD
         1.using Recusion
         2.Without recursion
        Your choice : 1

 67 found in the given array at position 6_
```



```
C:\BC5\BIN\DS_LAB\ex-3c.exe
                *****FIBONACCI SEARCH******
Enter the size of the array :  6

 Enter the array elements :
1
2
3
4
5
6

 Enter the element you want to search for : 2

         METHOD
         1.using Recusion
         2.Without recursion
        Your choice : 2

 2 found in the given array at position 2
```

Viva questions:

1. What is sequential search?
2. What are the advantages of linear search
3. What are the advantages of linear search
4. Write the algorithm for sequential search
5. Time complexity of linear and binary search
6. What is the necessary condition to implement binary search on a list
7. Calculate the efficiency of sequential search?

EX. NO: 4(A) BUBBLE SORT

Aim:

Write C programs that implement Bubble sort, to sort a given list of integers in ascending order.

Description:

Bubble sort is the simplest and oldest sorting technique. This method takes two elements at a time. It compares these two elements. If first elements is less than second one, they are left un disturbed. If the first element is greater then second one then they are swapped. The continues with the next two elements goes and ends when all the elements are sorted. But bubble sort is an inefficient algorithm. The order of bubble sort algorithm is O(n^2).

Algorithm:

```
                /* Bubble_ Sort */

Input: An integer n and a list of n elements stored in array elements a[0], . . . , a[n – 1]
Output: sorted array

1.Start
2. Read the size of the array _n‘
3. Read the elements of the array _ L‘
4. call  BUBBLE_SORT(L,n)
5. print array _L‘
6. Stop


 BUBBLE_SORT(L,n)

/* L[1: n] is an unordered list of data elements to be sorted in the ascending order */
for i = 1 to n-1 do       /* n – 1 passes */
        for j = 1 to n-1 do
```

```
                    if( L[i] > L[j] ) then
swap( L[i], L[j]);        /* swap pair wise elements     */
          end      /* the next largest element ―bubbles‖ to the last position     */
end
end BUBBLE_SORT.
```

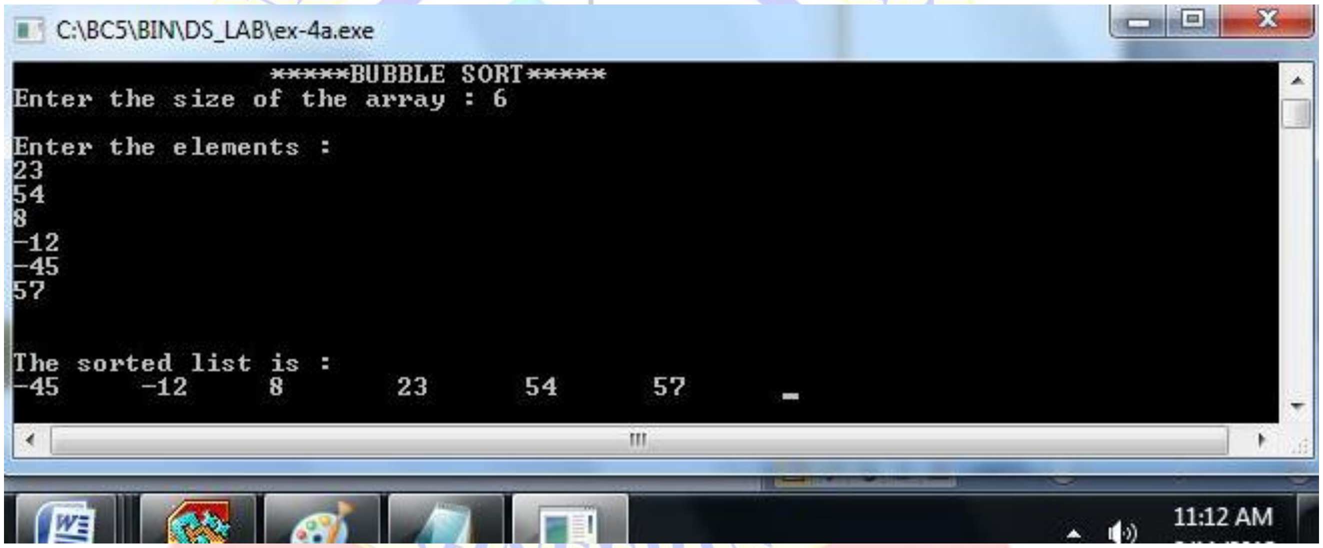
Sample Input:

Enter the array size: 3

Enter elements: 54 67 12

The sorted array is: 12 54 67

Observed Output:



C:\BC5\BIN\DS_LAB\ex-4a.exe

```
                         *****BUBBLE SORT*****
Enter the size of the array : 6

Enter the elements :
23
54
8
-12
-45
57


The sorted list is :
-45     -12     8       23      54      57      _
```

11:12 AM

EX. NO: 4(B) QUICK SORT

Aim:

Write C programs that implement Quick sort, to sort a given list of integers in ascending order

Description:

This method is invented by hoare, considered to be fast method to sort the elements. The method is also called partition exchange sorting. The method is based on divide and conquer technique. i.e., the entire list is divided into various partitions and sorting is applied again and again on the partition.

In this method the list is divided into two based on an element called pivot element. Usually the first element is considered to be the pivot element. Now move the pivot element to its correct position in the list. The elements to the left and pivot element are less that this while the elements to the right of pivot are greater than the pivot. The process is reapplied to each of these partitions till we got the sorted list of elements.

Algorithm:

/* Quick _ Sort */

Input: An integer *n* and a list of *n* elements stored in array elements *a*[0], . . . , *a*[*n* – 1]
Output: sorted array

```
1.Start
2. Read the size of the array _n‘
3. Read the elements of the array _ L‘
4. call  QUICK_SORT(L,n)

5. print array _L‘
6. Stop
```

QUICK_SORT (L, first, last)

```
/* L [first: last] is the unordered list of elements to be quick sorted. The call to the  to sort the list
L [1: n] would be Quick _ SORT (L, 1, n) */
if (first < last) then
{
        PARTITION (L, first, last, loc) ; /* partition the list into two sub lists at loc
        */ QUICK_SORT (L, first, loc-1); /* quick sort the sub list L[first, loc-1] */
        QUICK_SORT (L, loc+1, last); /* quick sort the sub list L[Loc+1, last] */
}
end QUICK_SORT.


PARTITION(L,last,loc)
/* L[first :last] is the list to be participated. Loc is the position where the pivot element finally
settles down */
Left=first
right=last+1;
pivot_elt=L[first]; /* set the pivot element to the first element in list L */
while(left<right) do
        repeat
                left=left+1;
                until L[left]>=pivot_elt;
                 repeat

                right=right-1;
                 until L[right]<=pivot_elt;
        if(left<right) then swap(L[left],L[right]);
              /* arrows face each other */
end
Loc=right
Swap(L[first],L[right]);
               /* arrows have crossed each other – exchange pivot element L[first] with L[right] */
end PARTITION
```

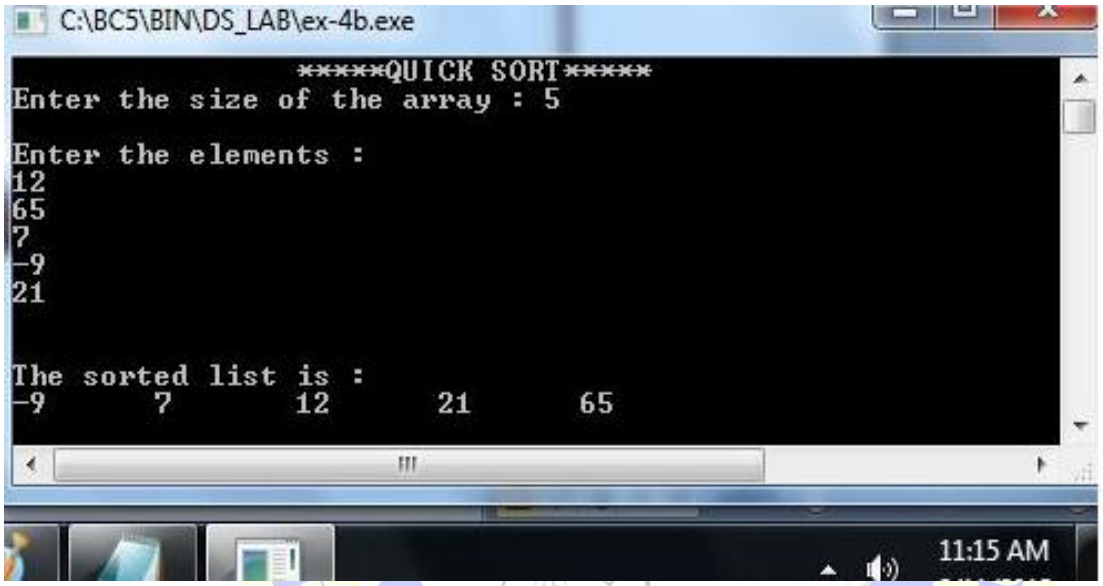
Sample Input:

Enter the array size: 3

Enter elements: 54 67 12

The sorted array is: 12 54 67

Observed Output:



```
C:\BC5\BIN\DS_LAB\ex-4b.exe
                        *****QUICK SORT*****
Enter the size of the array : 5

Enter the elements :
12
65
7
-9
21


The sorted list is :
-9      7       12      21      65
```

11:15 AM

EX. NO: 4(C) INSERTION SORT

Aim:

Write C programs that implement Insertion sort, to sort a given list of integers in ascending order

Description: Insertion sort is similar to playing cards. To sort the cards in your hand you extract a card shift the remaining cards and then insert the extracted card in its correct place. The efficiency of insertion sort is O(n^2).

Algorithm:

```
                    /* Insertion_ Sort */
Input: An integer n and a list of n elements stored in array elements a[0], . . . , a[n – 1]
Output: sorted array

1.Start
2. Read the size of the array _n‘
3. Read the elements of the array _ L‘
4. call  INSERTION _SORT (L,n)
5. print array _L‘
6. Stop0


 INSERTION_SORT (L, n)
/* L (1: n) is an unordered list of data elements to be sorted in the ascending order */
for i=2 to n do            /* n-1 passes*/
    Key = L[i];   /* key is the key to be inserted and position its location in the unordered list*/
    position = i;
        /* compare key with its sorted sub list of predecessors for insertion at the appropriate
    position */
    While((position > 1) and (L [position –1] > Key) do
        L[position] = L [position-1];
```

```
                Position = position -1;
                L[position] = Key;
        end
end
End INSERTION_SORT.
```

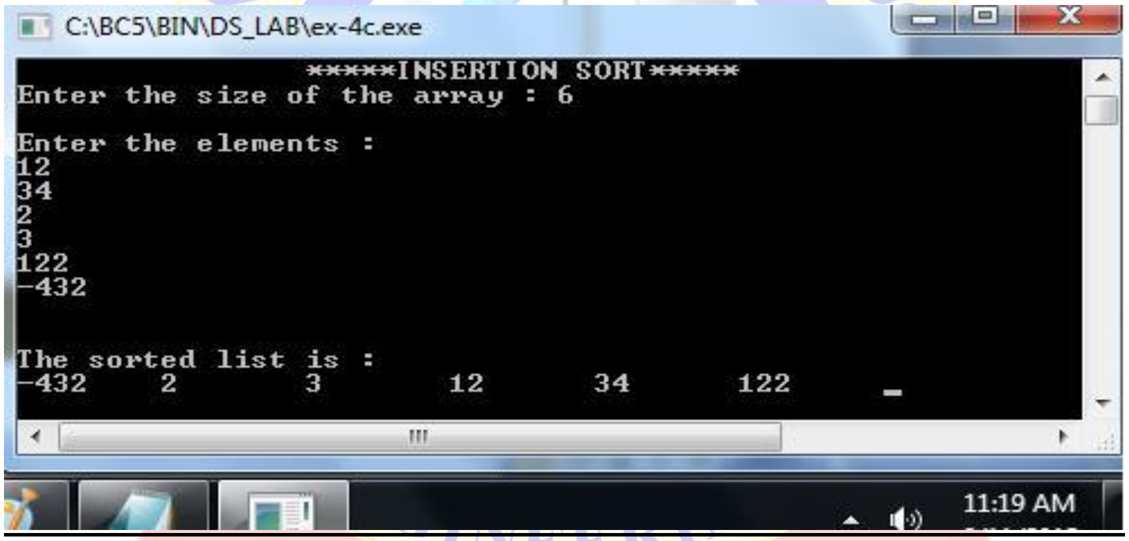
Sample Input:

Enter the range of array:5

Enter elements into array:56 23 34 12 8

The sorted order is: 8   12   23   34   56

Observed Output:



```
C:\BC5\BIN\DS_LAB\ex-4c.exe
                *****INSERTION SORT*****
Enter the size of the array : 6

Enter the elements :
12
34
2
3
122
-432


The sorted list is :
-432     2        3        12       34       122      _
```

Viva questions:

1. What is sorting.
2. In to types the sorting techniques are classified
3. Name some sorting algorithms
4. What is the best case of Bubble sort
5. What is the worst case of Bubble sort
6. What is the best case of insertion sort

8. What is the worst case of insertion sort

9. What is the average case of insertion sort

10. What are the best, average and worst case of quick sort. Ans:

11. What are the steps in quick sort algorithm

EX. NO: 5(A) HEAP SORT

Aim:

Write C programs that implement heap sort, to sort a given list of integers in ascending order

Description:

Heapsort is a comparison-based sorting algorithm to create a sorted array (or list), and is part of the selection sort family. Heapsort is a two step algorithm.

The first step is to build a heap out of data. The second step begins with removing the largest element from the heap . We insert the removed element into the sorted array. For the first element this would be the position of n-1 of the array. Next we reconstruct the heap and remove the next largest item, and insert it into the array. After we have removed all the objects from the heap, we have a sorted array. We can vary the direction of the sorted elements by choosing a min-heap or max-heap in step one.

Algorithm:

/* Heap_ Sort */

Input: An integer *n* and a list of *n* elements stored in array elements *a*[0], . . . , *a*[*n* – 1]
Output: sorted array

```
1.Start
2. Read the size of the array _n‘
3. Read the elements of the array _L‘
4. call  HEAP_SORT(L,n)
5. print array _L‘
6. Stop
 HEAP_SORT (L, n)
/* L [1: n] is the unordered list to be sorted. The output list is returned in L itself */
CONSTRUCT_HEAP (L, n);          /* construct the initial heap out of L[1:n] */
BUILD_TREE (L, n);              /* output root node and reconstruct heap */
end HEAP_SORT.
```

```
 BUILD_TREE (L, n)
for end _node _index = n to 2 step-1 do
{
        swap(L[1],L[end _node _index]; /* swap root node with the largest */
        RECONSTRUCT_HEAP (L, end_ node_ index); /* for reconstructing heap */
}
end BUILD_TREE.

 RECONSTRUCT_HEAP (L, end _node _index )
Heap = false ;
parent _ index = 1;
child _ index = parent _ index * 2;
while (not heap) and (child _ index < end _node _index) do
        right _child _index = child _ index + 1;
        if (right_ child_ index < end_ node_ index) then
                        /*choose which of the child nodes are greater than orequal to the parent /*
        if (L[right_ child _index] > end_ node_ index] ) then
                        child _index = right _child _index;
        if (L[child _ index]> L[parent_ index])then
        {

                swap (child_ index], L[parent_ index]) ;
                parent _index = child _index]
                Child _index =parent _index * 2;
        }
else heap = true;
 end
end RECONSTRUCT_HEAP.


 CONSTRUCT_HEAP (L, n)
                /* L[1 : n] is a list to be constructed into a heap */
        for child_ index = 2 to n do
    INSERT_HEAP (L, child_index);     /* insert elements one by one into the heap /*
end
end CONSTRUCT_HEAP.
```

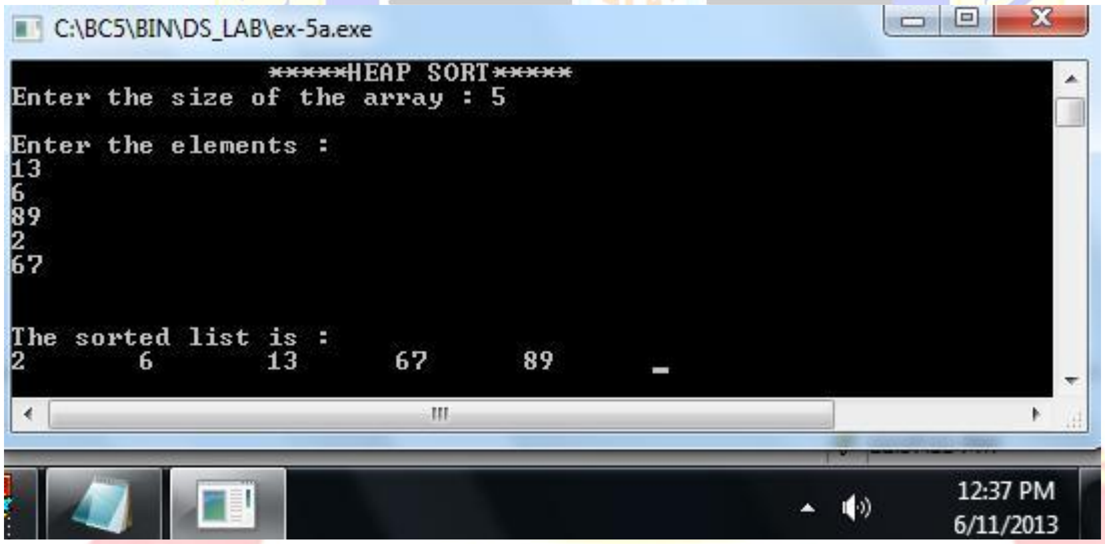
Sample Input:

Enter array size 7

Enter any 7 elements 23   12   1   4   2   45   3

After sorting, the element are   1   2   3   4 12 23 45

Observed Output:



```
                *****HEAP SORT*****
Enter the size of the array : 5

Enter the elements :
13
6
89
2
67


The sorted list is :
2       6       13      67      89      _
```

EX. NO: 5(B) RADIX SORT

Aim:

Write C programs that implement radix sort, to sort a given list of integers in ascending order

Description:

Radix sort is one of the linear sorting algorithms for integers. It functions by sorting the input numbers on each digit, for each of the digits in the numbers. Here the numbers are sorted on the least-significant digit first, followed by the second-least significant digit and so on till the most significant digit.

The Time Complexity of Radix sort is O(n).

Algorithm:

/* Radix_ Sort */

Input: An integer *n* and a list of *n* elements stored in array elements *a*[0], . . . , *a*[*n* – 1]

Output: sorted array

1.Start

2. Read the size of the array _n‘ and number of digits _d‘

3. Read the elements of the array _ L‘

4. call radixsort(L, n, 10, d)

5. print array _L‘

6. Stop

radixsort(L, n, r, d)

/* radix sort sorts a list L of n keys, each comprising d digits with radix r * */

Initialize each of the Q[0:r-1] linked queues representing the bins to be empty;

For i = d **to** 1 **step** -1 /* for each of the d passes over the list */

Sort the list L of n keys $K_i=k_1k_2k_3….k_d$ based on the digit i, inserting each of the keys K into the linked queue Q [k_i],

/* distribute the keys into Q [0 : (r-`)] based on the radix value of the digits */

Delete the keys from the queues Q [0: r-1] in order, and append the elements to the output list L;

end;

Return (L);

end radixsort.

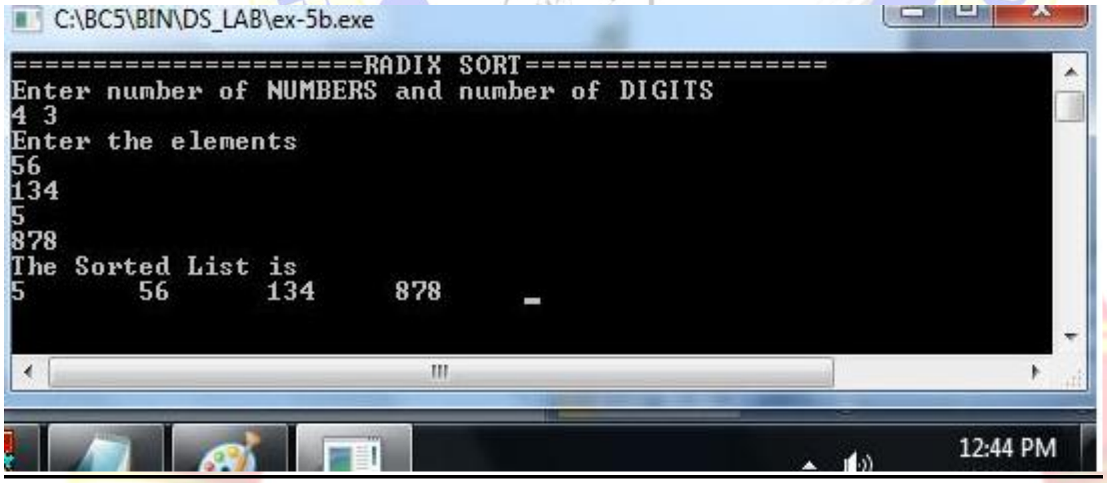
Sample Input:

Enter the number of elements: 6

Enter elements: 12 1 98 23 65 34

The Sorted array is 1 12 23 34 65 98

Observed Output:



```
C:\BC5\BIN\DS_LAB\ex-5b.exe
=====================RADIX SORT===================
Enter number of NUMBERS and number of DIGITS
4 3
Enter the elements
56
134
5
878
The Sorted List is
5       56      134     878
```

EX. NO: 5(C) MERGE SORT

Aim:

Write C programs that implement merge sort, to sort a given list of integers in ascending order

Description:

The sorting algorithm Merge sort produces a sorted sequence by sorting its two halves and merging them. With a time complexity of **O(nlog(n))** merge sort is optimal. Similar to the quick sort, the merge sort algorithm is based on a divide and conquer strategy first, the sequence to be sorted is decomposed into two halves **(Divide).** Each half is sorted independently **(Conquer).** Then the two sorted halves are merged to a sequence

Algorithm:

```
/* Merge_ Sort */
Input: An integer n and a list of n elements stored in array elements a[0], . . . , a[n – 1]
Output: sorted array

1.Start
2. Read the size of the array _n‘
3. Read the elements of the array _ L‘
4. call  MERGE _SORT(L,1,n)
5. print array _L‘
6. Stop

 Merge (x, first, mid, last)
/* x [first: mid] and x[mid+1:last] are ordered lists of data elements to be merged into a single
ordered list x[first : last] */

first1 = first;
 last1 =mid;
 first2 = mid +1;
last2 = last;     /* set the beginning and the ending indexes of the two lists into the appropriate
```

```
                        variables* /
i=first;        /* i is the index variable for the temporary output list tem */
        /* begin air wise comparisons of elements from the two lists */
While (first1<=last1) and (first2<=last2) do
                case: X[first1]<X[first2]:
                    {
                        temp[i]=X[first1];
                        first1= first1+1;
                        i=i+1;
                    }
                Case : X[first1]>X[first2]:
                    {
                        first2=first2+1;
                        i=i+2;        }



                Case : X[first1]=X[first2]:    temp[i]=X[first1];
                    {       temp[i+1]=X[ first2];
                                first1=first1+1;
                                first2=first2+1;
                                i=i+2;            }
                    end     /* end case */
                end     /* end while */

                    /* the first list gets exhausted */
                    while (first2<=last2) do
                    temp[i]=X[first2];
                    first2=first2+1;
        i=i+1;
end
                    /* the second list gets exhausted */
while (first1<=last1) do
```

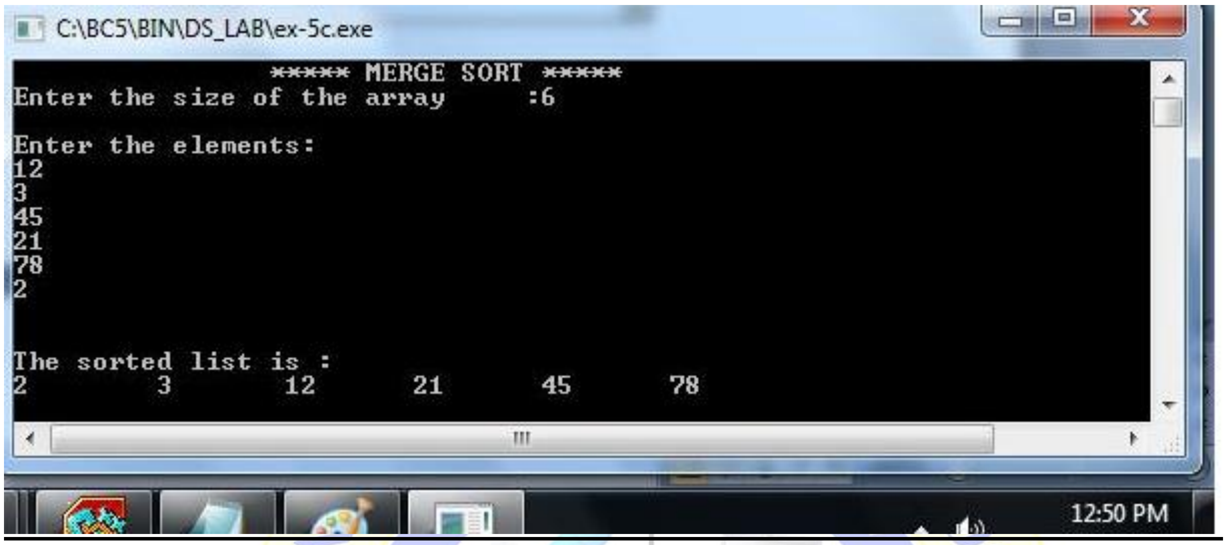
```
        temp[i]=X[first1];
        first1=first1+1;
        i=i+1;
end
                /* copy list temp to list x */
                for j=first to last do
        X[j]=temp[j]
end
end MERGE.
```

```
 MERGE_SORT(a, first, last)
/* a(first : last) is the unordered list of elements to be merge sorted .The call to the  to sort the
list a[1 : n] .would be MERGE_SORT(a, 1, n ) */
If (first < last) then
{
        mid=[(first + last)/2]; /* divide the list into two sub lists */
        MERGE_SORT (a, first, mid); /*merge sort the sub list a [first ,mid] */
        MERGE_SORT (a, mid+1, last); /* merge sort the sub list a [mid+1, last] */

        MERGE (a, first, mid, last); */ merge the two sub lists a [first, mid] and a[mid+1, last] */
}
end MERGE SORT.
```

Sample Input:

```
        Enter the size of array: 4
        Enter the elements: 15 4 23 2
        The sorted array is:  2          4        15       23
```

Observed Output:



```
C:\BC5\BIN\DS_LAB\ex-5c.exe
                    ***** MERGE SORT *****
Enter the size of the array      :6

Enter the elements:
12
3
45
21
78
2


The sorted list is :
2          3         12         21         45         78
```

Viva questions:

1. What is sorting?
2. Differentiate between sorting and searching.
3. What is a pass?
4. What is worst case and average case complexities of heap sort?
5. What are the best, worst case and average case complexities of radix sort?
6. What is best case complexity of merge sort?
7. What is worst case and average case complexities of merge sort?

EX. NO: 6(A) STACK USING ARRAY

Aim:

Write C programs that implement stack (its operations) using arrays

Description:

Stack is a linear data structure where it restricts operations to only one end. That end is called as "**TOP**". Stack works on the principle of **"last in first out" (LIFO)**. Operations to insert an element are stack is called **"PUSH".** And deleting an element from stack is called **"POP".**

Algorithm:

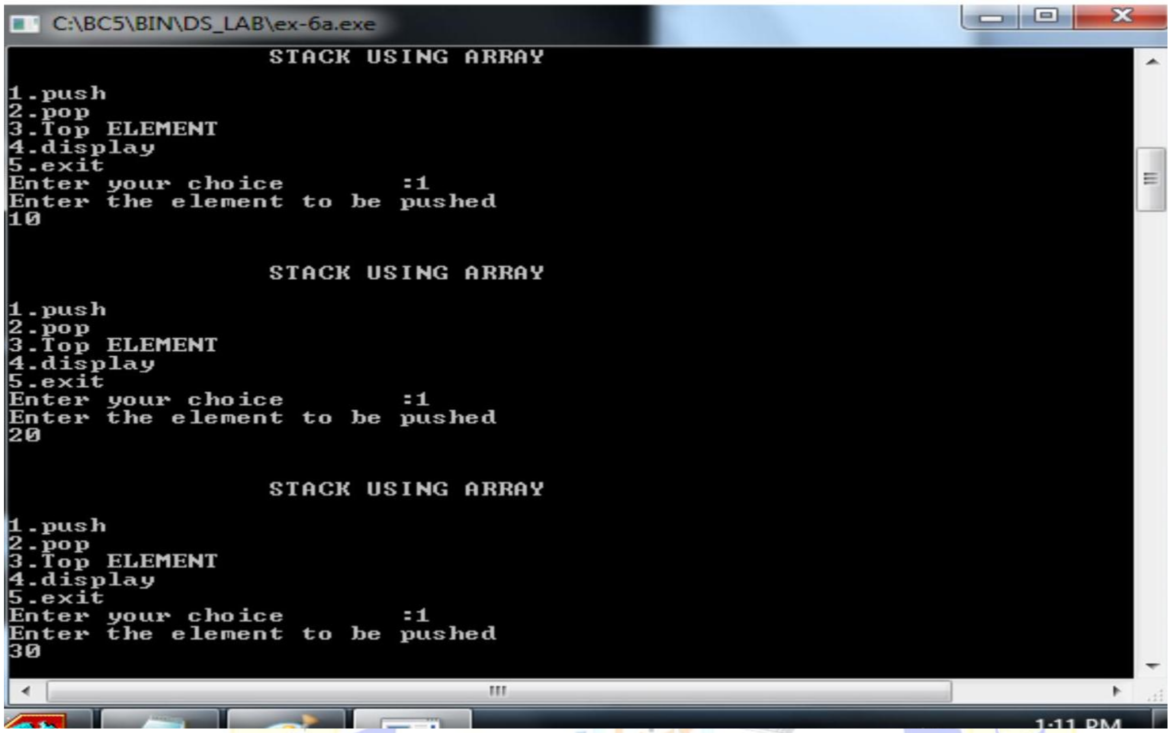
```
/*Implementation of push operation on a stack */
 PUSH(STACK, n, top, item)
if (top = n) then STACK_FULL;
else
{
    top = top + 1;
    STACK[top] = item; /* store item as top element of STACK */
}
end PUSH

/*Implementation of pop operation on a stack; */
 POP(STACK, top, item)
if (top = 0) then STACK_EMPTY;
else
{
         item = STACK[top];
        top = top - 1;
}
end POP
```

```
 Display() if top = -1
then
        print _ Stack is empty‘
else
{
        i=0;
        while(top>0) do
                print _stack[i];
                i++;
        end while.
}
end Display.
```

Sample Input:

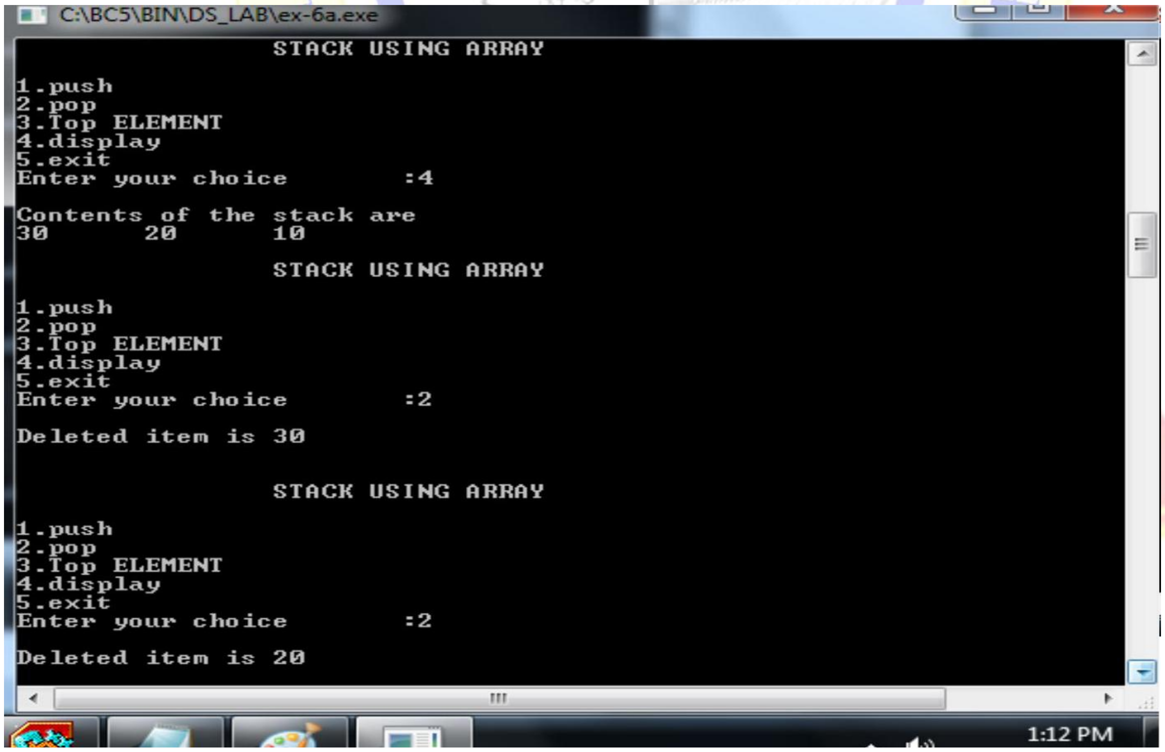
```
        Menu
           1.push
           2.pop
          Enter your choice 1
          Enter the element to insert 50
      Do you wish to continue press y for yes and n for no: y
    Menu
          1.push
          2.pop
          Enter your choice 1
          Enter the element to insert 20
       Do you wish to continue press y for yes and for no:n
        Stack elements are : 20 50
```

Observed Output:



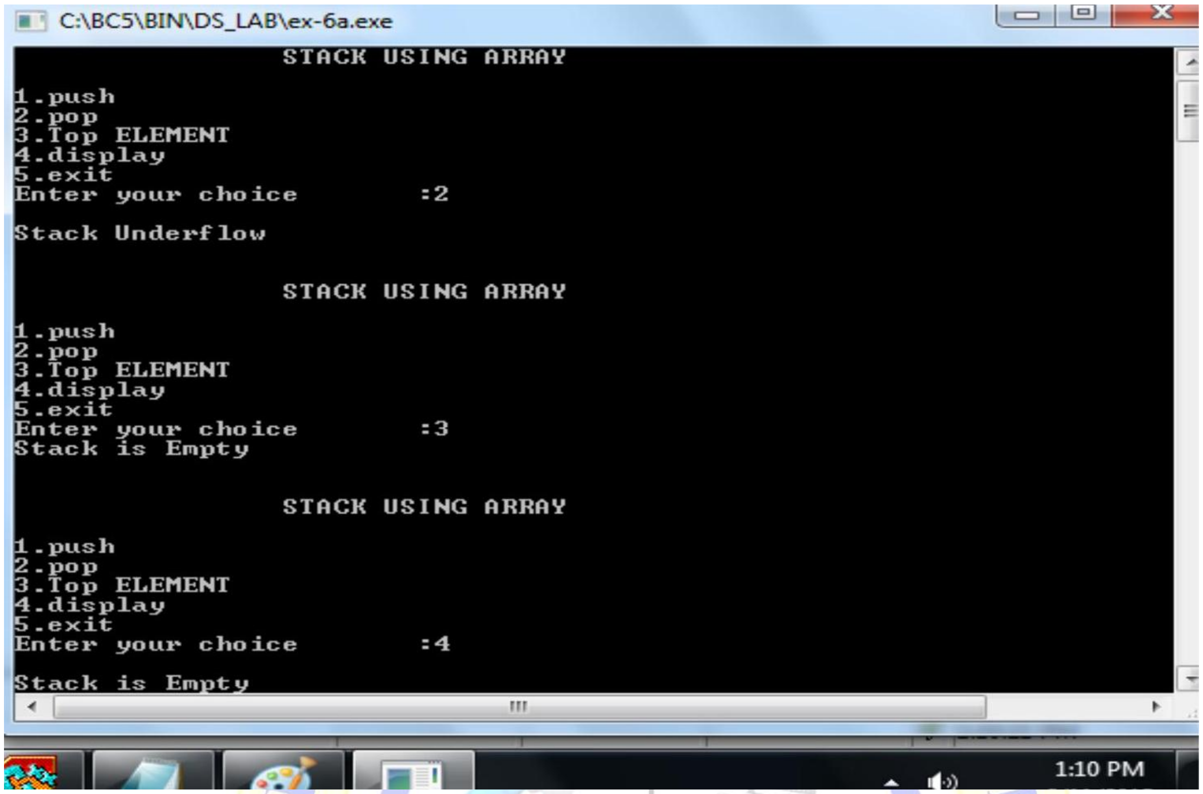
C:\BC5\BIN\DS_LAB\ex-6a.exe

```
                        STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice          :1
Enter the element to be pushed
10


                        STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice          :1
Enter the element to be pushed
20


                        STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice          :1
Enter the element to be pushed
30
```



C:\BC5\BIN\DS_LAB\ex-6a.exe

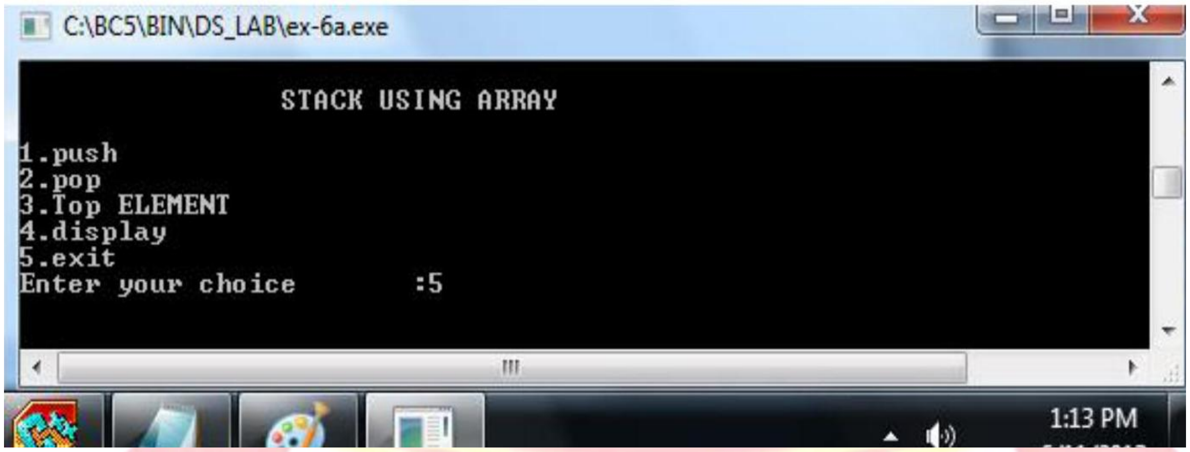
```
                        STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice          :4

Contents of the stack are
30      20      10
                        STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice          :2

Deleted item is 30


                        STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice          :2

Deleted item is 20
```



C:\BC5\BIN\DS_LAB\ex-6a.exe

```
                    STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :2

Stack Underflow


                    STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :3
Stack is Empty


                    STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :4

Stack is Empty
```

1:10 PM



C:\BC5\BIN\DS_LAB\ex-6a.exe

```
                    STACK USING ARRAY

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :5
```

1:13 PM

EX. NO: 6(B) STACK USING LINKED LISTS

Aim:

Write C programs that implement stack (its operations) using Linked list

Description:

The major problem with the stack using array is, it works only for fixed amount of numbers of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of the data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means the stack implemented using linked list can work for variable size of data. So there is no need to fix the size at the beginning of implementation.

Algorithm:

Algorithm: Push item ITEM into a linked stack S with top pointer TOP

```
PUSH_LINKSTACK (TOP, ITEM)
/* Insert ITEM into stack */
Call GETNODE(X)
DATA(X) = ITEM   /*frame node for ITEM */
LINK(X) = TOP   /* insert node X into stack */
TOP = X         /* reset TOP pointer */
end PUSH_LINKSTACK.
```

Algorithm: Pop from a linked stack S and output the element through ITEM

```
POP_LINKSTACK(TOP, ITEM)
  /* pop element from stack and set ITEM to the element */
if (TOP = 0) then call LINKSTACK_EMPTY
```

```
        /* check if linked stack is empty */
    else {
            TEMP = TOP
              ITEM = DATA(TOP)
            TOP = LINK(TOP)
            }

    call RETURN(TEMP) ;
    **end** POP_LINKSTACK.
```

Algorithm for display

```
 Display()
ptr ← top
ifptr=NULL then
        print _ Stack is empty‘
else
{
        While(ptr!=NULL) do
                print _ptr-> item‘
                ptr=ptr-> next
        end while.
}
end Display.
```

Sample Input:

```
    Linked stack
      1.Push
      2.Pop
      3.Display
     Enter your choice 1
```

Enter the number 18

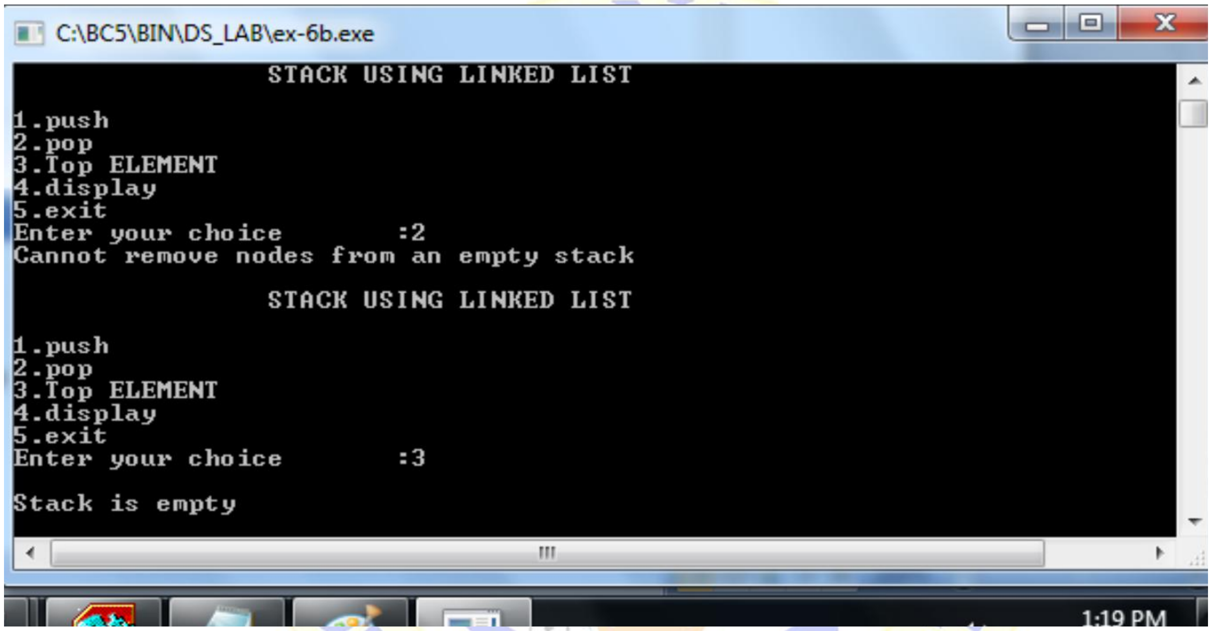
Linked stack

1. Push
2. Pop
3. Display

Enter your choice 3

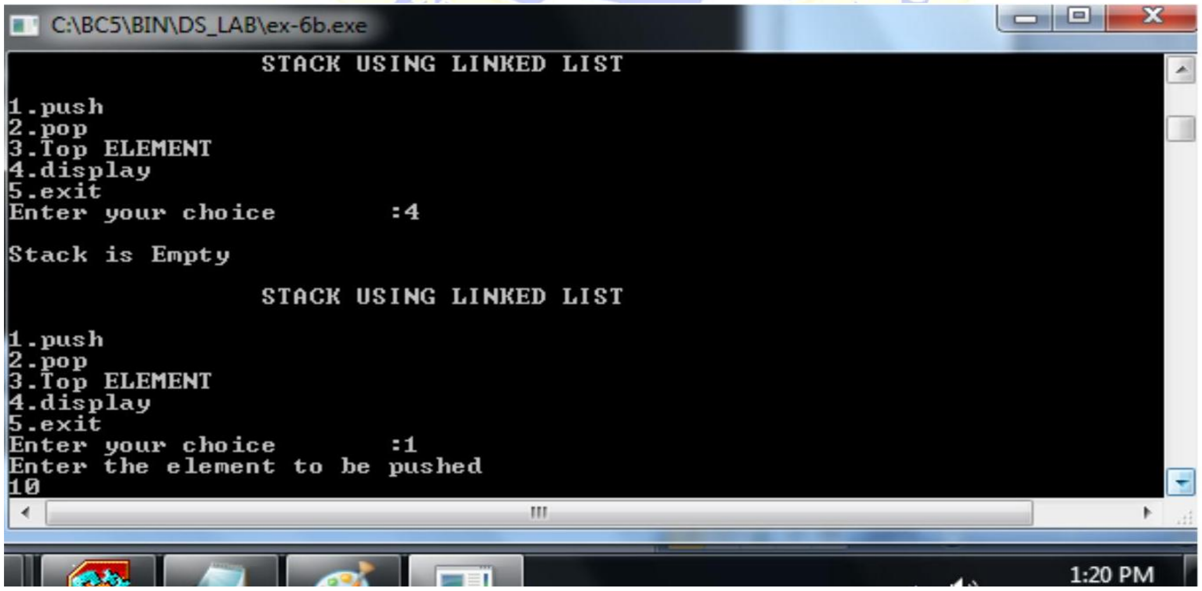
Stack elements: 18

Observed Output:



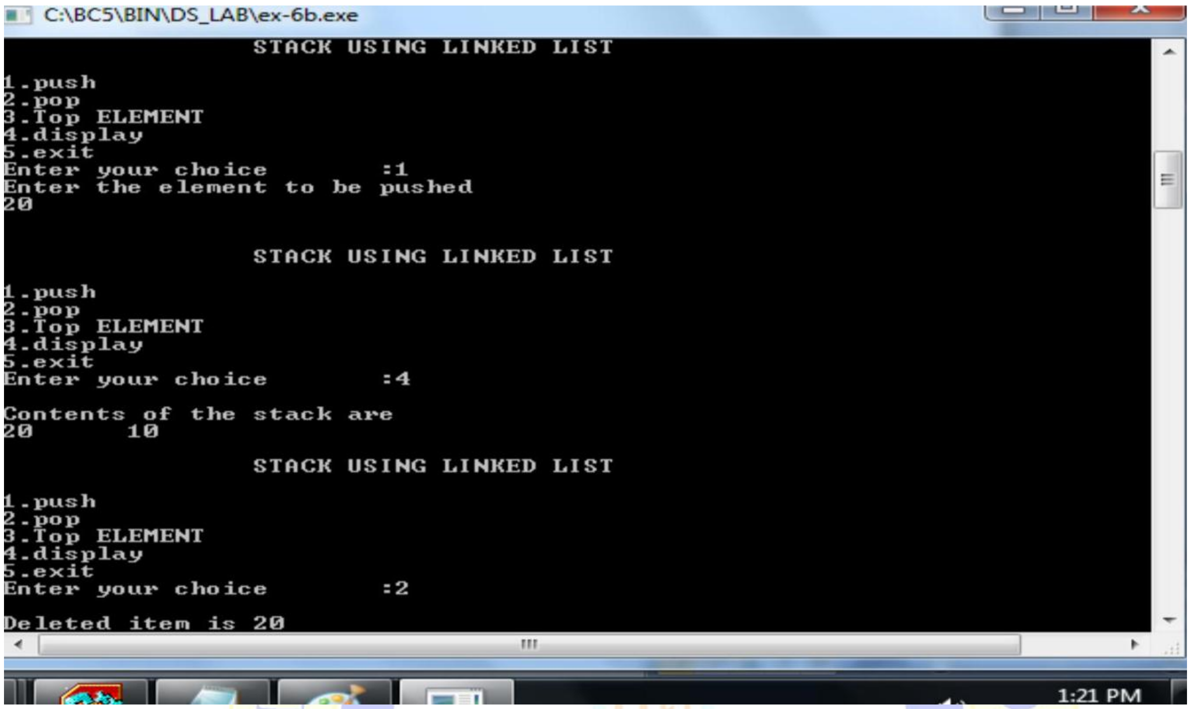
C:\BC5\BIN\DS_LAB\ex-6b.exe

```
                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :2
Cannot remove nodes from an empty stack

                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :3

Stack is empty
```

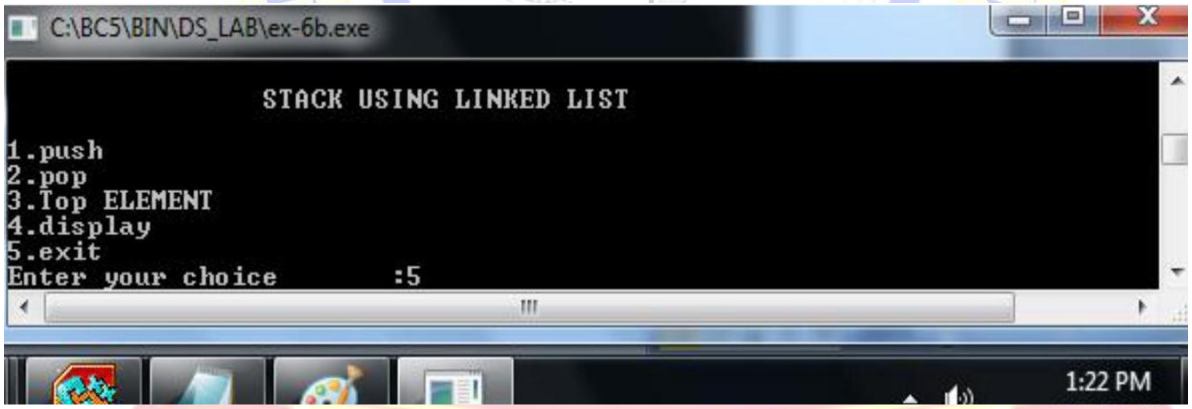


C:\BC5\BIN\DS_LAB\ex-6b.exe

```
                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :4

Stack is Empty

                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :1
Enter the element to be pushed
10
```



```
                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :1
Enter the element to be pushed
20


                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :4

Contents of the stack are
20       10
                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :2

Deleted item is 20
```



```
                    STACK USING LINKED LIST

1.push
2.pop
3.Top ELEMENT
4.display
5.exit
Enter your choice        :5
```

Viva questions:

1. What is stack?
2. What is the difference between a Stack and an Array?.
3. Give real time and system examples of stack?
4. What is meant by push and pop?
5. When overflow will occur in stack?

6.. What is the significance of top pointer?

7. State different applications of stack.

EX. NO: 7(A) CONVERT INFIX EXPRESSION INTO POSTFIX EXPRESSION

Aim:

Write a C program that uses Stack operations to convert infix expression into postfix expression

Description:

In normal algebra we use the Infix notation like a+b*c. The corresponding Postfix expression will look like abc*+.

In order to define the program, we will assume the following functions

- ReadSymbol(): From given Infix expression, this will read the next symbol
- ISP(X): Returns the in-stack priority for a symbol of X
- ICP(X): This function returns the in-coming priority value for a symbol X
- OUTPUT(X): Append the symbol into the resultant expression.

Algorithm:

1. Push ―(‖ onto stack, and add―)‖ to the end of P.
2. Scan P from left to right and repeat Steps 3 to 6 for each element of P until the stack is empty.
3. If an operand is encountered, add it to Q.
4. If a left parenthesis is encountered, push it onto stack.
5. If an operator ⊗ is encountered, then:
(*a*) Repeatedly pop from stack and add P each operator (on the top of stack), which has the same precedence as, or higher precedence than ⊗.
(*b*) Add ⊗ to stack.
6. If a right parenthesis is encountered, then:

(*a*) Repeatedly pop from stack and add to P (on the top of stack until a left parenthesis is encountered.
(*b*) Remove the left parenthesis. [Do not add the left parenthesis to P.]
7. Exit.

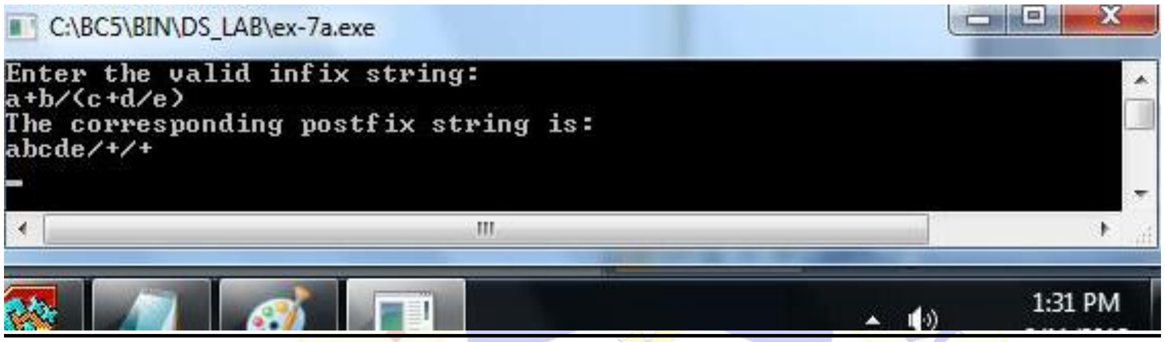
Sample Input:

Read the infix expression a+b*c-d

Given infix expression: a+b*c-d

Postfix expression: abc*c+d-

Observed Output:



```
Enter the valid infix string:
a+b/(c+d/e)
The corresponding postfix string is:
abcde/+/+
_
```

EX. NO: 7(B) QUEUE USING ARRAYS

Aim:

Write C programs that implement Queue (its operations) using arrays.

Description:

Queue is a linear data structure where operations are done at two ends. The end where element is intersected is called as “REAR”. The end where element is deleted is called as “FRONT”. Queue works on the principle of “FIRST IN FIRST OUT”. Operations to insert an element in queue are called ENQUEUE. And deleting an element from queue is called “DEQUEUE”

Algorithm:

```
/*Implementation of an insert operation on a queue */
 INSERTQ (Q, n, ITEM, REAR)
/* insert item ITEM into Q with capacity n */
if (REAR = n) then QUEUE_FULL;
REAR = REAR + 1;      /* Increment REAR*/
Q[REAR] = ITEM;     /* Insert ITEM as the rear element*/
end INSERTQ
```

```
/*Implementation of a delete operation on a queue */
 DELETEQ (Q, FRONT, REAR, ITEM )
if (FRONT =REAR) then QUEUE_EMPTY;
FRONT = FRONT +1;
ITEM = Q[FRONT];
end DELETEQ.
```

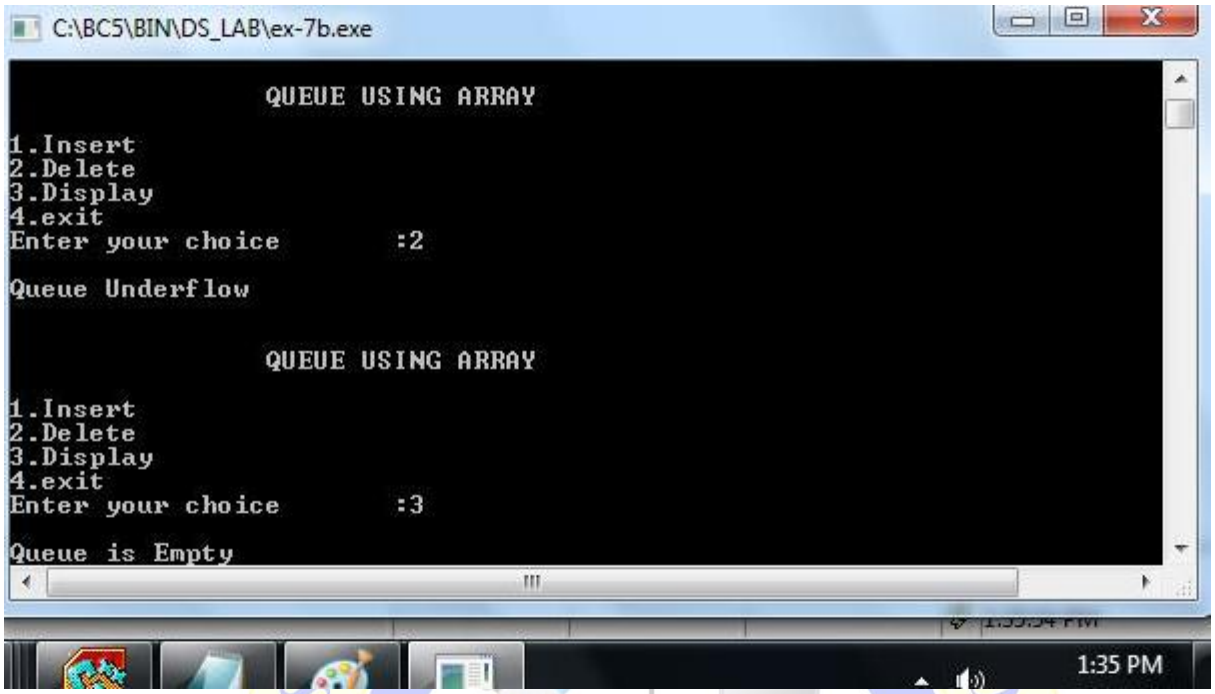
/*Implementation of Display*/

```
 Display() if front = -
1 then
        print _Queue is empty‘
else
{
        i=front+1;
        while(i< rear) do
        print queue[i]
                i++;
        end while
end Display.
```

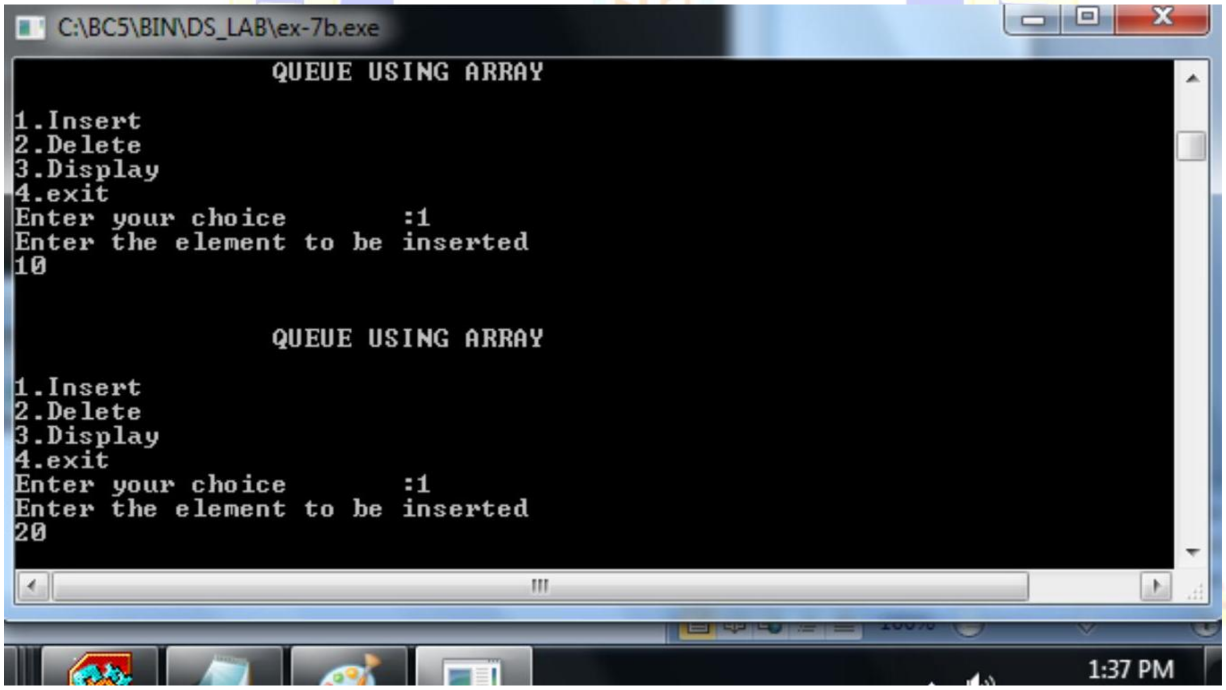
Sample Input:

```
   Menu
    1. Insert
    2. Delete
      Enter your choice 1
      Enter the element to insert 23
   Do you wish to continue press Y for yes n for no: Y
   Menu
    1. Insert
    2. Delete
      Enter your choice 1
      Enter the element to insert 18
   Do you wish to continue press Y for yes n for no: Y
  Menu
    1. Insert
    2. Delete
      Enter your choice 2
      Element deleted is 23
   Do you wish to continue press Y for yes n for no: N
The element is queue are 18
```

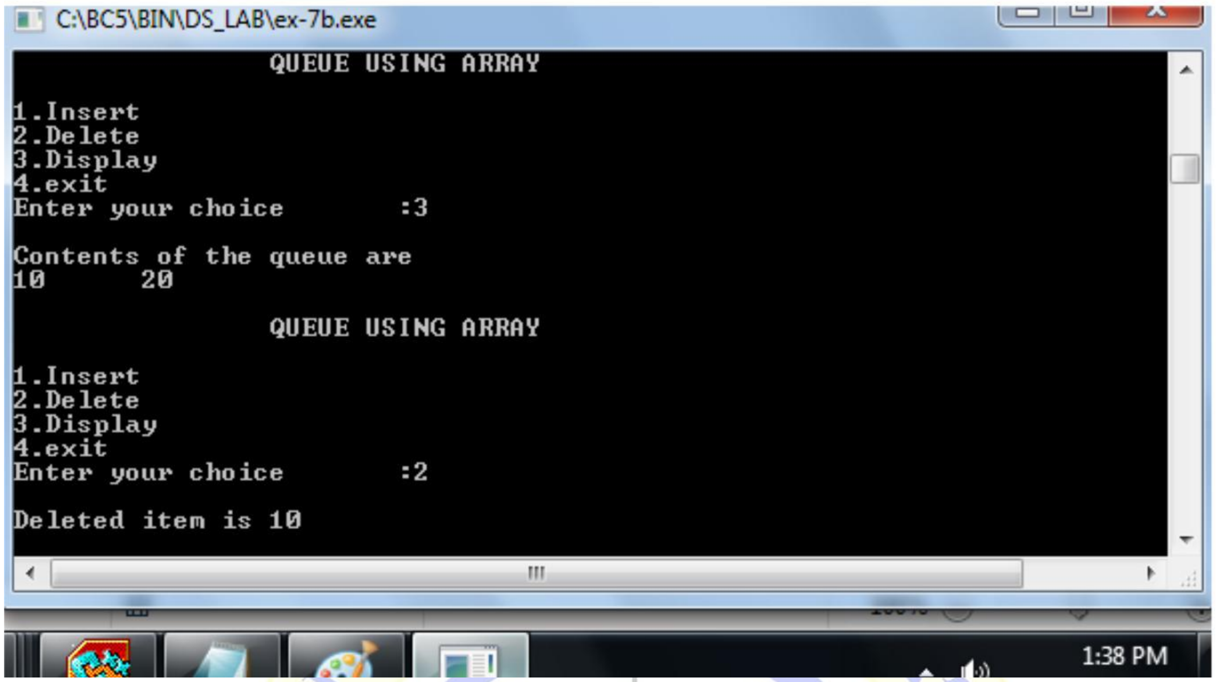
Observed Output:



```
                QUEUE USING ARRAY

1.Insert
2.Delete
3.Display
4.exit
Enter your choice       :2

Queue Underflow


                QUEUE USING ARRAY

1.Insert
2.Delete
3.Display
4.exit
Enter your choice       :3

Queue is Empty
```



```
                QUEUE USING ARRAY

1.Insert
2.Delete
3.Display
4.exit
Enter your choice       :1
Enter the element to be inserted
10


                QUEUE USING ARRAY

1.Insert
2.Delete
3.Display
4.exit
Enter your choice       :1
Enter the element to be inserted
20
```



```
                    QUEUE USING ARRAY

1.Insert
2.Delete
3.Display
4.exit
Enter your choice        :3

Contents of the queue are
10      20
                    QUEUE USING ARRAY

1.Insert
2.Delete
3.Display
4.exit
Enter your choice        :2

Deleted item is 10
```

EX. NO: 7(C) QUEUE USING LINKED LISTS

Aim:

Write C programs that implement Queue (its operations) using linked lists

Descritpion:

Queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position, known as enqueue, and removal of entities from the front terminal position, known as dequeue. This makes the queue a First-In-First-Out (FIFO) data structure. Linked list is a data structure consisting of a group of nodes which together represent a sequence. Here we need to apply the application of linkedlist to perform basic operations of queue.

Algorithm:

Algorithm: Push item ITEM into a linear queue Q with FRONT and REAR as the front and rear pointer to the queue

```
 INSERT_LINKQUEUE(FRONT,REAR,ITEM) Call
GETNODE(X);
DATA(X)= ITEM;
LINK(X)= NIL; /* Node with ITEM is ready to be inserted into Q */
if (FRONT = 0) then FRONT = REAR = X;
        /* If Q is empty then ITEM is the first element in the queue Q */
else {LINK(REAR) = X;
    REAR = X
   }
end INSERT_LINKQUEUE.
```

Algorithm: Delete element from the linked queue Q through ITEM with FRONT and REAR as the front and rear pointers

```
 DELETE_LINKQUEUE (FRONT,ITEM)
if (FRONT = 0) then call LINKQUEUE_EMPTY;
```

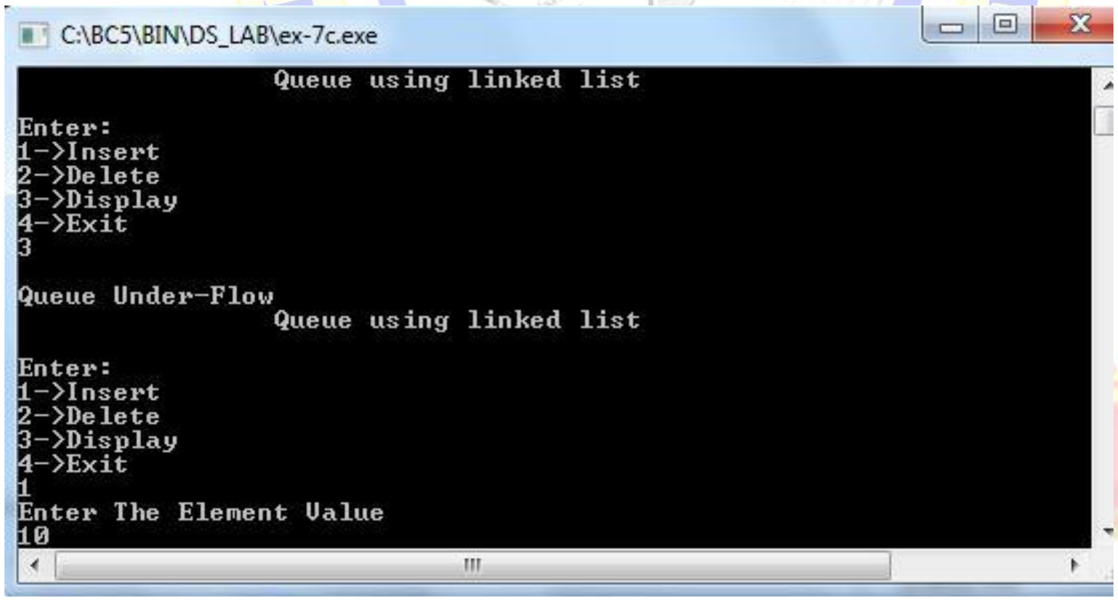
```
/* Test condition to avoid deletion in an empty queue */
else            {TEMP = FRONT;
                 ITEM = DATA (TEMP);
                 FRONT = LINK (TEMP);
                }
call RETURN (TEMP);  /* return the node TEMP to the free pool */
end DELETE_LINKQUEUE.
```

```
Algorithm: Display the Queue
 display()
 Temp=front
if(temp = = NULL)
        print —queue is empty‘
else
        while (temp->next!=NULL) do
                print temp->item
        temp=temp->next
        end while
end if.
end display.
```

Sample Input:

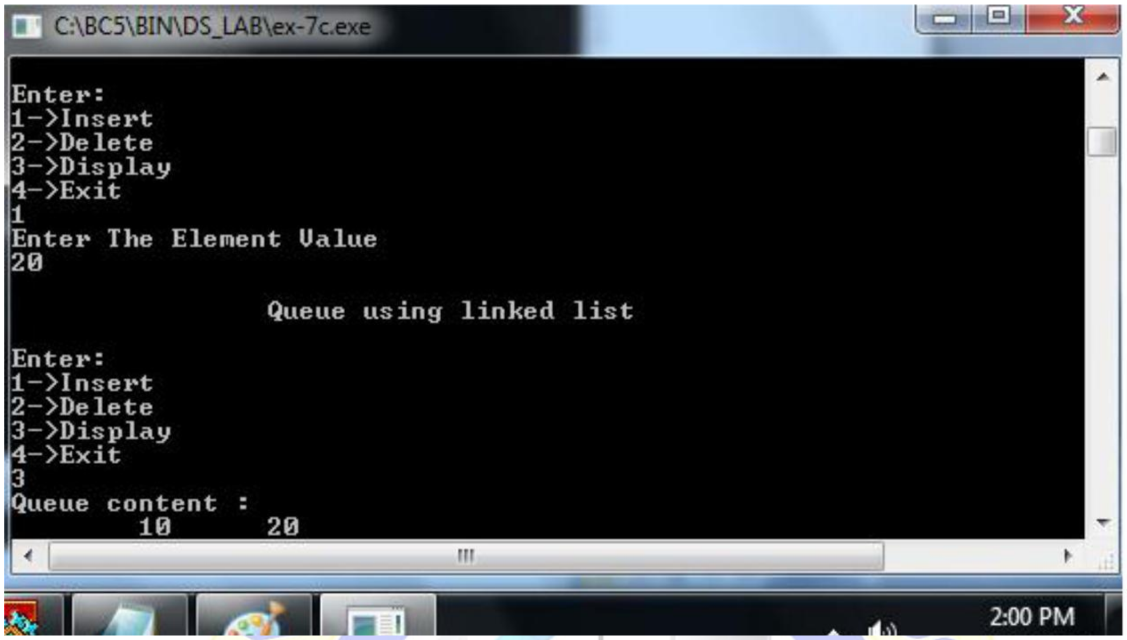
```
    Linked queue
     1. Enqueue
     2. Dequeue
     3. Display
    Enter your choice: 1
    Enter the item 48
  Linked queue
     1. Enqueue
     2. Dequeue
     3. Display
    Enter your choice: 1
    Enter the item 8
Linked queue
     1. Enqueue
     2. Dequeue
     3. Display
    Enter your choice: 2
    Deleted element is 48
Linked queue
     1. Enqueue
     2. Dequeue
     3. Display
    Enter your choice 3
    The elements are 8
```

Observed Output:

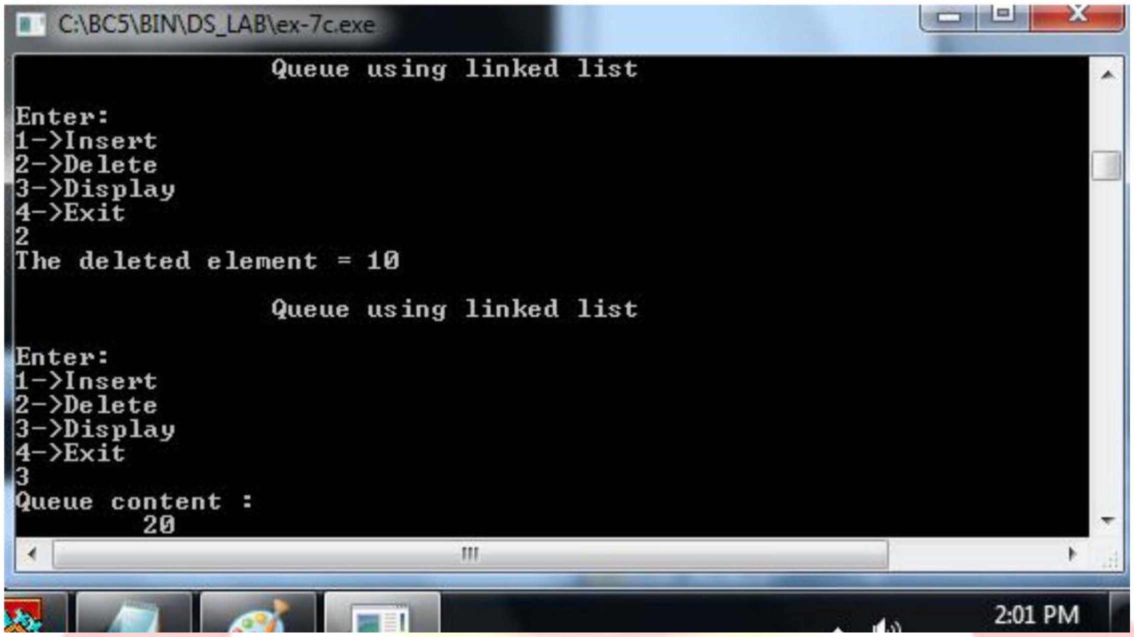


C:\BC5\BIN\DS_LAB\ex-7c.exe

```
                    Queue using linked list

Enter:
1->Insert
2->Delete
3->Display
4->Exit
3

Queue Under-Flow
                    Queue using linked list

Enter:
1->Insert
2->Delete
3->Display
4->Exit
1
Enter The Element Value
10
```



```
Enter:
1->Insert
2->Delete
3->Display
4->Exit
1
Enter The Element Value
20

                Queue using linked list

Enter:
1->Insert
2->Delete
3->Display
4->Exit
3
Queue content :
        10      20
```



```
                Queue using linked list

Enter:
1->Insert
2->Delete
3->Display
4->Exit
2
The deleted element = 10

                Queue using linked list

Enter:
1->Insert
2->Delete
3->Display
4->Exit
3
Queue content :
        20
```

Viva questions:

1. What is a linear queue?
2. What is rear and front pointer?
3. Give real time and system example of queue?
4. Give array representation of queue?
5. What happened if Rear=MAXSIZE?
6. What is meant by binary expression?
7. What is meant by prefix expression?
8. What is meant by infix expression?
9. What is meant by postfix expression?
10. Convert the following infix expression into postfix expression A+(B*C–(D/E^F)*G)*H
11. What is a priority queue?
12. What are the disadvantages of sequential storage?
13. What are the disadvantages of representing a stack or queue by a linked list?

EX. NO: 8 SINGLE LINKED LIST

Aim:

a) Write a C program that uses functions to create a singly linked list

b) Write a C program that uses functions to perform insertion operation on a singly linked list

c) Write a C program that uses functions to perform deletion operation on a singly linked list

Description:

Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data types, including stacks, queues, associative arrays, and symbolic expressions, The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal.

On the other hand, simple linked lists by themselves do not allow random access to the data, or any form of efficient indexing. Thus, many basic operations — such as obtaining the last node of the list (assuming that the last node is not maintained as separate node reference in the list structure), or finding a node that contains a given datum, or locating the place where a new node should be inserted — may require scanning most or all of the list elements.

Algorithm:

```
                              /* Singly_linkedlist */
declare struct linkedlist
        integer data
        struct linkedlist *next
typedef struct linkedlist node; Step 1:
Start
Step 2:head =NULL
Step3: repeat steps from 3 to 12 if ch!=9
```

```
        Diplay the Menu and Read the value of ch
        1. Append     2.Display All               3.Insert after a specified node
        4. Insert before a specified node         5.Delete a node after a specific node
        6. Search a node in the list              7.Count the nodes in the list
        8. Distroy the list                       9.Exit program
Step 4:If ch=1 call append(&head) function
Step 5:If ch=2 call display(head) function
Step 6:If ch=3 call insert_after(&head) function
Step 7:If ch=4 call insert_before(&head) function
Step 8: if ch=5 call del_after(&head) function
Step 9: if ch=6 call search(head) function
Step 10: if ch=7 call count(head) function
Step11: if ch=8 call destroy(&head) function
Step12: if ch=9 call exit function
Step 13:Stop
```

Algorithm

```
createnode()
 //input: nothing
//output: assign a node null and returns that node after allocating it
memorydynamically.
Step 1: Allocate the memory for new node
Step 2: Read value of item
Step 3:Assign values to NEW node data part
Step 4:Assign NEW node address part as null
Step5: return NEW node
```

Algorithm append (node **head)

//Input: address of the head node which inturn has the address of first node of the list.

//output: adds a node to end of the list.

```
Step1: NEW = createnode()
Step2: if head = NULL then do
Step3 else goto step4 Step3: head = NEW and return
Step4: temp = head
Step5: repeat
Step 6  : until temp=next!=NULL
Step7: temp = temp → next
Step8: temp → next = NEW
Step9: stop
```

Algorithm display(node *p)

```
//input: address of the head node
//output: displays the content of the linked list
Step1: print _Contents of the List‘
Srep2: if (p=NULL)  then print _List is empty‘ and return
Step3: repeat steps from 4 to 5 until p!=NULL
Step4: print p->data
Step5: p = p->next;
Step6: stop
```

Algorithm insert_before(node **h) //input:

```
address of the head node
//output: inserts a node before a node containing specific data. Step1.
temp =((*h)->next)
Step2. prev = (*h)
Step3. while temp!= null and temp-> data !=k) do
Step4. prev = temp
Step5. temp =(temp->next)
Step6. end while
Step7. if temp!= null then do
Step8. new =createnode()
Step9. (new  next)temp;
```

```
Step10.(prev next) new;
Step11 End.   Insert_before.
```

Algorithm for insert_before()

```
Step1: if(*h==NULL) then return
Step2: read _k‘ the data of node before which node
Step3: if((*h)->data == k) then goto step4 else goto step7
Step4: NEW = createnode()
Step5: NEW->next = *h
Step6: *h = NEW and return
Step7: temp = (*h) next; prev = *h;
Step8: repeat step 9 to 10 until temp!=NULL && temp->data!=k
Step9: prev=temp
Step10: temp=temp->next
Step11: if(temp!=NULL) then goto step12 else goto step15
Step12: NEW = createnode()
Step13: NEW->next = temp
Step14: prev->next = NEW
Step15: stop
```

Algoritm insert _after(node **h)

```
//input : address of the header node
//output : inserts a node after a node containing specific data.
Step1. Start
Step2. Declarenode * temp, * new
Step3. If *h =null then do
          return
Step4. Read k
Step5. Temp = *h
Step6. While temp!=null and temp ->data!=k do
```

```
Step 7. Temp =(temp->next);
Step8. End while.
Step9. If temp != null then do
Step10.New=createnode()
Step11.(new->next)=(temp->next)
Step12.(temp->next)= new
Step13.End insert_after
```

Algorithm delete_after(node **h)

```
//input: address of first node
//output: node is deleted after a node containing specific data.
Step1. Start
Step2. Declare node *temp, *p
Step3. *p  null
Step4. If *h=null then do
Step5. Return
Step6. Read k
Step7. Temp =*h
Step8. While temp != null temp -> data!= k do
Step9. Temp =(temp->next)
Step10.End while.
Step11.If temp != null then do
Step12.*(temp->next)
Step13.(temp ->next) = (p-> next)
Step14.Print —deleted node‖
Step15.Free(p);
Step16.End.   Delete _after.
```

Algorithm destroy (node **h)

```
//input: address of the head node
//output: destroy the linked list
Step1. Start
```

```
Step2. Declare node *p;
Step3. If  *h != null then do
        Return
Step4. While (*h != null) then do
Step 5. P=(*h)->next
Step 6. Free(*h);
Step 7. *h =p;
Step 8. End while
Step 9. End destroy.
```

Algorithm count (node **h)

```
//input: address of the head node
//output: returns the count of nodes
Step 1. Start
Step 2. I=0
Step3. While h!= null do
Step4. H=(h->next)
Step5  I=i+1
Step 6 end while
Step 7 return i
Step8  count.
```

Algorithm for insert_before()

```
Step1: if(*h==NULL) then return
Step2: read the data of node before which node
Step3: if((*h)=data == k) then goto step4 else goto step7
Step4: NEW = createnode()
Step5: NEW=next = *h
Step6: *h = NEW and return
Step7: temp = (*h)->next; prev = *h;
Step8: repeat step 9 to 10 until temp!=NULL && temp->data!=k
```

```
Step9: prev=temp
Step10: temp=temp->next
Step11: if(temp!=NULL) then goto step12 else goto step15
Step12: NEW = createnode()
Step13: NEW->next = temp
Step14: prev->next = NEW
Step15: stop
```

Sample Input:

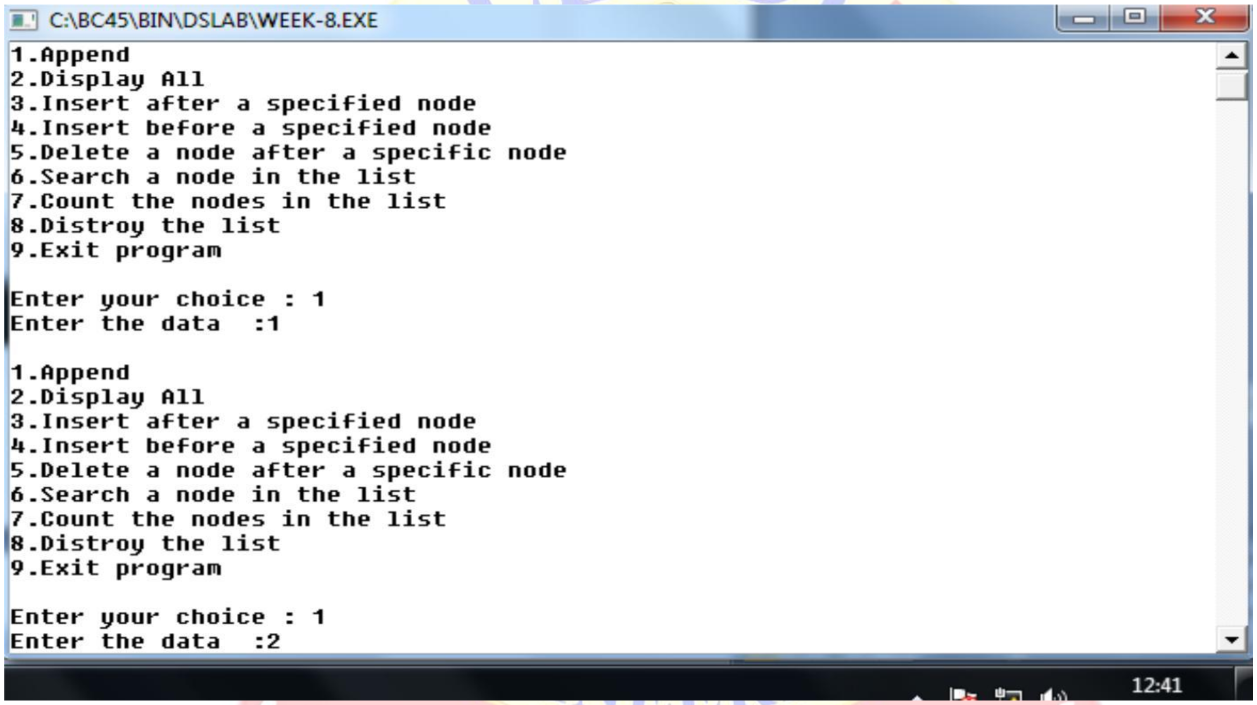
```
    1.Add at beginning
    2.Add at location
    3. Add at end
    4.Deletion
    5.Display
    6. exit
          Enter your choice : 1
          Enter the value : 33
     1.   Add at beginning
     2.   Add at location
     3.   Add at end
     4.   Deletion
     5.   Exit
            Enter your choice : 1
            Enter the value : 38
       Add at beginning
      1.   Add at location
      2.   Add at end
      3.   Deletion
      4.   Display
      5.   Exit
        Enter your choice : 1
        Enter the value : 40
```

1. Add at beginning
2. Add at location
3. Add at end
4. Deletion
5. Display
6. Exit

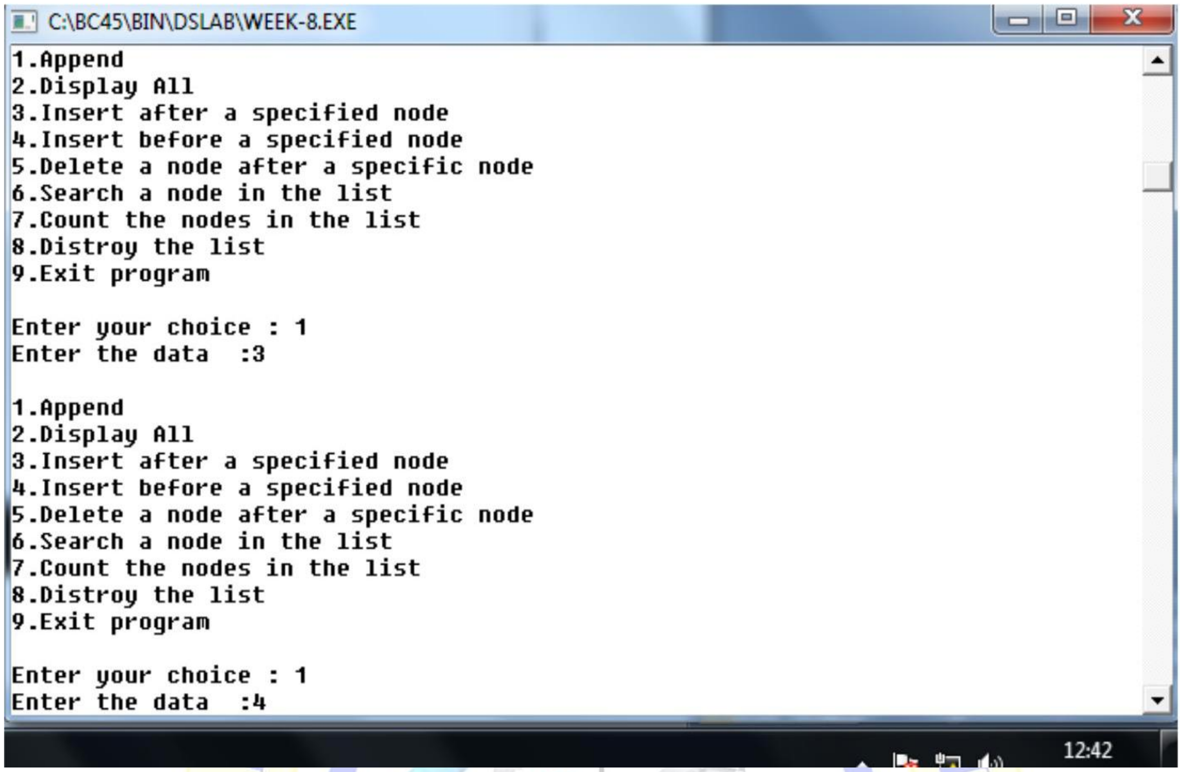
Enter your choice : 5

The elements in single linked list: 33 38 40

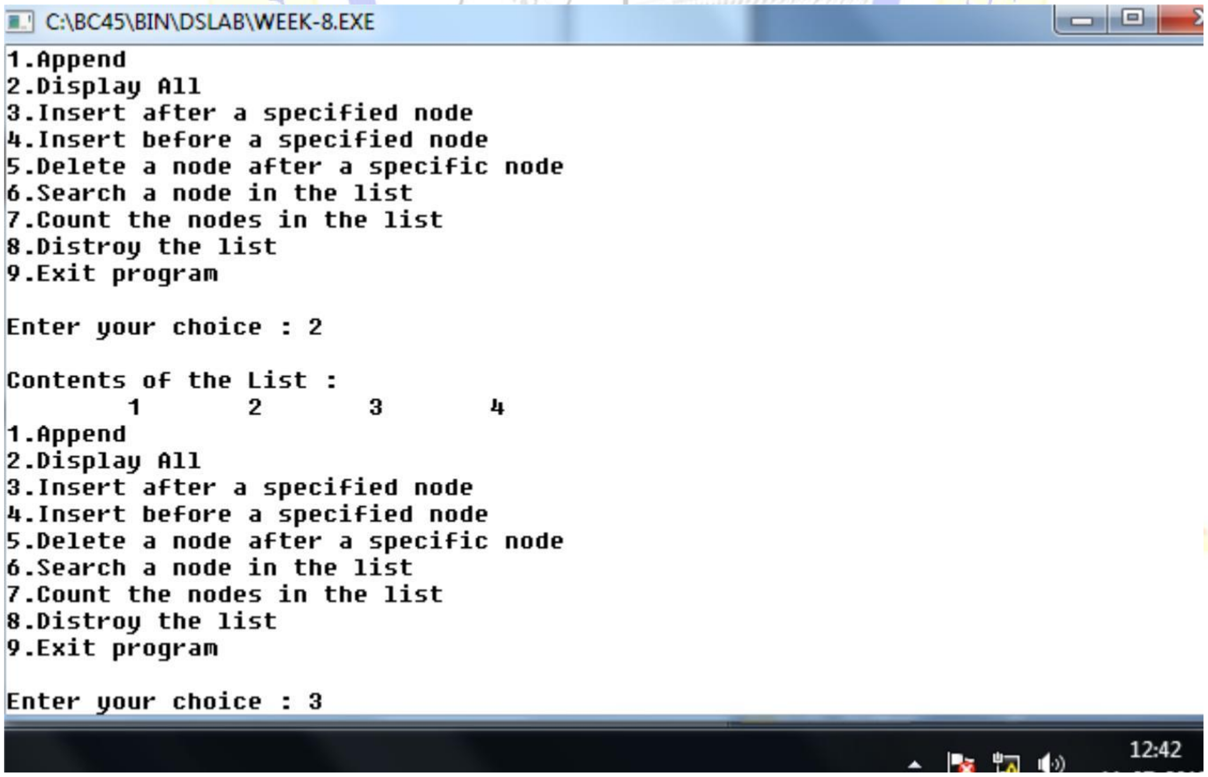
Observed Output:



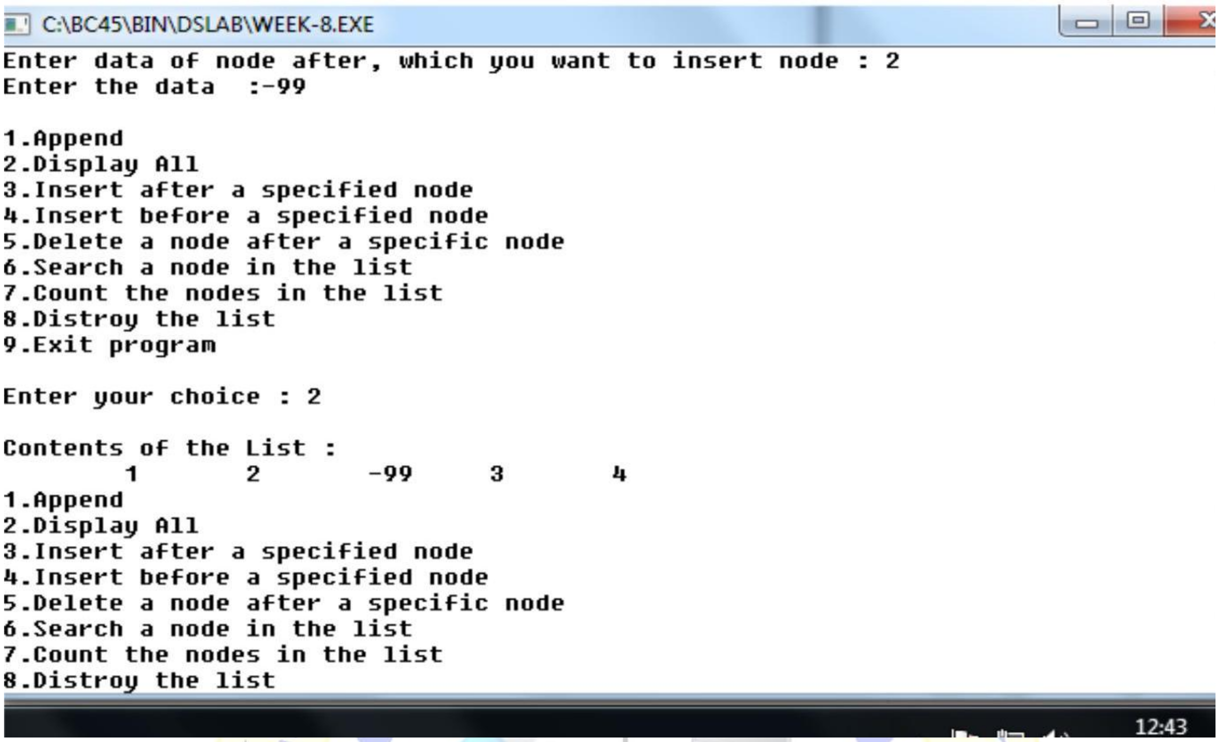
```
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 1
Enter the data  :1

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 1
Enter the data  :2
```


C:\BC45\BIN\DSLAB\WEEK-8.EXE

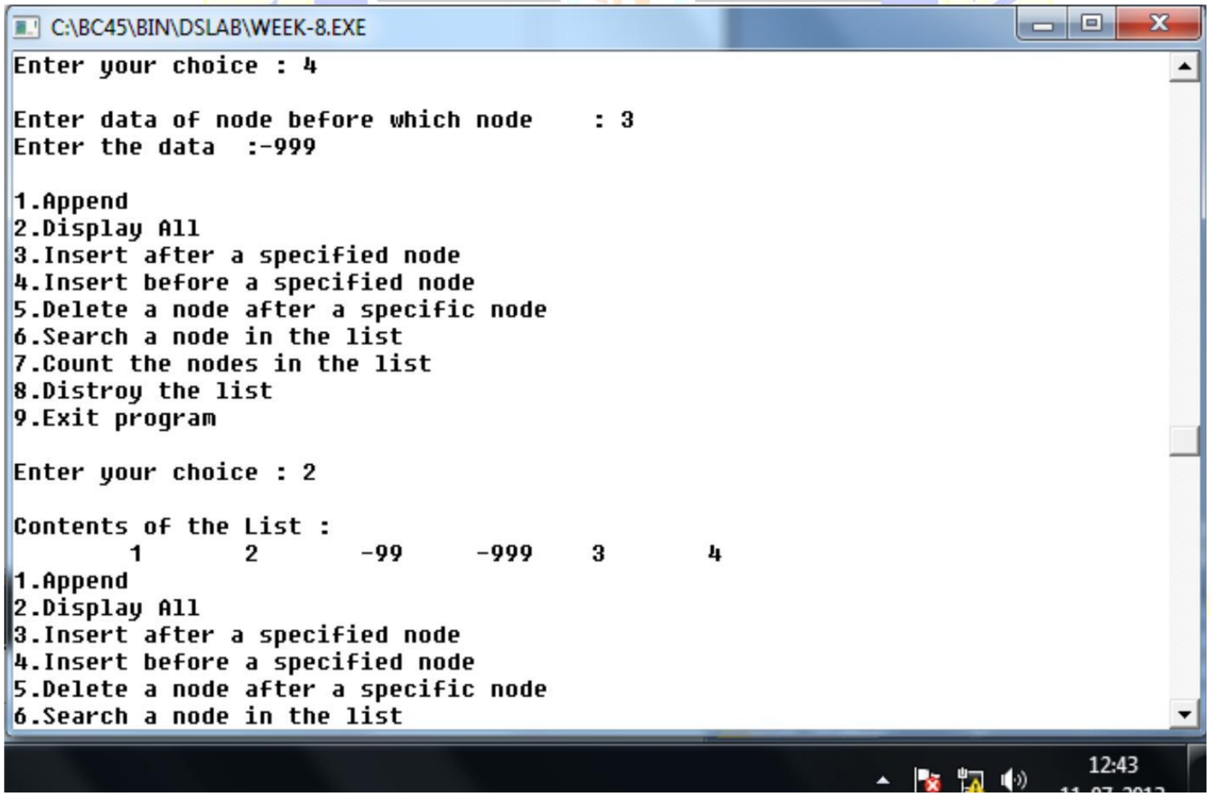
```
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 1
Enter the data  :3

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 1
Enter the data  :4
```


C:\BC45\BIN\DSLAB\WEEK-8.EXE

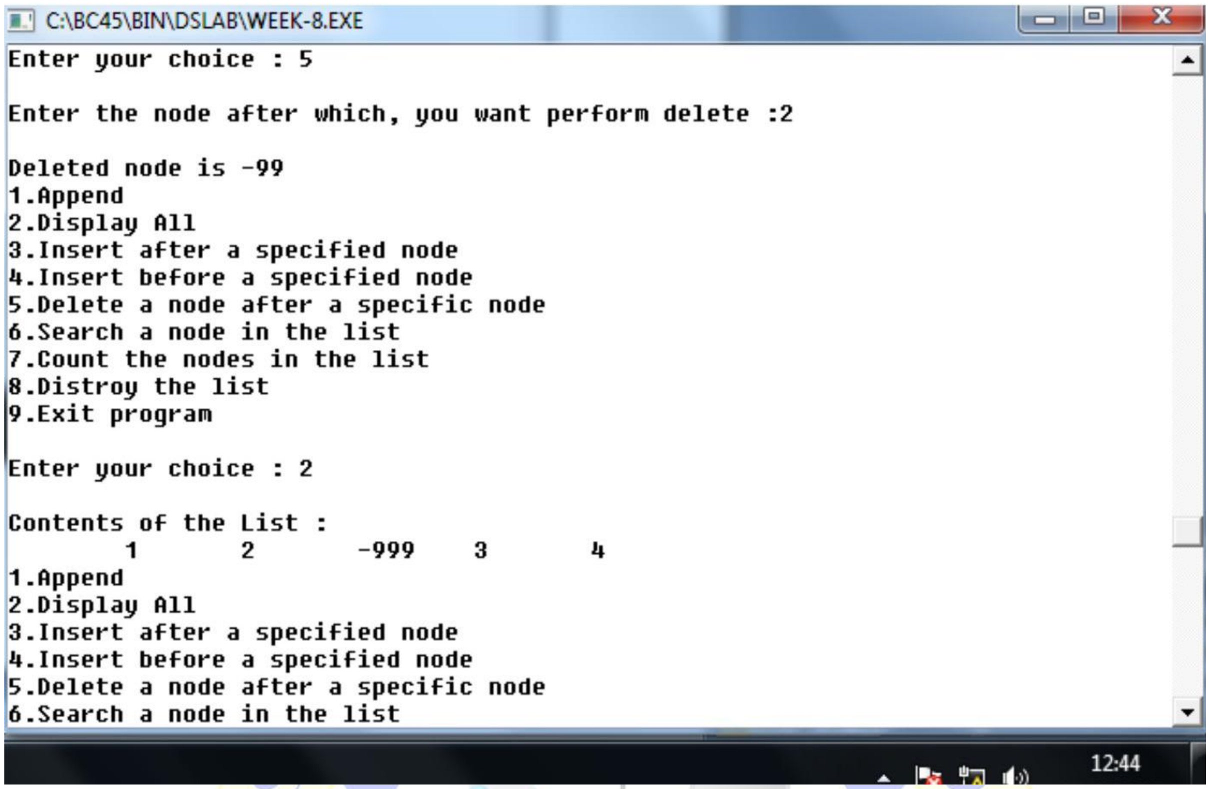
```
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 2

Contents of the List :
        1       2       3       4
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 3
```



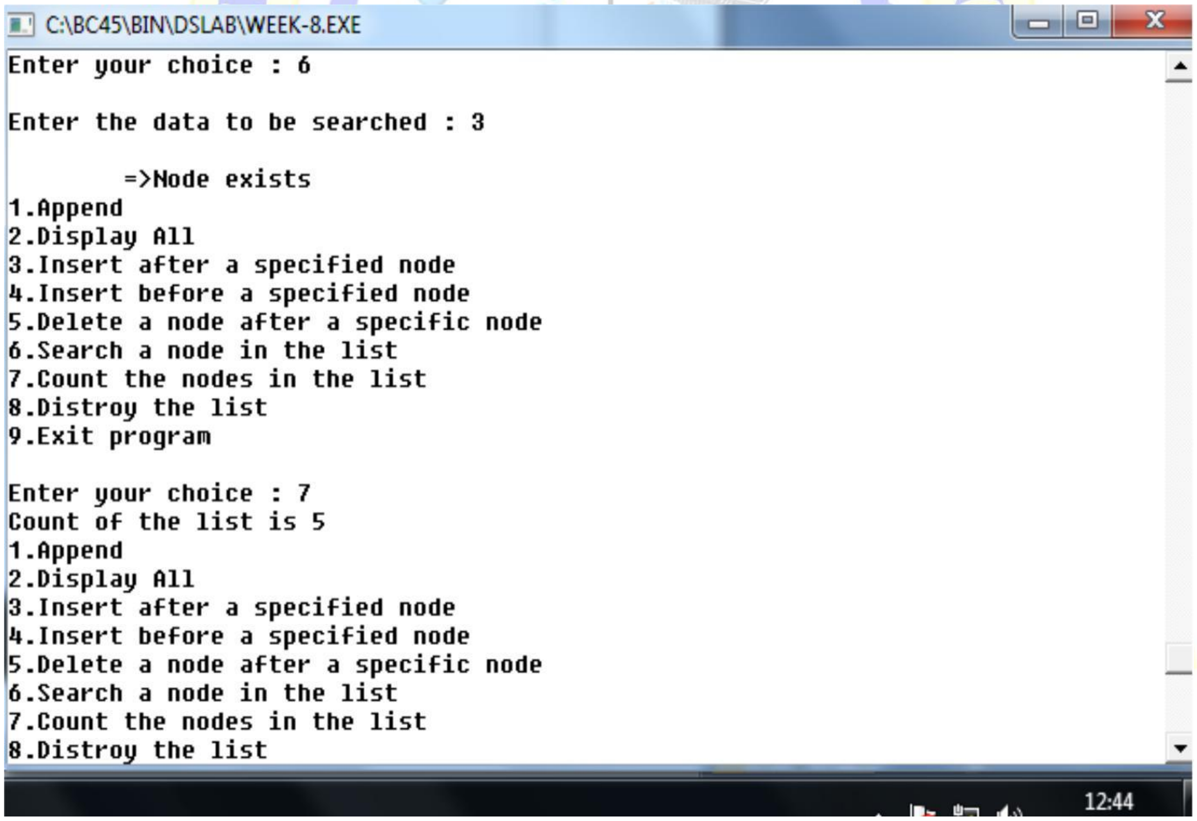
```
Enter data of node after, which you want to insert node : 2
Enter the data  :-99

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 2

Contents of the List :
        1       2       -99     3       4
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
```



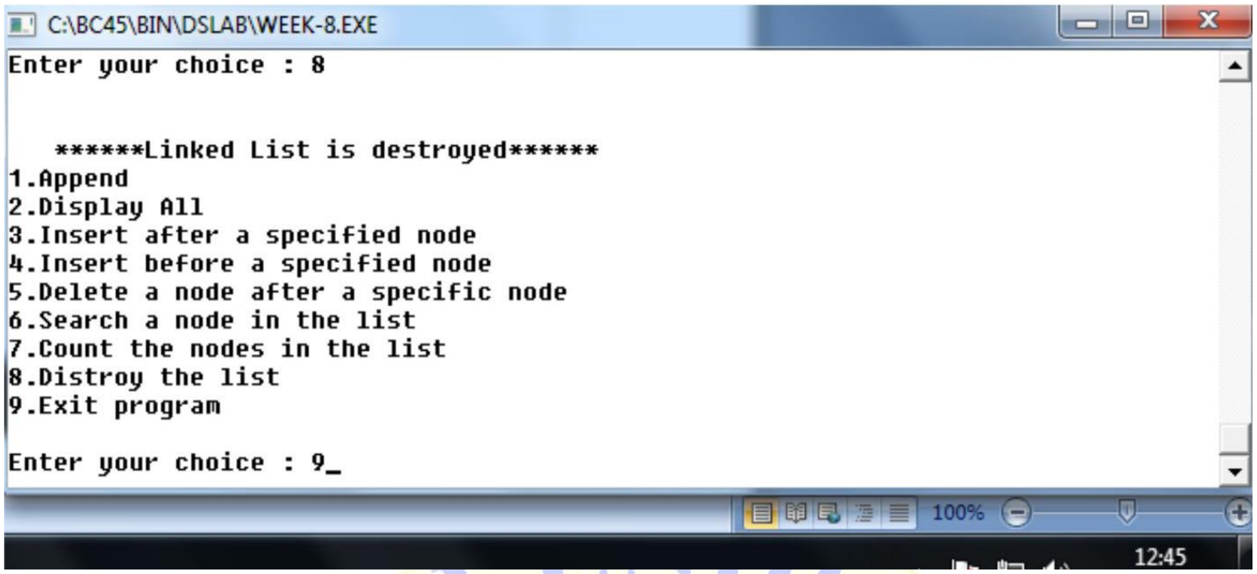
```
Enter your choice : 4

Enter data of node before which node    : 3
Enter the data  :-999

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 2

Contents of the List :
        1       2       -99     -999    3       4
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
```



```
Enter your choice : 5

Enter the node after which, you want perform delete :2

Deleted node is -99
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 2

Contents of the List :
        1       2       -999    3       4
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
```



```
Enter your choice : 6

Enter the data to be searched : 3

        =>Node exists
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 7
Count of the list is 5
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
```


C:\BC45\BIN\DSLAB\WEEK-8.EXE

```
Enter your choice : 8


   ******Linked List is destroyed******
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node after a specific node
6.Search a node in the list
7.Count the nodes in the list
8.Distroy the list
9.Exit program

Enter your choice : 9_
```

<u>Viva Questions:</u>

1. Describe the steps to insert data into a singly linked list.
2. Explain how to reverse singly link list.
3. Define circular linked list.
4. Define circular linked list.
5. Implement a linked list in C using a struct. Have the data be integers.
6. What is difference between Singly Linked List and Doubly Linked List data structure?
7. How to insert a node at the beginning of the list?
8. . How to represent a linked list node?
10. Graphically represent a linked list

EX. NO: 9(A) ADDITION OF TWO LARGE INTEGERS USING LINKED LIST

Aim:

Write a C program for adding two large integers which are represented in linked list fashion.

Description:

Given two linked list, each node contains one digit number, we need to add these two linked list. Result should be stored in third linked list. It should be noted that the head node contains the most significant digit of the number. For example, two numbers 12345 and 56789 are represented in form of linked list as follows:

First List: 1->2->3->4->5->NULL
Second List: 5->6->37->8->9->NULL
Resulting linked list should be: 6->9->1->3->4->NULL

Algorithm:

Step1: declare a structure for creating a node
Step2: declare the following functions

```
get_node();//function to allocate memory to a node
 read_num(node *head1);// function to read the number from a node
display_num(node *temp,node *head);//function to display the result
Sum(node *head1,node *head2,node *head3);//function to add those two
```

Step3: Allocate memory for three header nodes and keep a negative number(-999) in their last node.
Step4: Read the first number and keep it in a linked list format where each node carries one digit. The first node of this list is head1.
Step5: Read the second number and keep it in a linked list format where each node carries one digit.the first node of this list is head2.
Step6: Add the to linked list nodes one by one with the carry if any
Step7: Store the result in a third linked list

Step8: Display the resultant list(third list).

Step9: stop.

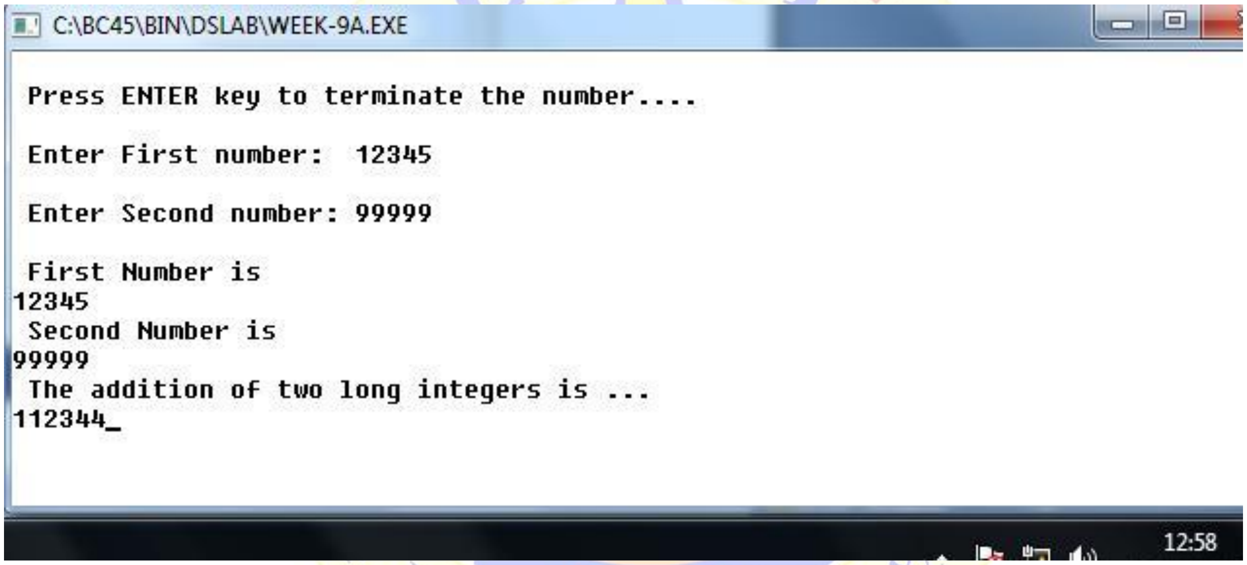
Sample Input:

Enter the first number a= 2->1->7

Enter the second number b= 3->4

Result= 2->5->1

Observed Output:



C:\BC45\BIN\DSLAB\WEEK-9A.EXE

```
 Press ENTER key to terminate the number....

 Enter First number:  12345

 Enter Second number: 99999

 First Number is
12345
 Second Number is
99999
 The addition of two long integers is ...
112344_
```

12:58

EX. NO: 9(B) REVERSE ELEMENTS OF A SINGLE LINKED LIST

Aim:

Write a C programme to reverse elements of a single linked list.

Description:

Reversing the linked list starting from the very first node – the head node. What it basically comes down to is changing pointers from one node to the next so that the entire linked list becomes reversed. There is definitely a processm that we will want to follow in order to do that:

1. The head node's next pointer should be set to NULL since the head will become the tail. This is an exception for the head node, and can be done outside the while loop. But, before we do this we will need a temp variable to point to the 2nd node (the node after the head node), because the only way to reference the 2nd node is through the head node's next pointer.

2. The 2nd node (the node after the head node) should have it's own next pointer changed to point to the head node. This will reverse the order of the nodes. But, remember that the 2nd node's next pointer will at first be pointing to the 3rd node. This means that before we change the 2nd node's next pointer, we have to save a reference to the 3rd node otherwise we will have no way of referencing the 3rd node. So, we simply store a reference to the 3rd node in a variable before we change the 2nd node's next pointer.

3. The 3rd node then becomes the "first" node in the while loop and we repeat the process of changing pointers described in step 2.

4. Continue step 3 until we come across a node that has a next pointer set to NULL. When we do come across a NULL next pointer we just set the head node to point to the node that has the NULL next pointer. This node was previously the tail node, but is now the head node because we are reversing the linked list.

Algorithm:

```
struct node
{
        int info;
        struct node *link;
};
//create a node
typedef struct node *NODE;
NODE *first;
```

Algorithm for main function

```
Step 1: print the number of elements in the list
Step 2: read n
Step 3: i=1
Step 4: repeat the following until i<=n
        Print _Enter the item to be inserted
        Read x
        Call function insert(x)
         i=i+1
Step 5: call function display ()
Step 6: call function reverse ()
Step 7: print _after reversing‘
Step 8: display ()
Step 9: stop
```

Algorithm for insert a new node - insert()

```
Step 1: call function for allocating memory space for element-
        temp=getnode()
Step 2: temp->info-I;
Step 3: temp->link=NULL
Step 4: check whether the list is empty or not, if empty then make temp as first node
        Otherwise add temp node to list
        if(first==NULL)
        {
                first=temp; return ;
        }
Step 5: cur=first;
Step 6: add the temp at the end of the list, to reach the end do the following steps
        while(cur->link!=NULL)
                cur=cur->link;
Step 7: now add the temp at the end
         cur->link=temp
Step 8: stop
```

Algorithm for getnode(int info)

```
Step 1: create a new node
        NODE x;
        x=(struct llist *)malloc(sizeof(Struct llist));
Step 2: check overflow
        if x==NULL print overflow;
        return;
Step 3: if memory available then
        return x
```

Algorithm for reverse a list—reverse()

```
Step 1: create three nodes
        NODE p,q,r;
Step 2: keep the p=first;
Step 3: if it is not the last element in the
        list if(p&&p->link)
        {
                q=p->link;
                 while(q->link)
```

```
                    {
                              r=q->link;
                              q->link=p;
                              p=q;
                              q=r;
                    }
                    q->link=p;
                    first->link=NULL;
                    first=q;
          }
          return;
}
```

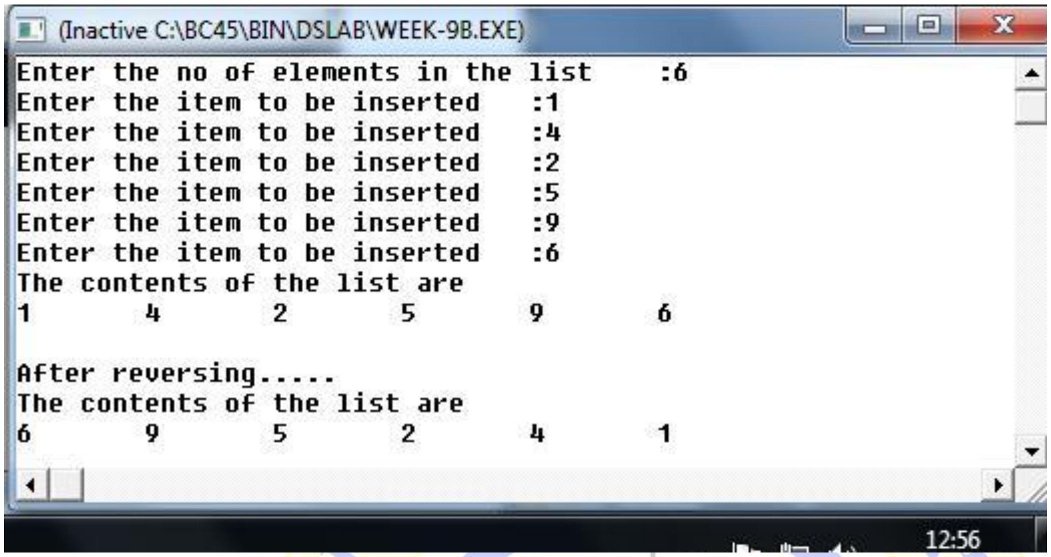
Algorithm for display()

```
Step 1: if(first==NULL) then
          printf("LIST EMPTY\n"); return;
Step 2: print _The contents of the list are‘
          temp=first;
          while(temp!=NULL)
          {
                    printf("%d\t",temp->info);
                    temp=temp->link;
          }
Step 3: stop.
```

<u>Sample Input:</u>

```
Enter no of elements: 4
Enter elements: 1 2  3  4
The elements are        1         2        3        4
After reversing         4        3        2        1
```

Observed Output:



```
Enter the no of elements in the list     :6
Enter the item to be inserted   :1
Enter the item to be inserted   :4
Enter the item to be inserted   :2
Enter the item to be inserted   :5
Enter the item to be inserted   :9
Enter the item to be inserted   :6
The contents of the list are
1       4       2       5       9       6

After reversing.....
The contents of the list are
6       9       5       2       4       1
```

EX. NO: 9(C) STORE A POLYNOMIAL EXPRESSION IN MEMORY USING LINKED LIST

Aim:

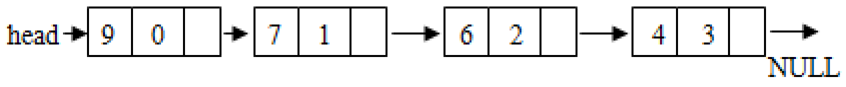
Write a C programme to store a polynomial expression in memory using linked list

Description:

A polynomial may also be represented using a linked list. A structure may be defined such that it contains two parts- one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

```
Struct polynomial
{
int coefficient;
int exponent;
struct polynomial *next;
};
```

Thus the above polynomial may be represented using linked list as shown below:



Algorithm:

/* algorithm for storing a polynomial expression in memory using linked list*/ Declare structure:

```
Struct node

{
float coeff;
int exp;
struct node * link;
```

```
        };
typedef struct node * node;
Node first= null;
1: start
2: declare p g node type
3: declare ch, I, new integer type and assign i=1
4: declare c g float type
5: read n.
6: while i<=n
        6.1: read c, e
        6.2: increment I by 1
        6.3: if first!=0 then
                6.3.1: assign p=first
                6.3.2: while(p->link)!=0
                do
                6.3.3: p=(p->link)
                6.3.4: allocate memory to p->link using getnode()
                6.3.5: assign p=(p->link)
                6.3.6: (p->coeff)=c and (p->exp)=e
                6.3.7: assign (p->link)=null
                6.3.8:end if.
        6.4 : else do
            6.4.1 : (first)=getnode().
            6.4.2 : (first -> coeff)=c.
            6.4.3 : (first -> link)=null
           6.4.4 : end else
7.  end while
8.  call display() function.
9.  end.
```

Algorithm for getnode()

//output: allocates memory to a pointer type variable in the system memory.

```
1. Start
2. Declare _x‘ of node type
3. Allocate memory tox using _malloc‘ library function
         The syntax follows
       =(node) malloc (size of (struct node));
4. If x=null then
   Print ―memory cannot be allocated‖.
   4.1 call exit (0).
   4.2 End if
5. Return the value of x.
6. End.
```

Algorithm for display()

//output: displays the content of the linked list.

```
1. Start
2. Declare temp of node type
3. If first=null
     3.1.1  print ―list is empty‖

     3.1.2  return.
     3.1.3  end if
4. Assign temp= first.
5. While temp!= 0 do
       5.1: temp =(temp ->link)
        5.2: end while.
6. end.
```

Sample Input:

Enter the number of terms: 3

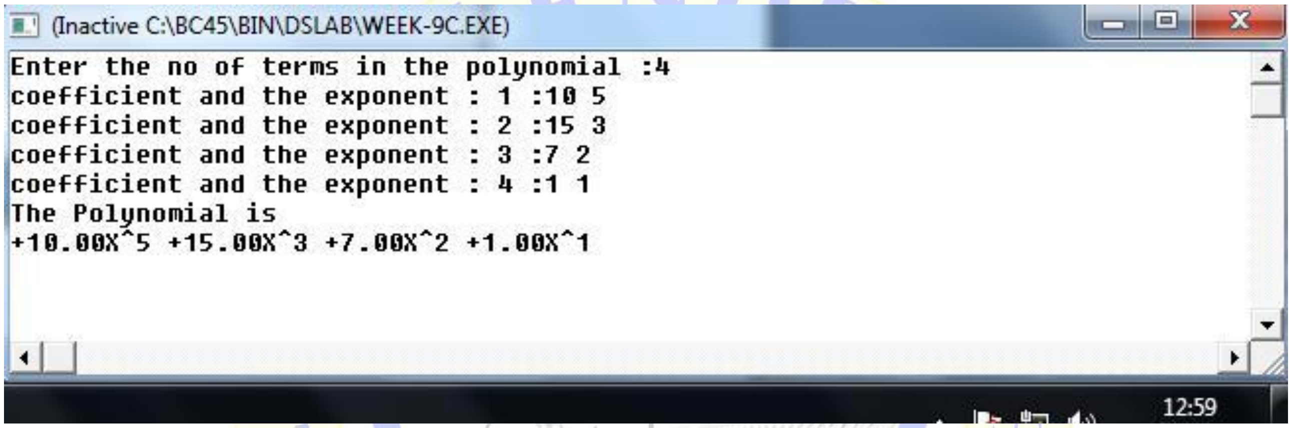
Enter the coefficient and exponent 3 2

Enter the coefficient and exponent 2 1

Enter the coefficient and exponent 8 0

Polynomial expression is : 3x^2+2x^1+8x^0

Observed Output:



```
Enter the no of terms in the polynomial :4
coefficient and the exponent : 1 :10 5
coefficient and the exponent : 2 :15 3
coefficient and the exponent : 3 :7 2
coefficient and the exponent : 4 :1 1
The Polynomial is
+10.00X^5 +15.00X^3 +7.00X^2 +1.00X^1
```

EX. NO: 9(D) REPRESENT THE GIVEN SPARSE MATRIX USING ARRAYS

Aim:

Write a C programme to representation the given sparse matrix using arrays.

Description:

A matrix that has relatively few non-zero entries. It may be represented in much less than n × m space. An n × m matrix with k non-zero entries is sparse if k << n × m. It may be faster to represent the matrix compactly as a list of the non-zero indexes and associated entries, as a list of lists of entries (one list for each row), coordinate format (the values and their row/column positions), or by a point access method.

Algorithm:

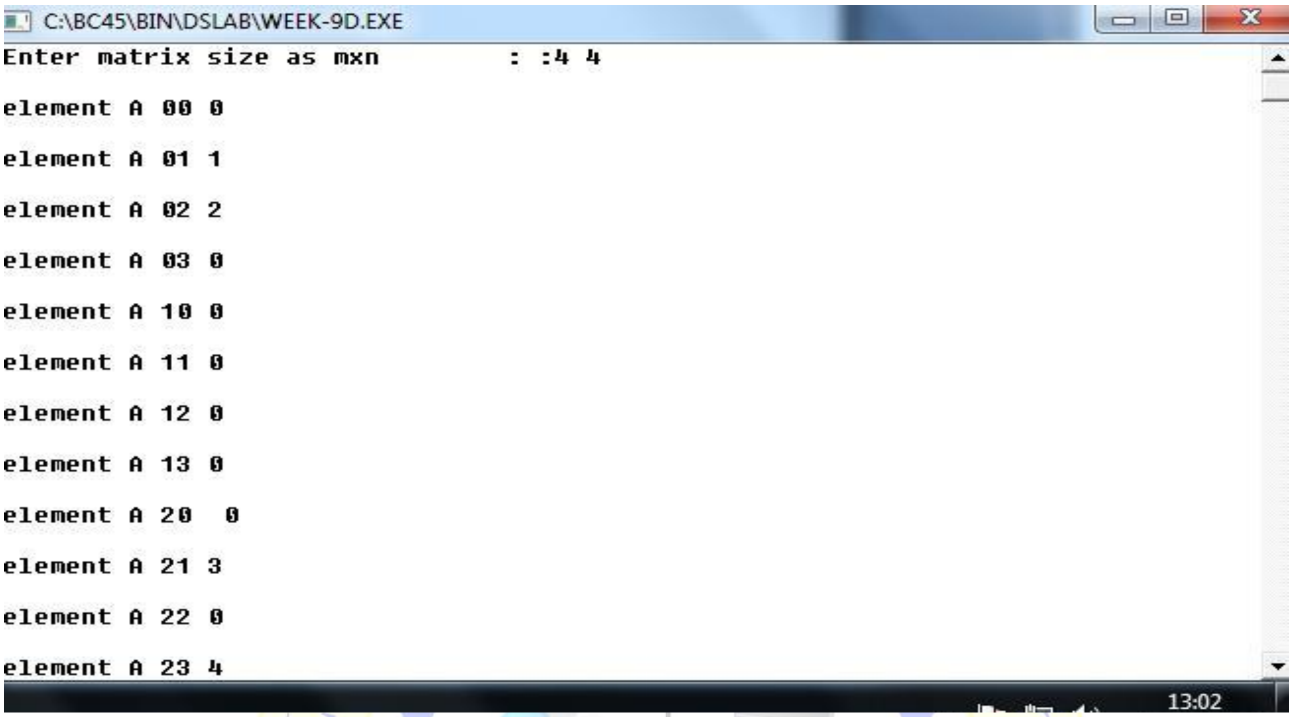
```
/*Algorithm for main function*/
Step 1 print _Enter matrix size as mxn\t: :‘
Step 2 read m,n values set count=0, k=1,i=0.
Step 3. Repeat step 3 until i<m
        Step 3.1.     Repeat step 3.1 until j<n
            Step 3.1.1 printf ‘nelementA ,i,j‘
            Step 3.1.2 read a[i][j]
Step 4 assign i=0,j=0;
Step 5. Repeat step 5 until i<m
    Step 5.1 Repeat step 5.1 until j<n
        Step 5.1.1 check a[i][j] !=0 if true
            Step 5.1.1.1 assign spm[k][0]=I ,spm[k][1]=j,spm[k++][2]=a[i][j].
             Step 5.1.1.2 increment count
Step 6. assign spm[0][0]=m, spm[0][1]=n,    spm[0][2]=count.
Step 7. Printf _nsparse matrix is : :\n‘
Step 8. Repeat step 8 until i<m
        Step 8.1.Repeat step 8.1 until j<n
            Step 8.1.1 printf ‘spm[i][j \n ]
```

Step 9 getch();

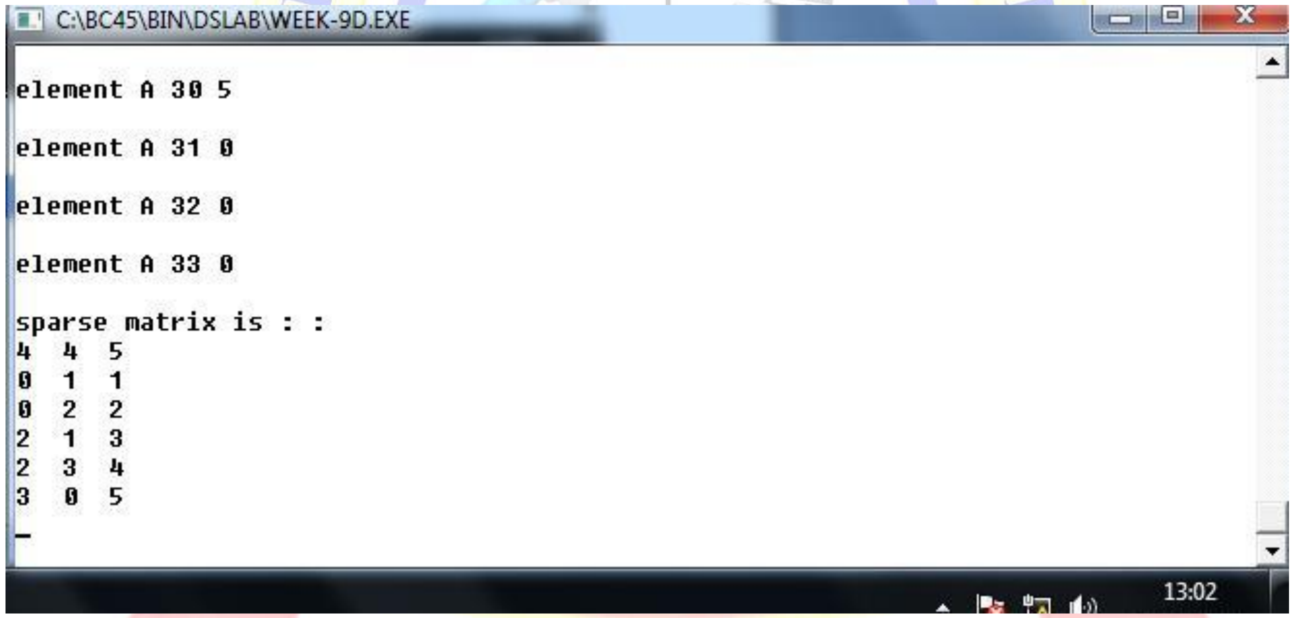
Sample Input:

```
Enter the no of rows: 3
Enter the no of columns: 3
Enter the elements in the array :
1
0
7
5
0
0
3
4
7
 Elements of the  matrix:
         1      0      7
         5      0      0
         3      4      7
Elements of sparse matrix:
         0      0      1
         0      2      7
         1      0      5
         2      0      3
         2      1      4
         2      2      7
```

Observed Output:



```
Enter matrix size as mxn        : :4 4

element A 00 0

element A 01 1

element A 02 2

element A 03 0

element A 10 0

element A 11 0

element A 12 0

element A 13 0

element A 20  0

element A 21 3

element A 22 0

element A 23 4
```



```
element A 30 5

element A 31 0

element A 32 0

element A 33 0

sparse matrix is : :
4  4  5
0  1  1
0  2  2
2  1  3
2  3  4
3  0  5
_
```

EX. NO: 9(E) REPRESENT THE GIVEN SPARSE MATRIX USING LINKED LIST

Aim:

Write a C programme to representation the given sparse matrix using linked list.

Description:

A matrix that has relatively few non-zero entries. It may be represented in much less than n × m space. An n × m matrix with k non-zero entries is sparse if k << n × m. It may be faster to represent the matrix compactly as a list of the non-zero indexes and associated entries, as a list of lists of entries (one list for each row), coordinate format (the values and their row/column positions), or by a point access method. It is implemented with linked list

Algorithm:

```
struct spars
{
        int row,col,item;
        struct spars*next;      }ptr;
structhead_sp
{       introws,col,item;
        struct spars * next;    }*header;
```

```
1. Start
2. Declarei,j as integer type
3. Allocating memory to the header pointerusingmalloc function syntax
     follows Header=(structhead_sp*)malloc(sizeof(structhead_sp));
4. Read header->rows, heade->cols, header->items;
5. If(header->item)=0 then
   5.1. Exit(0);
   5.2. End if
6. Allocate memory to (header_next) (struct spars*)maloc(sizeof(struct spars));
```

```
7.   Read header->next->row,header->next->col,header->next->item
8.   Initializing ptr=(header->next)
9.   Create nodes for sparse matrix elements For(i=2 where i<=header:i++)
     9.1. Allocate memory for(ptr□next)
     9.2. Initialize ptr=(ptr->next)
     9.3. End for loop
10.  Assign(ptr->next)->NULL
11.  Displaying the matrix elements
     11.1.      Ptr=(header→next)
     11.2.      For(i=0;i<(header→rows;i++)
            11.2.1.1.   For(j=0;j<(header→cols);j++
                  11.2.1.1.1.1.        If  ptr!=NULL then
                        11.2.1.1.1.1.1.1.1.     Ifi=(ptr→rows)  and j=(ptr→col)then
                        11.2.1.1.1.1.1.1.2.     Ptr=(ptr→next)
                        11.2.1.1.1.1.1.1.3.     Otherwise print (—0\‖)
                  11.2.1.2otherwise print(—0\‖);
                  11.2.1.3end if
            11.2.2 end for
        11.3 end for
12.end
```

Sample Input:

Sparse Matrix using Linked List

Enter the number of rows of the matrix:3

Enter the number of columns of the matrix:3

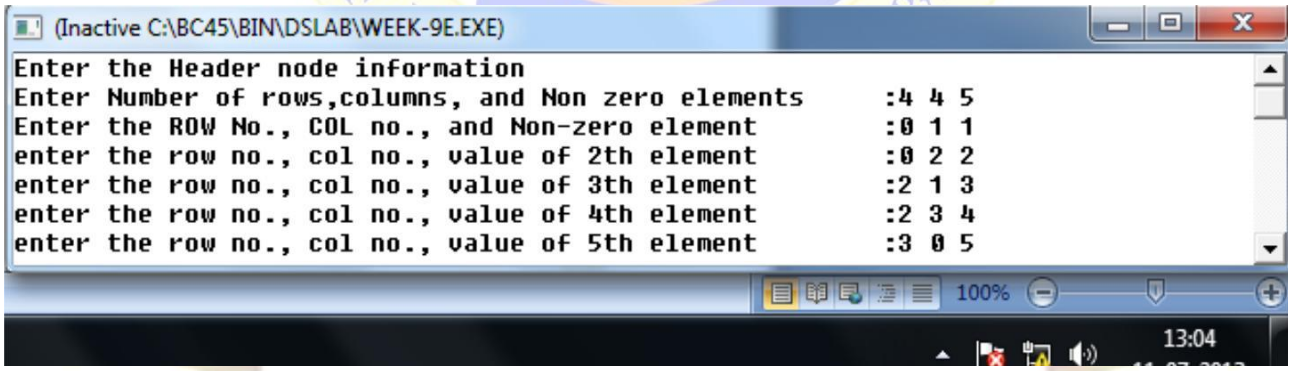
Enter the elements:

1

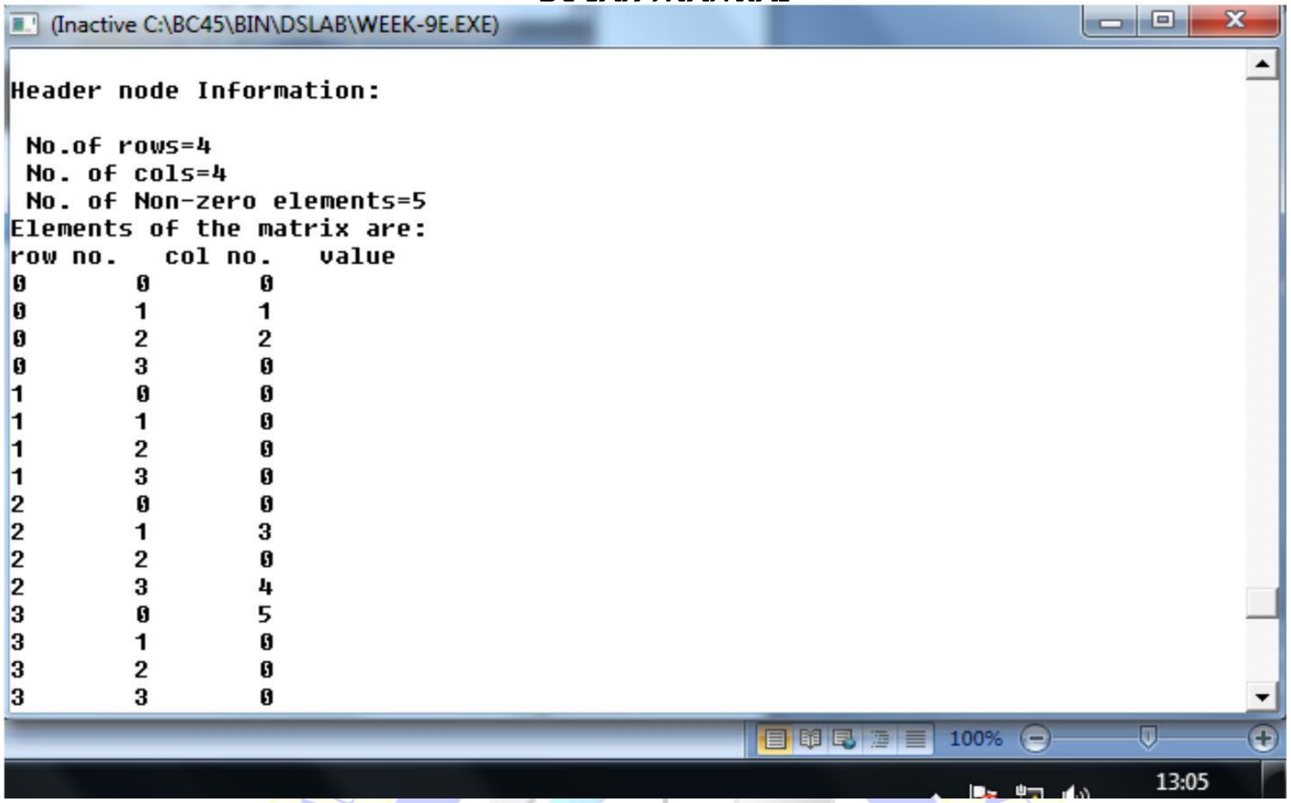
8

```
0

0

7

0

7

0

0

Sparse matrix using linked list:

Row          Column          Element

0            0               1

0            1               8

1            1               7

2            0               7
```

Observed Output:



```
Enter the Header node information
Enter Number of rows,columns, and Non zero elements      :4 4 5
Enter the ROW No., COL no., and Non-zero element         :0 1 1
enter the row no., col no., value of 2th element         :0 2 2
enter the row no., col no., value of 3th element         :2 1 3
enter the row no., col no., value of 4th element         :2 3 4
enter the row no., col no., value of 5th element         :3 0 5
```



```
Header node Information:

 No.of rows=4
 No. of cols=4
 No. of Non-zero elements=5
Elements of the matrix are:
row no.   col no.   value
0       0       0
0       1       1
0       2       2
0       3       0
1       0       0
1       1       0
1       2       0
1       3       0
2       0       0
2       1       3
2       2       0
2       3       4
3       0       5
3       1       0
3       2       0
3       3       0
```

Viva questions:

1. Explain Traversing the Linked List
2. What does the following function do for a given Linked List?
3. When to use Linked List or Array List?
4. What is data structure?.
5. List out the areas in which data structures are applied extensively?
6. What are the major data structures used in the following areas : RDBMS, Network data model and Hierarchical data model.
7. If you are using C language to implement the heterogeneous linked list, what pointer type will you use?
8. Minimum number of queues needed to implement the priority queue?
9. What is the data structures used to perform recursion?
10. What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

12. Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.

13. Sorting is not possible by using which of the following methods? (Insertion, Selection, Exchange, Deletion)

13. What are the methods available in storing sequential files ?
14. List out few of the Application of tree data-structure?
15. List out few of the applications that make use of Multilinked Structures?
16. In tree construction which is the suitable efficient data structure? (Array, Linked list, Stack, Queue)

EX. NO: 10 CREATE A BINARY TREE OF INTEGERS

Aim:

a) Write a C program to Create a Binary Tree of integers
b) Write a recursive C program, for traversing a binary tree in preorder, inorder and postorder.
c) Write a non-recursive C program, for traversing a binary tree in preorder, inorder and postorder.
d) Program to check balance property of a tree.

Description:

A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root. Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes. Nodes which are not leaves are called internal nodes. Nodes with the same parent are called siblings.

Algorithm:

Structure definition for node in a binary tree typedefstructbst

```
{
    int data;
    struct bst *left,*right;
}node;
```

Algorithm for main function

```
Step 1. Print ― program for binary tree‖
Step 2. Assign root=NULL
Step 3. Check while if true
    3.1 display menu 1. Create 2.display 3. exit
    3.2 ask for choice
```

```
3.3 if case 1:root=NULL;
        do
            {
                    New=get_node();
                    printf("\n Enter The Element: ");
                    scanf("%d",&New->data);
                    if(root==NULL)
                    root=New;
                else
                    insert(root,New);
                printf("\n Do You want To Enter More elements?(y/n):");
                ans=getche();
                }while(ans=='y'||ans=='Y');
                clrscr();
                break;
3.4 ifcase 2:display(root);
                break;
3.5 if case 3:exit(0)
```

Algorithm for getnode()

```
Step 1. Allocate memory for temp node *temp;
 Step 2.temp=(node *)malloc(sizeof(node));
 Step 3 temp->left=NULL;
 Step 4.temp->right=NULL;
 Step 5.  return temp;
```

Algorithm for insert function

```
Step 1.Print _Where to insert left/right of root->data‘
Step 2.get choice ch
Step3. if ((ch=='r')||(ch=='R')) then
```

```
        3.1. if(root->right==NULL)
        3.2. Root->right=New;
        3.3 else insert(root->right,New);
Step 4. else if (root->left==NULL)
        4.1. root->left=New;
Step5. else insert(root->left,New);
```

Algorithm for display()

```
Step1. If root=NULL then print _Tree is not created‘ and return
Step2. Print the following menu
                Print _Which method you want to use‘
                Print _1. Recursive Traversal 2.Non-Recursive Traversal
Step3. Read choice
Step4. If ch=1 then
        4.1 print _You have chosen recursive traversal….‘
        4.2 call  INORDER_TRAVERSAL(root)
        4.3 call  PREORDER_TRAVERSAL(root)
        4.4 call  POSTORDER_TRAVERSAL(root)
        4.5 call  h=height_tree(root);
Step5. else   case 2:
        5.1 print _You have chosen non-recursive traversal….‘
        5.2 call  nonrec_preorder(root)
        5.3 call  nonrec_preorder(root)
        5.4 call  nonrec_preorder(root)
        5.5 call  h=height_tree(root);
Step6. Print _The Height of Tree is _, h
Step7. Stop
```

INORDER_TRAVERSAL (NODE)

```
/* NODE refers to the Root node of the binary tree in its first call to the . Root node is the
starting point of the traversal */
If NODE    NIL then
{
call INORDER_TRAVERSAL (LCHILD(NODE));
print (DATA (NODE)) ;
call INORDER_TRAVERSAL (RCHILD(NODE));
}
end INORDER_TRAVERSAL.
```

```
 POSTORDER_TRAVERSAL (NODE)
/* NODE refers to the Root node of the binary tree in its first call to the . Root node is the
starting point of the traversal */
If NODE    NIL then
 {
     call POSTORDER_TRAVERSAL (LCHILD(NODE));
     call POSTORDER_TRAVERSAL (RCHILD(NODE));
     print (DATA (NODE)) ;
 }
end POSTORDER_TRAVERSAL.
```

```
 PREORDER_TRAVERSAL (NODE)
/* NODE refers to the Root node of the binary tree in its first call to the . Root node is the
starting point of the traversal */
If NODE    NIL then
          {
                    print (DATA (NODE)) ;                    /* Process node  (P) */
call PREORDER_TRAVERSAL (LCHILD(NODE));
call PREORDER_TRAVERSAL (RCHILD(NODE));
}
```

end PREORDER_TRAVERSAL.

Algorithm for nonrec_preorder ():

```
Step 1: Initially push NULL into the STACK, and initialize PTR.
        Set TOP=1 STACK[1]=NULL and PTR=ROOT
Step 2: repeate step 3 to 5 untill PTR != NULL
Step 3: Apply PROCESS to PTR->INFO
Step 4: check for right child
        If  PTR->RIGHT !=NULL then push on stack
                Set TOP=TOP+1 and
                STACK[TOP]=PTR->RIGHT
Step 5:Check for left child
        If    PTR->LEFT != NULL then PTR=PTR->left
        Else set ptr=STASCK[TOP] and TOP=TOP-1
Step 6: stop.
```

Algorithm for nonrec_postorder:

```
Step 1: Initially push NULL into the STACK, and initialize PTR.
        Set TOP=1 STACK[1]=NULL and PTR=ROOT
Step 2: push left most path into the STACK
        repeate step 3 to 5 untill PTR != NULL
step 3: pushes PTR on STACK
        TOP=TOP+1 stack[top]=PTR
Step 4: if PTR-> RIGHT != NULL, then push on the STACK
        Set TOP=TOP+1   STACK [TOP]=PTR->RIGHT
Step 5: PTR=PTR->LEFT
Step 6: PTR=STACk[TOP]  TOP=TOP-1
Step7: repeat while ptr>0
                (a)apply PROCESS to PTR->info (b)set
                PTR=STACK[TOP TOP=TOP-1
Step 8: if PTR<=0,then
                (a)set PTR=-PTR
                (b)go to step 2
```

```
Step 9: end
```

Algorithm for nonrec_inorder:

```
Step 1: Initially push NULL into the STACK, and initialize PTR.
        Set TOP=1 STACK[1]=NULL and PTR=ROOT
Step 2: repeate while PTR!=NULL        [pushes left   most path in STACK]
        (a) Set TOP=TOP+1 and STACK[TOP]= PTR [save nodes]
        (b) Set PTR=PTR->LEFT      [updates PTR]
Step 3: set PTR=STACK[TOP] and TOP=TOP-1 [pops node from stack
Step 4: repeate steps 5 to 7 while PTR != NULL      [backtracking]
Step 5: apply PROCESS to INFO->PTR
Step 6: check for right child
        If  PTR->RIGHT != NULL, then
        (a)  Set PTR=PTR->RIGHT
        (b)  Go to step 3
Step 7: set PTR=STACK[TOP] and TOP=TOP-1    [pops node]
Step 8: end
```

```
/*Algorithm to determine height of the binary tree- Height (NODE)*/
 Tree_Height(NODE)
if NODE = NULL, then return 0
else
{
LeftHeight = Tree_Height (NODE ->LEFT)
RightHeight = Tree_Height (NODE ->RIGHT)
 if(LeftHeight > RightHeight )
                    return LeftHeight + 1
    else
                    return RightHeight + 1
}
end Tree_Height.
```

SampleOutput:

```
1. Insert
2. Inorder
3. Preorder
4. Postorder
5. Exit;
 Enter your choice : 1
 Enter the item 12
1. Insert
2. In order
3. Preorder
4. Postorder
5. Exit;
 Enter your choice : 1
 Enter the item 23
Where to insert left/right of root: l
1. Insert
2. Inorder
3. Preorder
4. Postorder
5. Exit;
 Enter your choice : 1
 Enter the item: 56
Where to insert left/right of root: r
1. Insert
2. Inorder
3. Preorder
4. Postorder
5. Exit;
 Enter your choice : 2
Inorder:23  12  56
    1. Insert
```

2. Inorder
3. Preorder
4. Postorder
5. Exit;

Enter your choice : 3

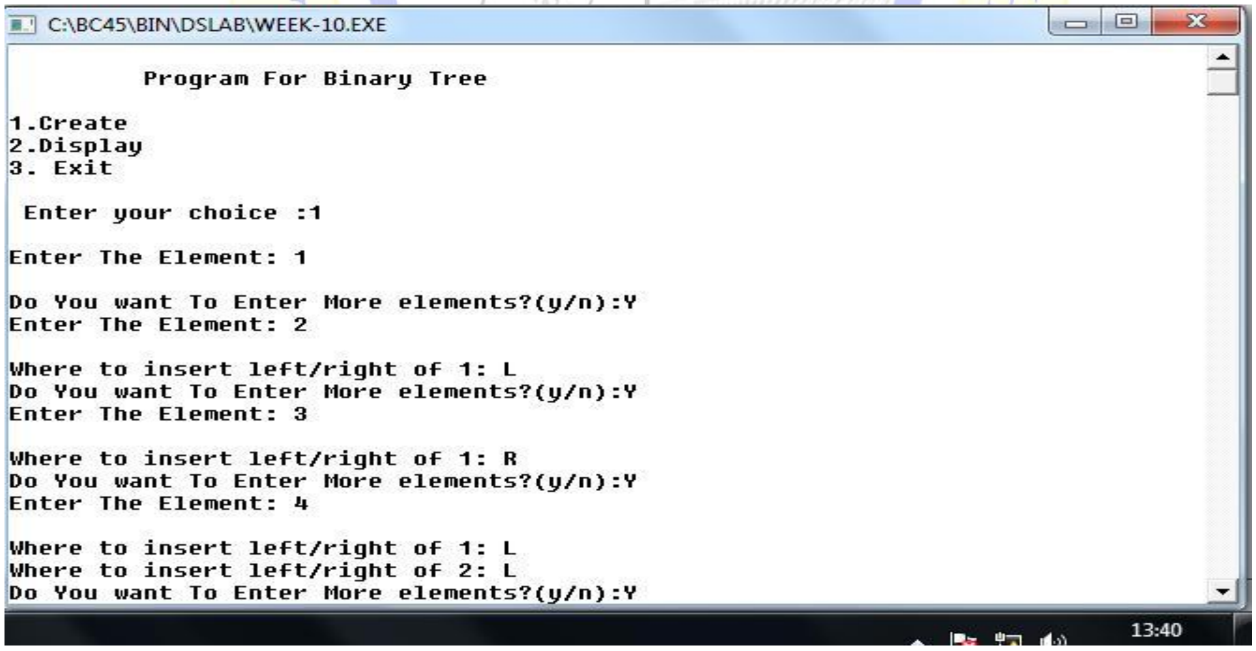
Preorder: 12 23 56

1. Insert
2. Inorder
3. Preorder
4. Postorder
5. Exit;

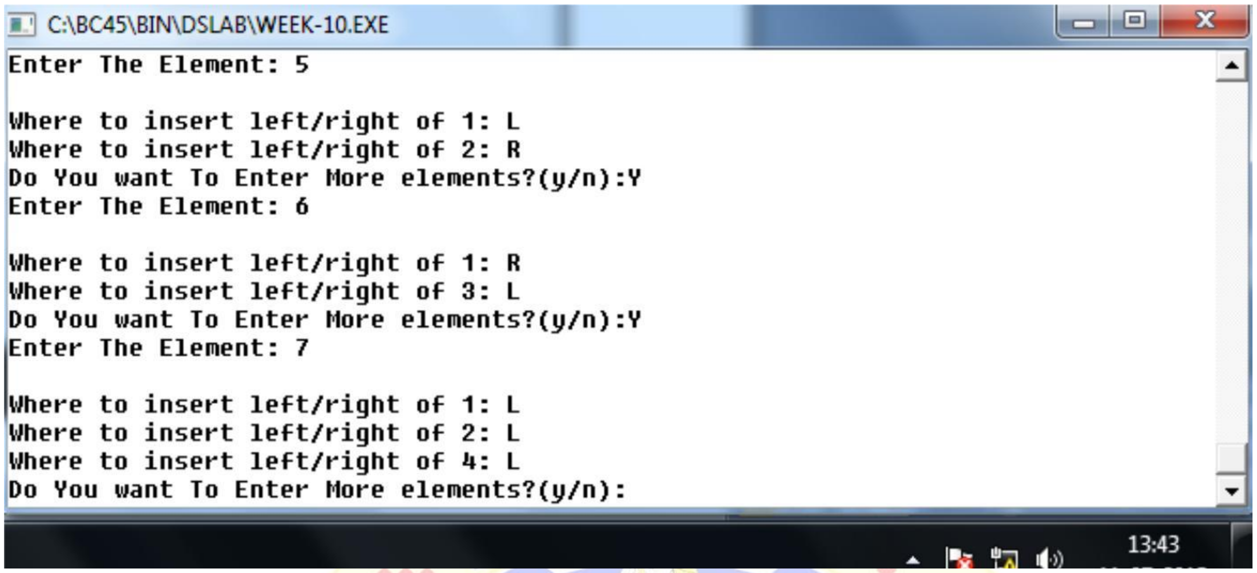
Enter your choice : 4

Postorder: 23 56 12

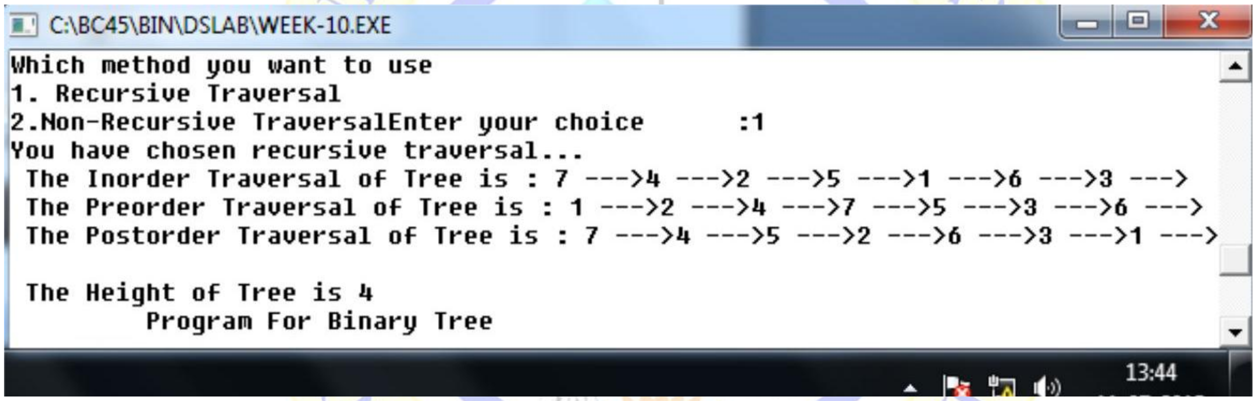
Observed Output:



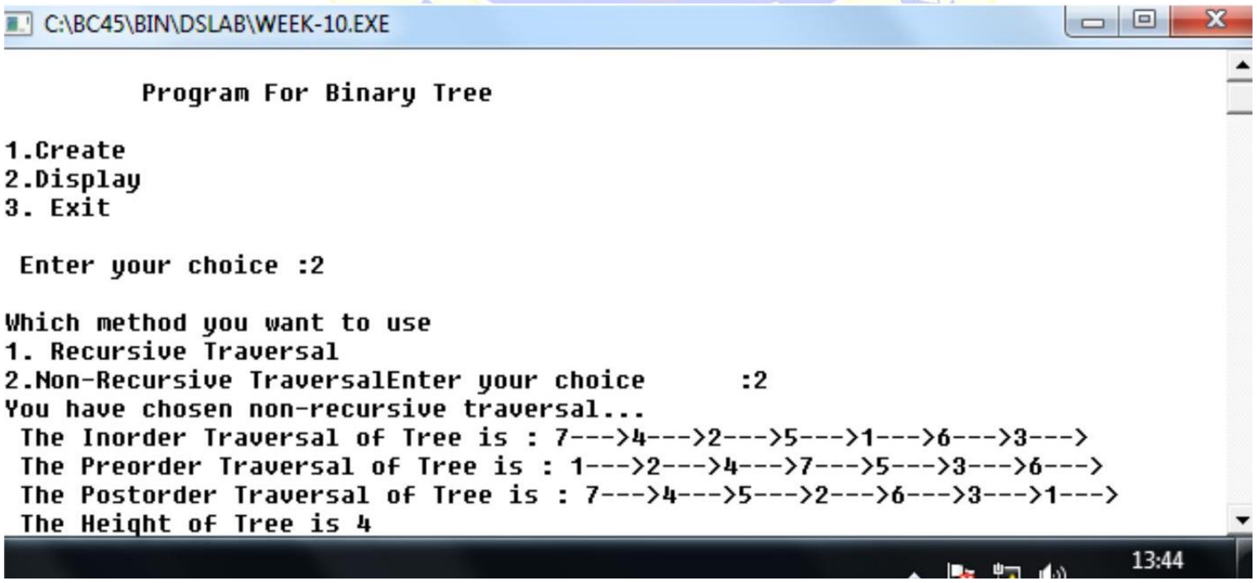
```
C:\BC45\BIN\DSLAB\WEEK-10.EXE

          Program For Binary Tree

1.Create
2.Display
3. Exit

 Enter your choice :1

Enter The Element: 1

Do You want To Enter More elements?(y/n):Y
Enter The Element: 2

Where to insert left/right of 1: L
Do You want To Enter More elements?(y/n):Y
Enter The Element: 3

Where to insert left/right of 1: R
Do You want To Enter More elements?(y/n):Y
Enter The Element: 4

Where to insert left/right of 1: L
Where to insert left/right of 2: L
Do You want To Enter More elements?(y/n):Y
```



```
Enter The Element: 5

Where to insert left/right of 1: L
Where to insert left/right of 2: R
Do You want To Enter More elements?(y/n):Y
Enter The Element: 6

Where to insert left/right of 1: R
Where to insert left/right of 3: L
Do You want To Enter More elements?(y/n):Y
Enter The Element: 7

Where to insert left/right of 1: L
Where to insert left/right of 2: L
Where to insert left/right of 4: L
Do You want To Enter More elements?(y/n):
```



```
Which method you want to use
1. Recursive Traversal
2.Non-Recursive TraversalEnter your choice      :1
You have chosen recursive traversal...
 The Inorder Traversal of Tree is : 7 --->4 --->2 --->5 --->1 --->6 --->3 --->
 The Preorder Traversal of Tree is : 1 --->2 --->4 --->7 --->5 --->3 --->6 --->
 The Postorder Traversal of Tree is : 7 --->4 --->5 --->2 --->6 --->3 --->1 --->

 The Height of Tree is 4
         Program For Binary Tree
```



```

         Program For Binary Tree

1.Create
2.Display
3. Exit

 Enter your choice :2

Which method you want to use
1. Recursive Traversal
2.Non-Recursive TraversalEnter your choice      :2
You have chosen non-recursive traversal...
 The Inorder Traversal of Tree is : 7--->4--->2--->5--->1--->6--->3--->
 The Preorder Traversal of Tree is : 1--->2--->4--->7--->5--->3--->6--->
 The Postorder Traversal of Tree is : 7--->4--->5--->2--->6--->3--->1--->
 The Height of Tree is 4
```

Viva questions:

1. Explain pre-order and in-order tree traversal.
2. Explain implementation of traversal of a binary tree.
3. Explain implementation of deletion from a binary tree.
4. Describe Tree database. Explain its common uses.
6. What is binary tree? Explain its uses.
7. How do you find the depth of a binary tree?
8. What is Root Node
8. What is Leaf Node
9. What is Complete Tree
10. What is Height of a tree

EX. NO: 11 BINARY SEARCH TREE

Aim:

a) Write a C program to Create a BST

b) Write a C programme to insert a note into a BST.

c) Write a C programme to delete a note from a BST.

Description:

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.

ALGORITHM:

```
1: start
2: declare node *new,*root.
3: ans=N'.
4: root=NULL.
5: do
        5.1: read choice.
        5.2: switch(choice)
                Case:1
                        1.1root=NULL
                        1.2do
                          1.2.1: new=getnode();
                          1.2.2:read new=data.
                          1.2.3:if root=NULL then do
                         Root=new.
                        1.2.4:Else do
                        Insert(root,new).
```

```
                    1.2.5:read ans.
                    1.2.6:while ans=‘y‘ OR _Y‘
                    End do while
        case    2:
                    1: if root=NULL do
                        Print ―tree not created ―.
                    2:else do
                        2.1:new=getnode().
                        2.2:read new=data.
                        2.3:insert(root,new).
        case    3:read i.
                Delete(root,i)
        case    4: if root=NULL then do
                    Print ―tree not created‖.

2.declare node *temp.
3.allocate memory to temp
temp=(node *)malloc(sizeof(node));
4.(temp->LCHILD)=NULL
5.(temp->RCHILD)=NULL
6.return  temp.
7.end
```

ALGORITHM insert(T,ITEM):

```
/*T refers to the root node of the bst in its first call and item is the element to be inserted */
1.start
2.if T=NULL then do
        2.1.T<-getnode()
        2.2.T->LCHILD=NULL.
        2.3.T->RCHILD=NULL
        2.4.T->data=item
```

```
3.else if ITEM<(T->data) then do
        3.1 call insert(T->LCHILD,ITEM)
4.else if ITEM>(T->data) then do
        4.1 call insert(T->RCHILD,ITEM)
5.else do
        5.1 print ―duplicate element‖
        5.2 return
6.end insert
```

ALGORITHM delete(T,K).

```
//T refers to the root node of the bst and K is the item to be deleted
1.start
2.declare node *p,*s,*ps,*c,*pp
3.p<-T
4.while p!=NULL AND (p->data) !=K do
        4.1 pp=p
        4.2 if K<(p->data) then do
        4.3 p=(p->LCHILD)
        4.4 else do
                p=p->RCHILD) 5.end
        while
6.if !p then do
        6.1 print ―no element with key K‖
        6.2 return
7. if (p->LCHILD AND p->RCHILD )!=NULL then do
        7.1 s=(p->LCHILD)
        7.2ps=p
        7.3 while s->RCHILD !=NULL do
                7.3.1 ps=s
                7.3.2 s=(s->RCHILD)
                7.3.3 end while
        7.4( p->data)=(s->data)
        7.5 p=s
        7.6 pp=ps
8. if (p->LCHILD!=NULL) then do
        8.1 c = (p->LCHILD)
 9.else do
        9.1 c = ( p->RCHILD)
10. if (p = root) then do
        10.1 root = c
11.   else do
        11.1 if (p == pp->LCHILD) then do
                11.1.1 pp->LCHILD =c
        11.2  else do
              11.2.1 pp->RCHILD = c
    12. free(p)
    13. return
    14. end
```

ALGORITHM inorder(T)

```
//inorder display of tree with root node T
1.start
2. if T=NULL then do
        2.1 return
3.call inorder(T->LCHILD)
4.print ―T->data‖
5. call inorder(T->RCHILD)
6.end inorder
```

ALGORITHM preorder(T)

```
//preorder display of tree with root node T
1.start
2. if T=NULL then do
```

```
        2.1 return
3.print ―T->data‖
4.call preorder(T->LCHILD)
5. call preorder(T->RCHILD)
6.end preorder
```

ALGORITHM postorder(T)

```
//postorder display of tree with root node T
1.start

2. if T=NULL then do
        2.1 return
3.call postorder (T->LCHILD) 4.
call postorder (T->RCHILD)
5.print ―T->data‖
6.end postorder
```

Sample Input:

```
                OPERATIONS—
1-Insert an element into tee
2-delete an element into tee
3-In order traversal
4-preoreder traversal
5-post order traversal
6-exit
Enter your choice:1
Enter  data of node to be ionserted:40
Enter your choice:1
Enter  data of node to be ionserted:20
Enter your choice:1
Enter  data of node to be ionserted:10
```

Enter your choice:1

Enter data of node to be ionserted:30

Enter your choice:1

Enter data of node to be ionserted:60

Enter your choice:1

Enter data of node to be ionserted:80

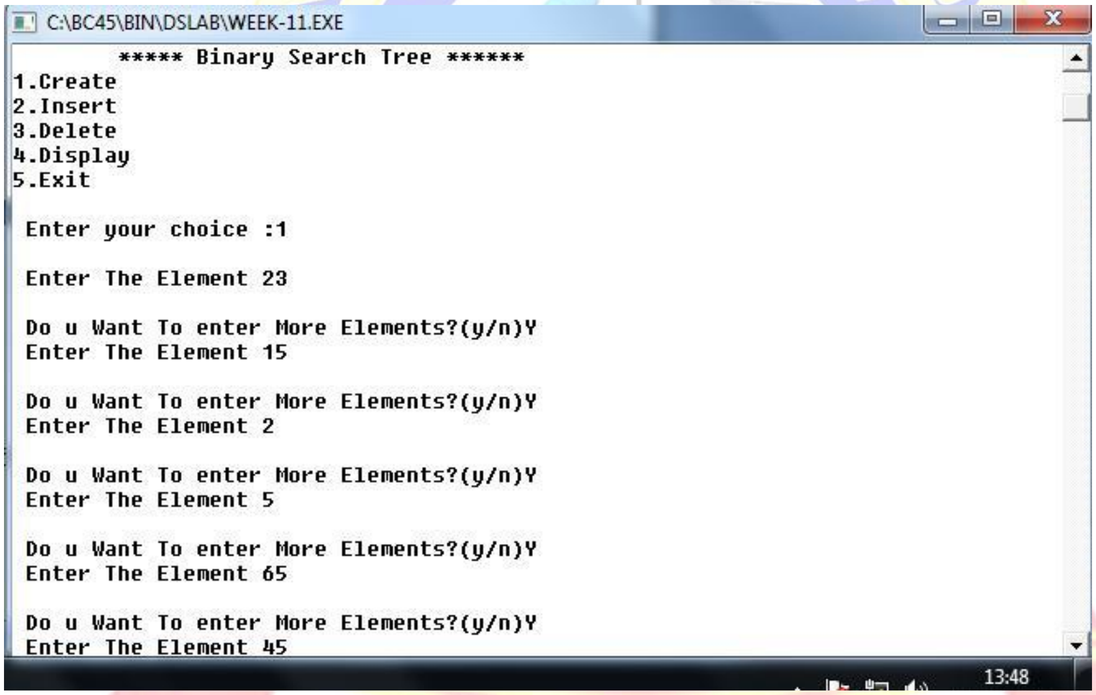
Enter your choice:1

Enter data of node to be ionserted:90

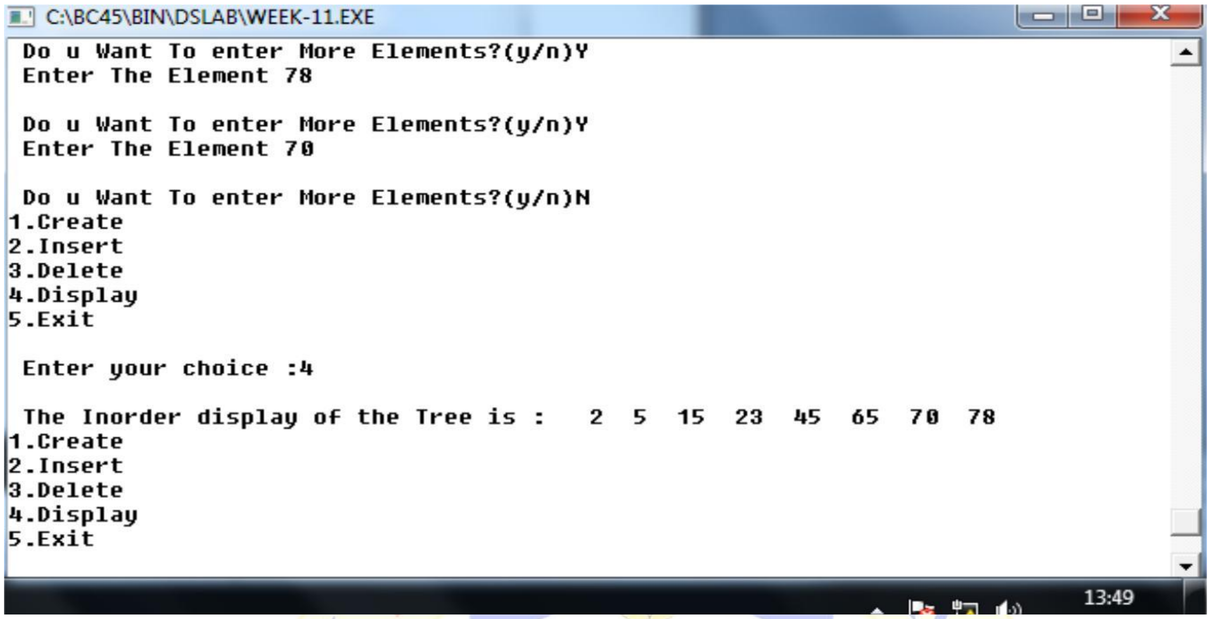
Enter your choice:3

10->20->30->40->60->80-.>90

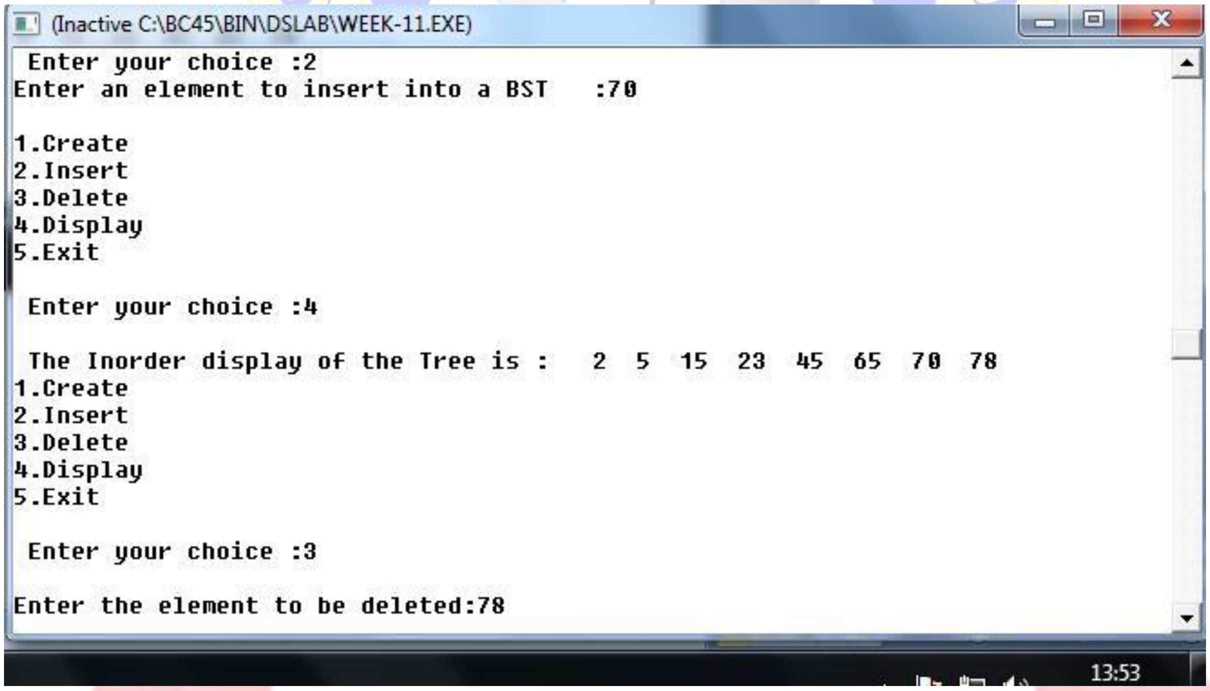
Observed Output:



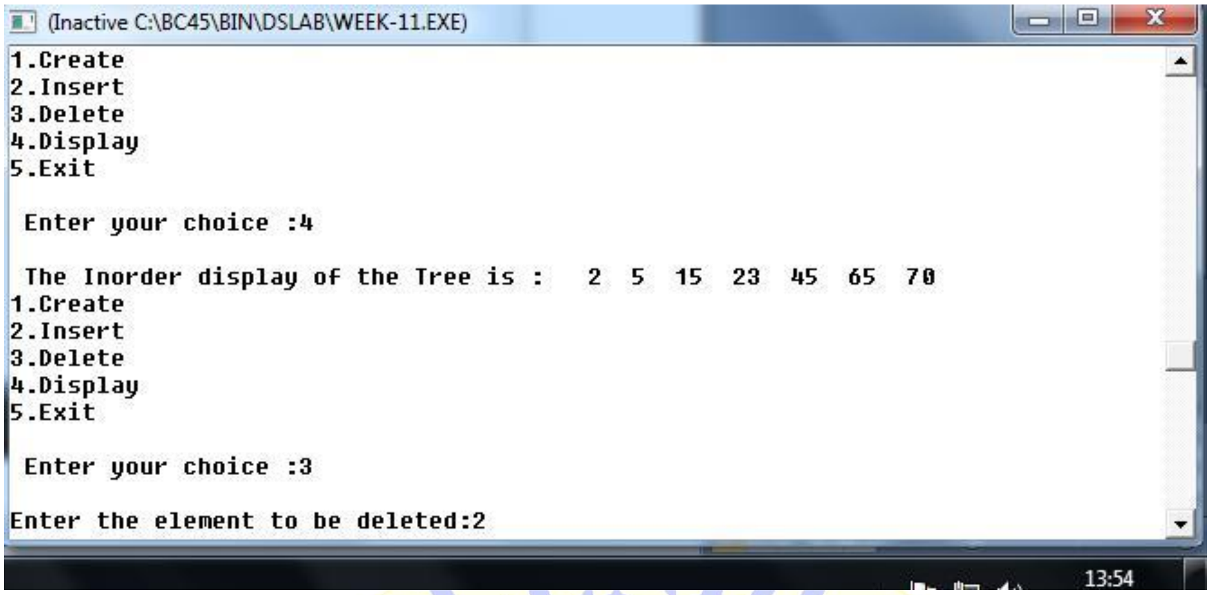
```
        ***** Binary Search Tree ******
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :1

 Enter The Element 23

 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 15

 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 2

 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 5

 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 65

 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 45
```



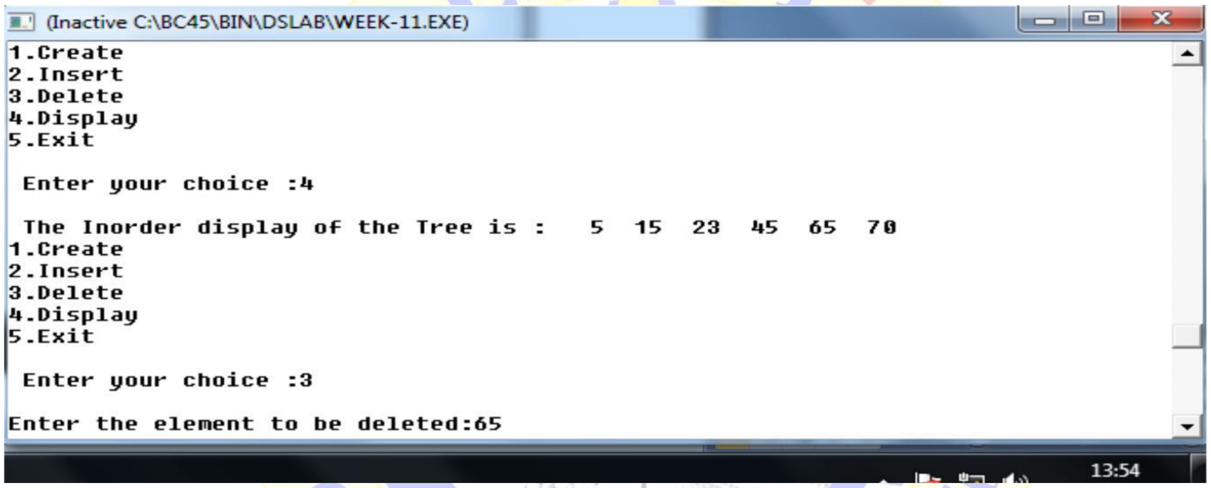
```
C:\BC45\BIN\DSLAB\WEEK-11.EXE
 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 78

 Do u Want To enter More Elements?(y/n)Y
 Enter The Element 70

 Do u Want To enter More Elements?(y/n)N
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :4

 The Inorder display of the Tree is :   2  5  15  23  45  65  70  78
1.Create
2.Insert
3.Delete
4.Display
5.Exit
```



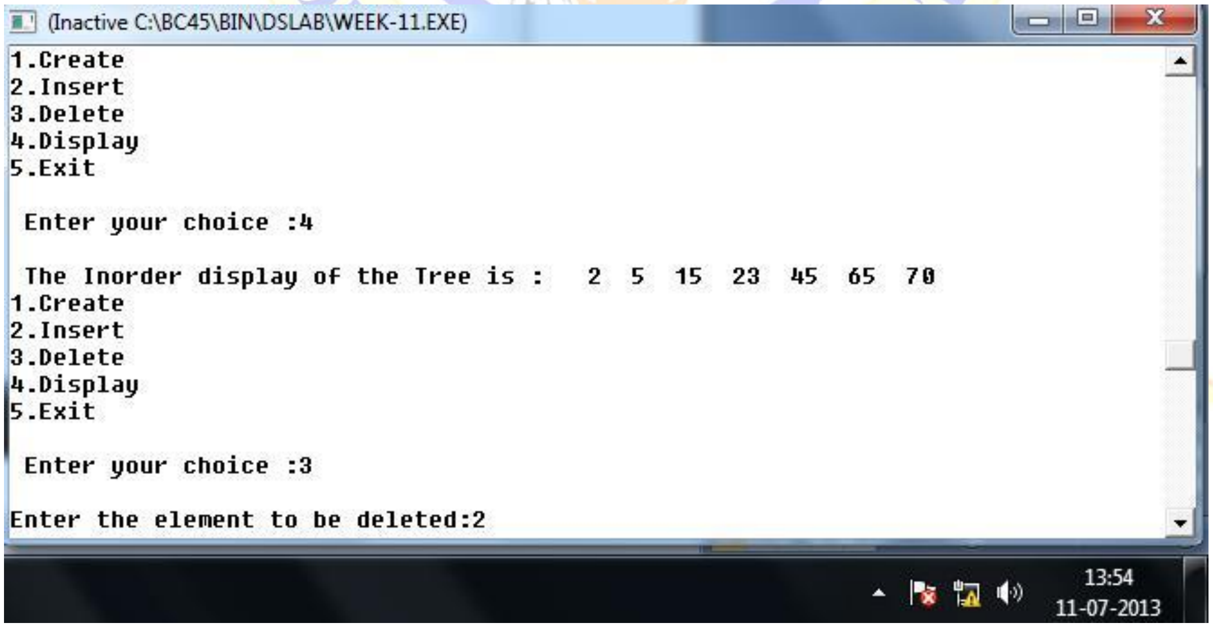
```
(Inactive C:\BC45\BIN\DSLAB\WEEK-11.EXE)
 Enter your choice :2
Enter an element to insert into a BST   :70

1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :4

 The Inorder display of the Tree is :   2  5  15  23  45  65  70  78
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :3

Enter the element to be deleted:78
```



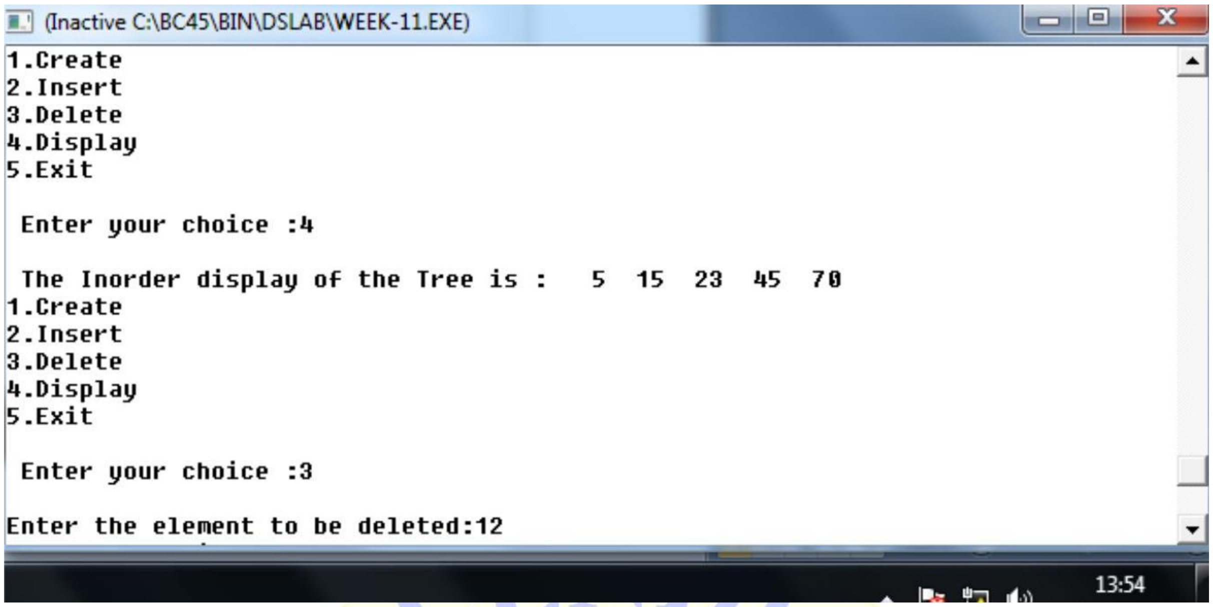
```
(Inactive C:\BC45\BIN\DSLAB\WEEK-11.EXE)
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :4

 The Inorder display of the Tree is :   2  5  15  23  45  65  70
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :3

Enter the element to be deleted:2
```



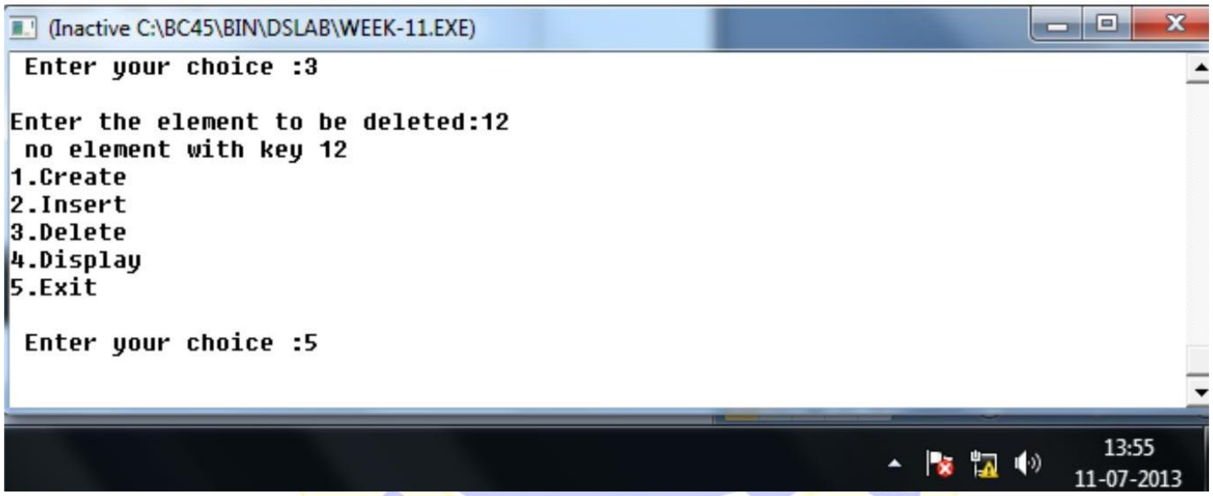
```
(Inactive C:\BC45\BIN\DSLAB\WEEK-11.EXE)
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :4

 The Inorder display of the Tree is :   5  15  23  45  65  70
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :3

Enter the element to be deleted:65
```



```
(Inactive C:\BC45\BIN\DSLAB\WEEK-11.EXE)
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :4

 The Inorder display of the Tree is :   2  5  15  23  45  65  70
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :3

Enter the element to be deleted:2
```



```
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :4

 The Inorder display of the Tree is :   5  15  23  45  70
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :3

Enter the element to be deleted:12
```



```
 Enter your choice :3

Enter the element to be deleted:12
 no element with key 12
1.Create
2.Insert
3.Delete
4.Display
5.Exit

 Enter your choice :5
```

Viva questions:

1. balanced tree definition
2. Show the linked list representation of a binary tree
3. Array implementation of Binary Trees
4. What are the rules of ordered binary tree?
5. Describe the following term in a tree.
6. Describe binary tree and its property.
7. Describe full binary tree and complete binary tree.
8.Explain Extended Binary tree.
9. What are different dynamic memory allocation technique in C .
10. What are the difference between malloc() and calloc()?
11. How will you free the memory that is allocated at run time?
12. You want to insert a new item in a binary search tree. How would you do it?
13. What are threaded binary trees?

EX. NO: 12(A) COMPUTE THE SHORTEST PATH OF A GRAPH USING DIJKSTRA'S ALGORITHM

Aim:

Write a C programme to compute the shortest path of a graph using Dijkstra‘s algorithm

Descritpion

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithm.

Algorithm:

```
 DIJKSTRA_SSSP(N, COST)
/*N is the number of vertices labeled { 1, 2, 3,…N} of the weighted digraph. COST[1:N,1:N] is the
cost matrix of the graph.   If there is no edge then COST [i,j] = ∞ */
/* The  computes the cost of the shortest path from vertex 1 the source,            to every other
vertex of the weighted digraph */

T = {1};             /* Initialize T to source vertex */
for i = 2 to N do
        DISTANCE[i] = COST[1,i]; /*Initialize DISTANCE vector to the cost of the edges
                                          connecting vertex i with the source vertex 1.   If there is no
                edge then COST [1, i] = ∞ */
end
for i = 1 to N –1 do
        Choose a vertex u in V – T such that DISTANCE[u] is a minimum;
        Add u to T;
        for each vertex w in V-T do
                DISTANCE[w] = minimum(DISTANCE[w],DISTANCE[u] + COST[u,w] );
        end
end
```

end DIJKSTRA-SSSP .

Sample Input:

Enter the number f vertices: 4

Enter the adjacency matrix

```
0    1    1    1
1    0    1    0
1    1    0    1
1    0    1    0
```

Enter the starting node: 1

Distance of 0 =1

Path=0<-1

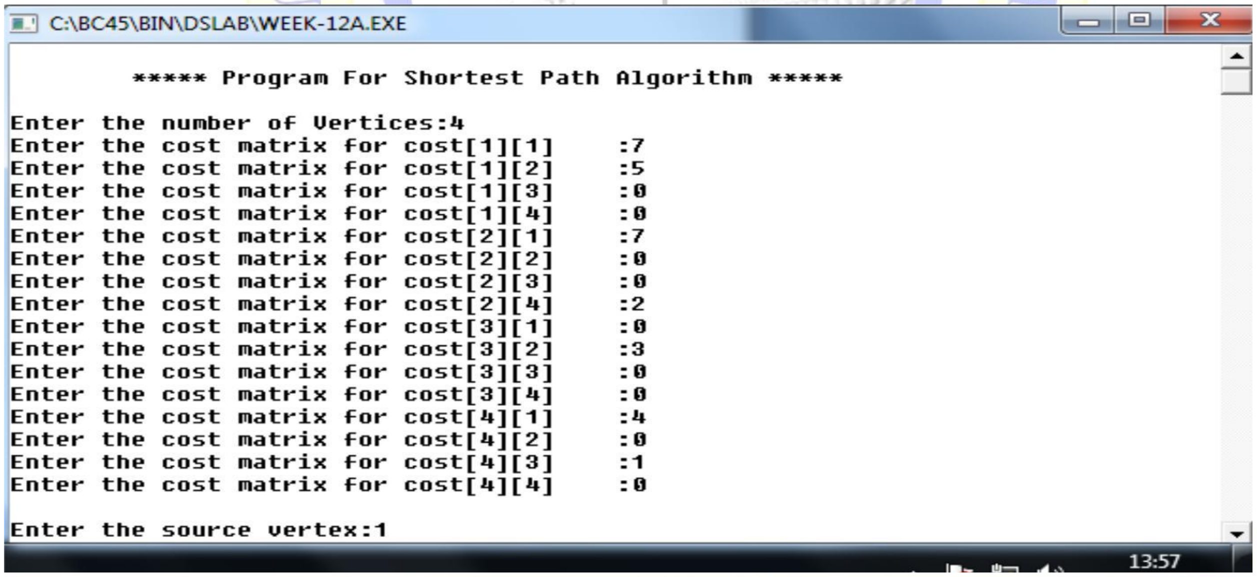
Distance of 2 =1

Path=2<-1

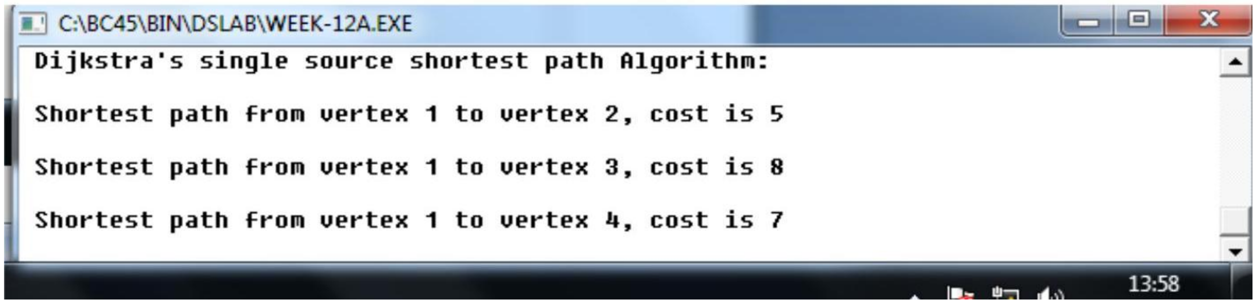
Distance of 3 =2

Path=3<-0<-1

Observed Output:



```
C:\BC45\BIN\DSLAB\WEEK-12A.EXE

        ***** Program For Shortest Path Algorithm *****

Enter the number of Vertices:4
Enter the cost matrix for cost[1][1]     :7
Enter the cost matrix for cost[1][2]     :5
Enter the cost matrix for cost[1][3]     :0
Enter the cost matrix for cost[1][4]     :0
Enter the cost matrix for cost[2][1]     :7
Enter the cost matrix for cost[2][2]     :0
Enter the cost matrix for cost[2][3]     :0
Enter the cost matrix for cost[2][4]     :2
Enter the cost matrix for cost[3][1]     :0
Enter the cost matrix for cost[3][2]     :3
Enter the cost matrix for cost[3][3]     :0
Enter the cost matrix for cost[3][4]     :0
Enter the cost matrix for cost[4][1]     :4
Enter the cost matrix for cost[4][2]     :0
Enter the cost matrix for cost[4][3]     :1
Enter the cost matrix for cost[4][4]     :0

Enter the source vertex:1
```



```
C:\BC45\BIN\DSLAB\WEEK-12A.EXE
 Dijkstra's single source shortest path Algorithm:

 Shortest path from vertex 1 to vertex 2, cost is 5

 Shortest path from vertex 1 to vertex 3, cost is 8

 Shortest path from vertex 1 to vertex 4, cost is 7
```

EX. NO: 13. MINIMUM SPANNING TREE USING WARSHALL'S ALGORITHM

Aim:

Write a C programme to find the minimum spanning tree using Warshall's Algorithm

Description:

Warshall algorithm is a dynamic programming formulation, to solve the all-pairs shortest path problem on directed graphs. It finds shortest path between all nodes in a graph. If finds only the lengths not the path. The algorithm considers the intermediate vertices of a simple path are any vertex present in that path other than the first and last vertex of that path.

Algorithm:

Algorithm: Warshall's algorithm for finding minimum spanning tree.

```
WARSHALL(N, W)
/*N is the number of vertices labeled { 1, 2, 3,…N} of the weighted digraph. W[1:N,1:N] is the
cost matrix of the graph. If there is no edge then W [i,j]= ∞. The final D matrix will store all the
shortest paths. */
for i = 1 to N do
        for j = 1 to N do
                if W[i][j] = 0 then D[i][j]= ∞
                ; else D[i][j]= W[i][j];
        end
end
for k = 1 to N do
        for i = 1 to N do
                for j = 1 to N do
```

```
                    D[i][j] = min( D[i][j], D[i][k] + D[k][j] )
                end
            end
        end
end WARSHALL.
```

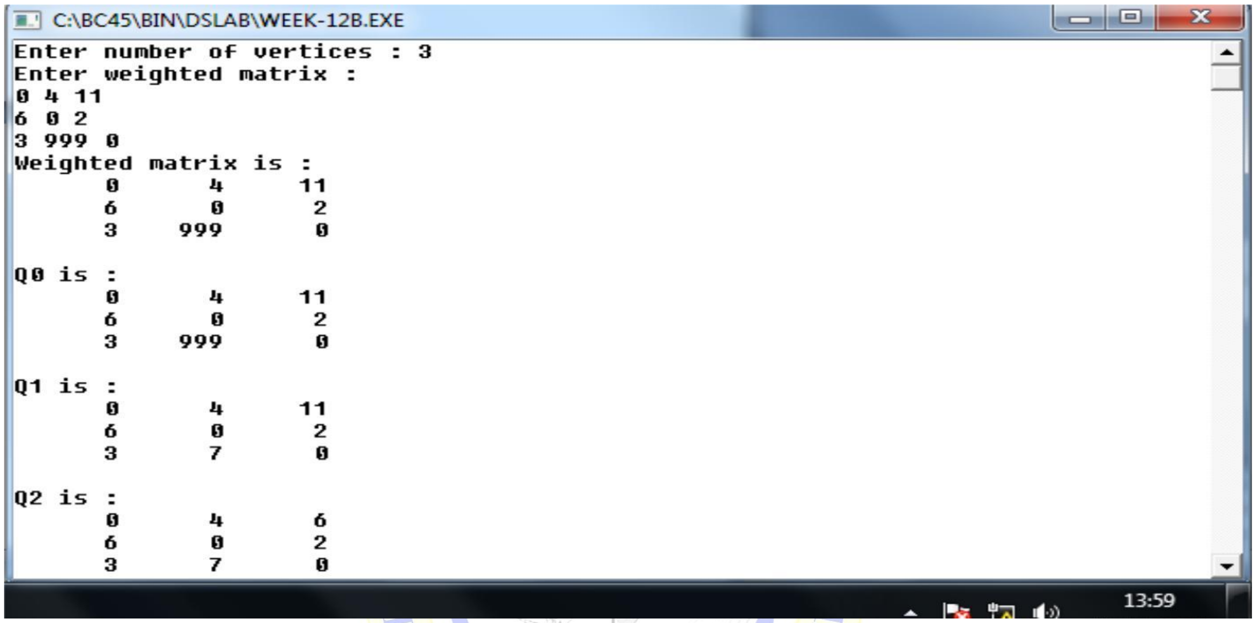
Sample Input:

```
    Enter the values of adjacency matrix
    0    3    6    0    0    0    0
    3    0    2    4    0    0    0
    6    2    0    1    4    2    0
    0    4    1    0    2    0    4
    0    0    4    2    0    2    1
    0    0    2    2    0    2    1
    0    0    0    4    1    1    0
    Minimum cost with respect to Node:0
    0    3    5    6    8    7    8
    Minimum cost with respect to Node:1
    3    0    2    3    5    4    5
    Minimum cost with respect to Node:2
    5    2    0    1    3    2    3
    Minimum cost with respect to Node:3
    6    3    0    1    2    3    3
    Minimum cost with respect to Node:4
    8    5    3    2    0    2    1
    Minimum cost with respect to Node:5
    7    4    2    3    2    0    1
```

Minimum cost with respect to Node:6

8 5 3 3 1 1 0

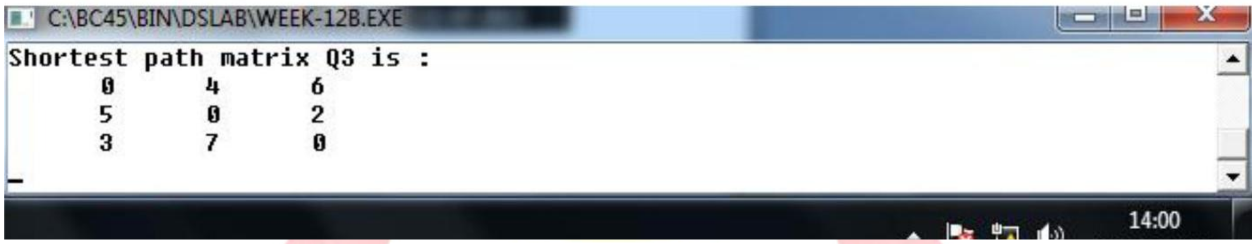
Observed Output:



C:\BC45\BIN\DSLAB\WEEK-12B.EXE

```
Enter number of vertices : 3
Enter weighted matrix :
0 4 11
6 0 2
3 999 0
Weighted matrix is :
        0       4      11
        6       0       2
        3     999       0

Q0 is :
        0       4      11
        6       0       2
        3     999       0

Q1 is :
        0       4      11
        6       0       2
        3       7       0

Q2 is :
        0       4       6
        6       0       2
        3       7       0
```

13:59



C:\BC45\BIN\DSLAB\WEEK-12B.EXE

```
Shortest path matrix Q3 is :
        0       4       6
        5       0       2
        3       7       0
_
```

14:00

Viva Questions:

1. What is a spanning Tree?
2. Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?
3. What is the runtime complexity of warshalls algorithm?
4. What is the type of the algorithm used in solving the 8 Queens problem?
5. In an AVL tree, at what condition the balancing is to be done?
6. What is the bucket size, when the overlapping and collision occur at same time?
7. Classify the Hashing Functions based on the various methods by which the key value is found.
8. What are the types of Collision Resolution Techniques and the methods used in each of the type?
9. In RDBMS, what is the efficient data structure used in the internal storage representation?
10. What is a spanning Tree?
11. Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?
12. Which is the simplest file structure? (Sequential, Indexed, Random)
13. Whether Linked List is linear or Non-linear data structure?

EX. NO: 13 CIRCULAR QUEUE USING ARRAYS

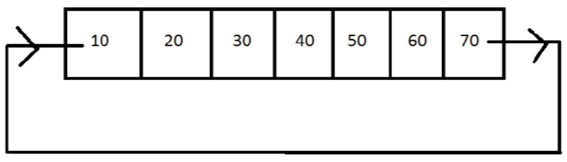
Aim:

Write a C program that implements Circular Queue and its operations using arrays.

Description:

In a standard queue data structure re-buffering problem occurs for each dequeue operation. To solve this problem by joining the front and rear ends of a queue to make the queue as a circular queue
Circular queue is a linear data structure. It follows FIFO principle.

- In circular queue the last node is connected back to the first node to make a circle.
- Circular linked list fallow the First In First Out principle
- Elements are added at the rear end and the elements are deleted at front end of the queue
- Both the front and the rear pointers points to the beginning of the array.
- It is also called as "Ring buffer".
- Items can inserted and deleted from a queue in O(1) time.
  .



Circular Queue

Algorithm:

In circular queue, the insertion of a new element is performed at the very first location of the queue if the last location of the queue is full, in which the first element comes just after the last element.

Insertion and Deletion:

Rear = (rear+1)%Maxsize

Alogarithm Steps:

```
Step 1: create and set the variables front,rear,MAXSIZE,cq[]
Step 2: Read the circular queue opeartion type.
Step 3: If operation type is Insertion below steps are executed.
    1. Assign rear=rear%MAXSIZE.
    2. if front equal to (rear+1)%MAXSIZE then display queue is overflow.
    3. if front equal to -1 then assign front=rear=0.
    4. Otherwise assign rear=(rear+1)%MAXSIZE and read queue data .
    5. Assign cq[rear] as data.(i.e. cq[rear]=data).
Step 4: If operation type is Deletion below steps are executed.
    1. Check front=-1 then display queue is underflow.
    2. Set temp as cq[front] (i.e. temp=ca[front]).
    3. Check front equal to rear if it is true then assign front=rear=-1(Move the   front to begining)
    4. Assign front=(front+1)%MAXSIZE.
```

Sample Input:

```
MAIN MENU
1. INSERTION
```

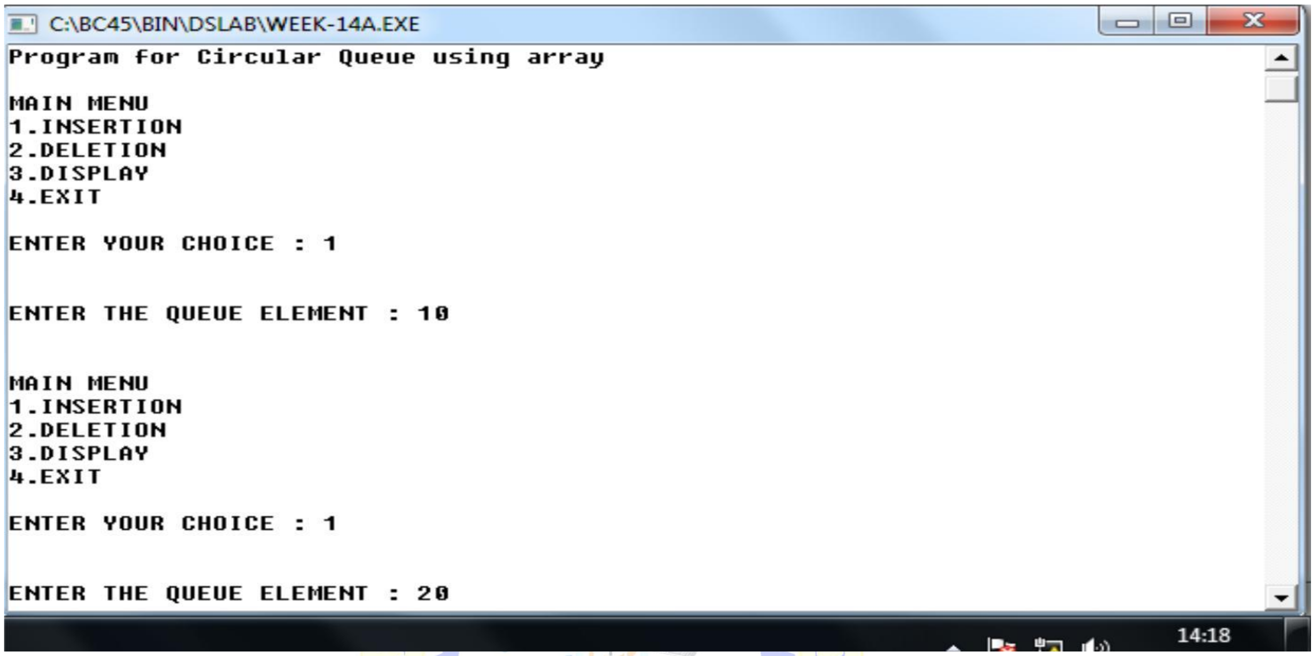
```
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

ENTER THE QUEUE ELEMENT : 10

REAR=0     FRONT=0

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

ENTER THE QUEUE ELEMENT : 20

REAR=1     FRONT=0

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

ENTER THE QUEUE ELEMENT : 30

REAR=2     FRONT=0

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT
```

```
ENTER YOUR CHOICE : 1

ENTER THE QUEUE ELEMENT : 40

REAR=3     FRONT=0

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

ENTER THE QUEUE ELEMENT : 50

REAR=4    FRONT=0

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

ENTER THE QUEUE ELEMENT : 60

CIRCULAR QUEUE IS OVERFLOW.

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 2
```

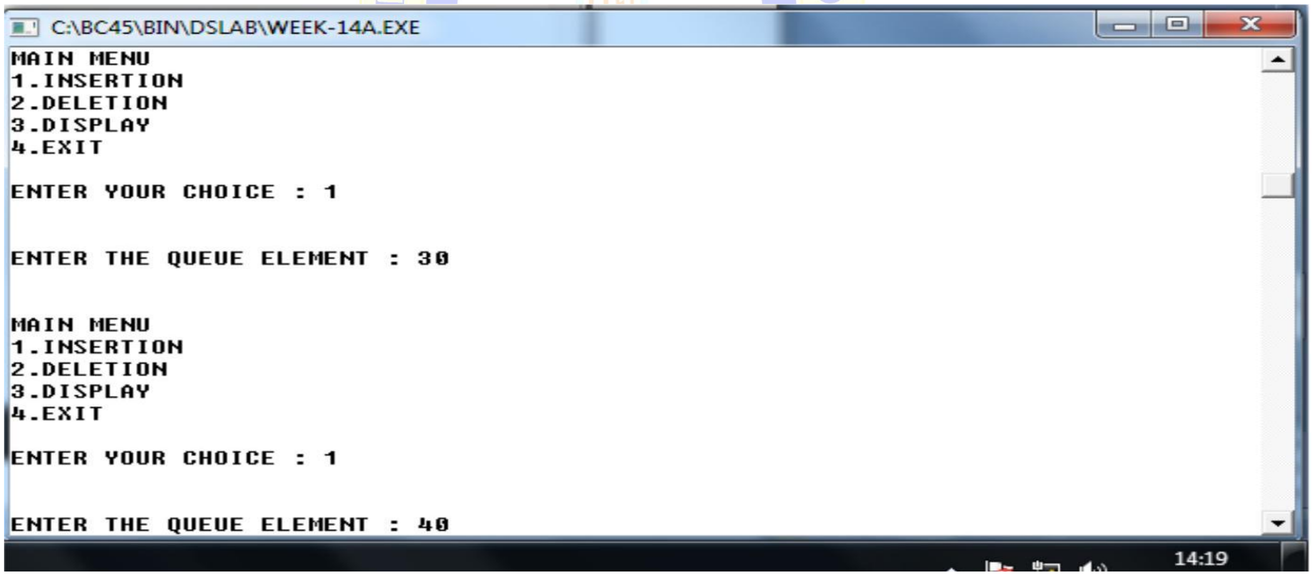
```
DELETED ELEMENT FROM QUEUE IS : 10

REAR =4    FRONT=1

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 2

DELETED ELEMENT FROM QUEUE IS : 20

REAR =4    FRONT=2

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 2

DELETED ELEMENT FROM QUEUE IS : 30

REAR =4    FRONT=3

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT
```

```
ENTER YOUR CHOICE : 2

DELETED ELEMENT FROM QUEUE IS : 40

REAR =4    FRONT=4

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 2

DELETED ELEMENT FROM QUEUE IS : 50

REAR =-1    FRONT=-1

MAIN MENU
1. INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 2

CIRCULAR QUEUE IS UNDERFLOW
```

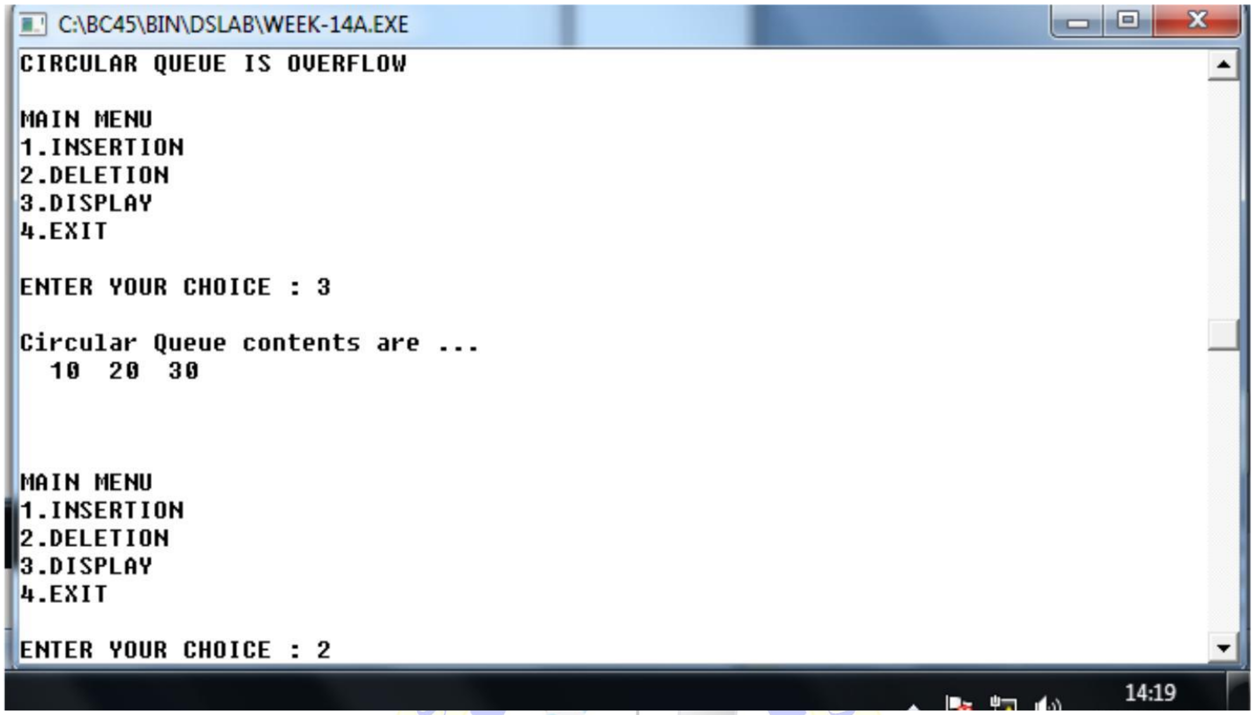
Obserevd Output:



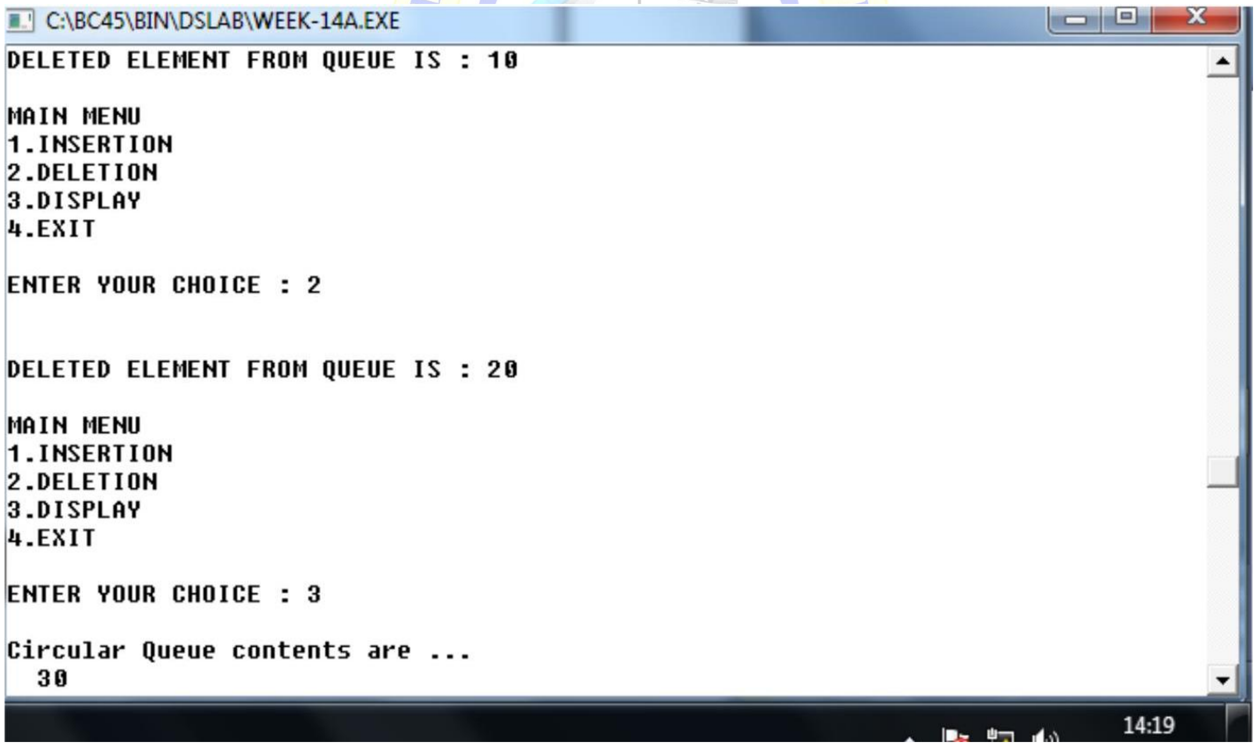
```
Program for Circular Queue using array

MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 1


ENTER THE QUEUE ELEMENT : 10


MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 1


ENTER THE QUEUE ELEMENT : 20
```



```
MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 1


ENTER THE QUEUE ELEMENT : 30


MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 1


ENTER THE QUEUE ELEMENT : 40
```



```
CIRCULAR QUEUE IS OVERFLOW

MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 3

Circular Queue contents are ...
  10  20  30



MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 2
```



```
DELETED ELEMENT FROM QUEUE IS : 10

MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 2


DELETED ELEMENT FROM QUEUE IS : 20

MAIN MENU
1.INSERTION
2.DELETION
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE : 3

Circular Queue contents are ...
  30
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Viva questions:

1. Describe stack operation.

2. Describe queue operation.

3.Discuss how to implement queue using stack.

4.Explain stacks and queues in detail.

5.What are priority queues?

6. What is a circular singly linked list?

EX. NO: 14 CREATION & OPERATIONS ON DLL

Aim:

Write a C Program to perform various operations such as creation, insertion, deletion, search and display on doubly link lists (DLL).

Descritpion:

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **Prev** − Each link of a linked list contains a link to the previous link called Prev.
- **LinkedList** − A Linked List contains the connection link to the first link called First and to the last link called Last.

Algorithm for DLL:

```
1. Start
2. Read ch
3. Repeat while
   (1) Do
     Display all the cases
```

```
4.  Read
    switch(ch)
    Do
    Case1:append              (head);
    Case2:forward_display(head);
    Case3:insert_after(head);
    Case4:insert_before(head);
    Case5: delnode(&head);
    Case6:search(node);
    Case7:destroy(head);
    Case8:exit;
    Default;
5.  Endswitch
6.  End while
7.  Stop.
```

Algorithm createnode()

Input: null

Output: a node is created by using malloc function and the node is returned.

```
    1.  Start
    2.  Create a node ―new‖ using malloc
//new=(struct node*)malloc (sizeof(struct node));
    3.  Read the data into node ‗X‘.
    4.  Assign new(llink)=null;
                New(rlink)=null;
    5.  Stop.
```

Algorithm append(node)

Input: the first created node

Output: the node created will be added to the existing dist and will be returned.

```
1. Start
2. Assign new=createnode();
3. If(*h=null)
   {
   *h=new;
   Return;
   }
4. Temp=*h;
5. Repeat
   while(temp(rlink)!=null)
   Temp=temp(rlink);
   Temp=(rlink)=new;
   New(llink)=temp;
6. End while
7. Stop.
```

Algorithm forward (node)

Input: a node is inserted

Output: prints the contents of linked list in forward order.

```
1. Start
2. Repeat
   while(p!=null) Do
```

```
        Display  p(data),
       and P=p(rlink)
3.  End while
4.  Display the contents of the list.
5.  Stop
```

Algorithm insert_after(node,*h)

Input: an node is inserted in the list in forward order

```
1.  Start
2.  Repeat while(p@= null) Do
        Display  p(data),
       and P=p(rlink)
3.  End while
4.  Display the contents of the list
5.  Stop.
```

Algorithm insert_after (node, *h)

Input: an node is inserted in the list and h is a pointer declared.

Output: a new node is inserted after a given node of linked list

```
1.  Start
2.  Declare _k' of integer type
3.  If (*h=null) Return
4.  Read k
5.  Temp =*h; Return;
    End
6.  Temp=*h;
7.  Repeat while (temp!=null and temp(data)!=k) Do
```

```
      Temp=temp(rlink) End
8.  If(temp!=null) Do
      New=create node();
     New(rlink)=temp;
     New(llink)=temp(llink);
     New(llink)=rlink(new);
     Temp->llink=new;
9.  End if
10. End
11. Stop
```

Algorithm delete (node)

Input: a node which is to be deleted is given as input and h is declared as pointer variable.

Output: a node is deleted from the list

```
1.  Start
2.  Declare _k‘ of integer type
3.  If(*h=null) return
4.  Read K
5.  If ((*h)data=k) Do
    Temp=*h;
    *h=(*h)rlink;
    (*h)->llink=null;
 6. free(temp)
 7. return
 8. temp= *h;
 9.repeat while (temp!=null and temp(data)=k)
    Do
```

```
            Temp=temp(rlink)
        End
            If (temp!=null)
            Do
                Temp(rlink)(llink)=temp(llink);
                Temp(llink)(rlink)=temp(rlink);
      10. display the node and and the data
      11.  free the memory of the node
      12. end while
      13. stop
```

Algorithm search(node)

Input: a node is inserted and a pointer variable h is declared

Output: display the searched node with particular data

```
    1.  Start
    2.  Declare _K‘ of integer type
    3.  If (h=null)return
    4.  Read k
    5.  Temp=h
    6.  Repeat while (temp!=null and temp(data)!=k)do
    7.  Temp=temp->rlink
        If(temp=null)
    8.  Display  node  does  not
        Node exist
    9.  Stop
```

Algorithm deleting entire list (node)

Input: node

Output: deletes entire list and displays the list

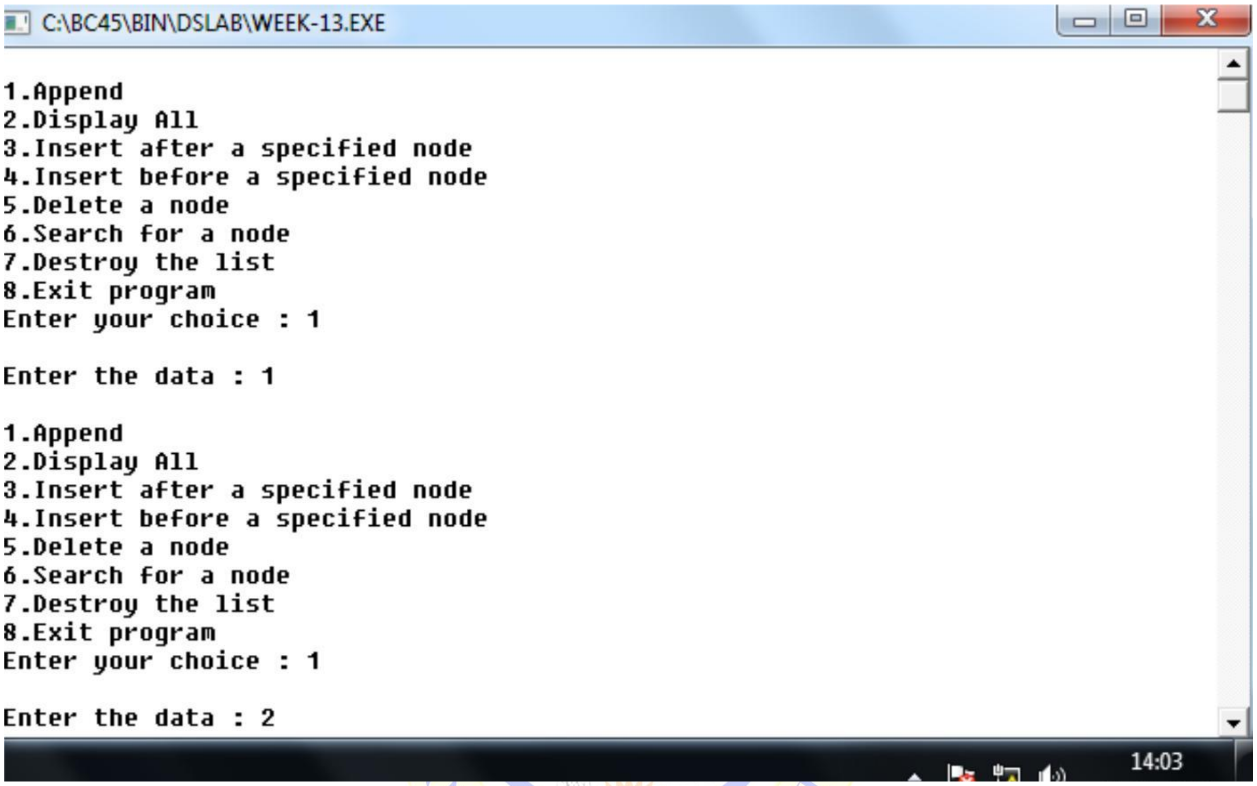
```
1. Start
2. If(*h=null)return
3. Repeat        while
   (*h!=nul)do
   P=(*h)rlink;
   Free(*h);
4. *h=p;
5. Display linked list is destroyed
6.   stop
```

Sample Input:

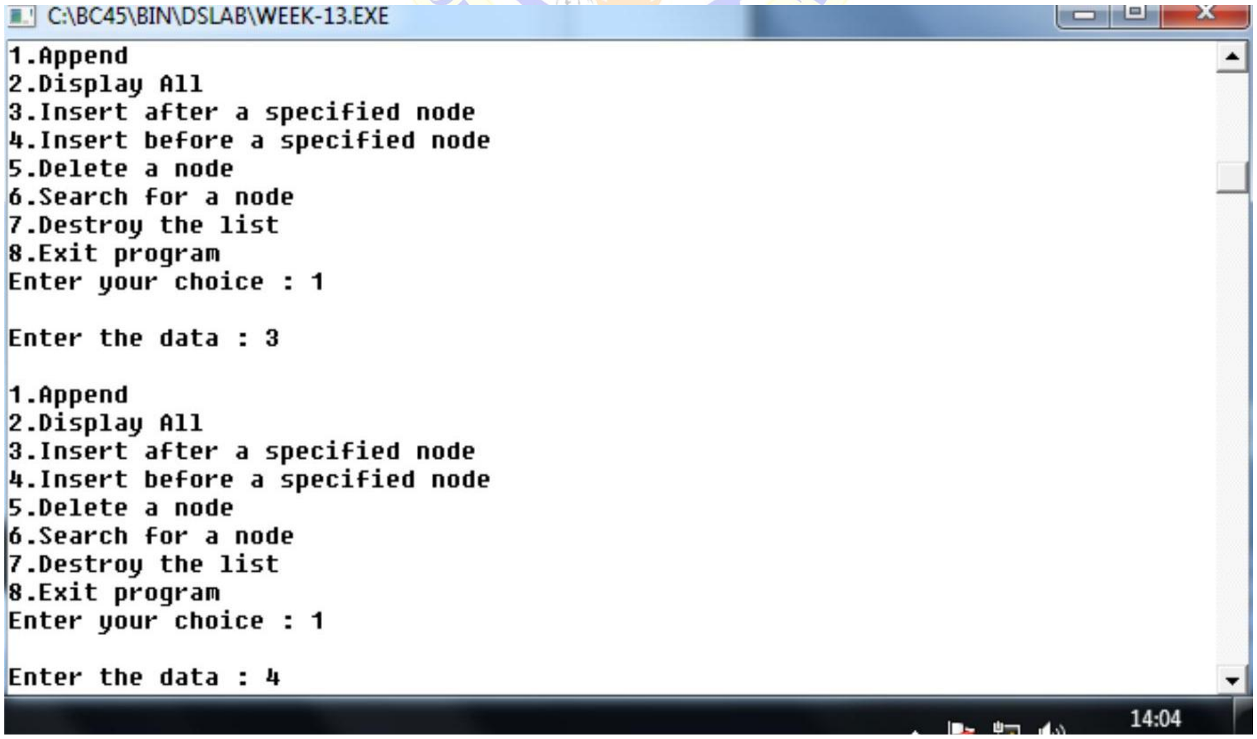
```
1.Add at beginning
2.Add at location
3. Add at end
4.Deletion
5.Display
6. exit
    Enter your choice : 1
    Enter the value : 33
6.  Add at beginning
7.  Add at location
8.  Add at end
9.  Deletion
10. Exit
```

```
        Enter your choice : 1
        Enter the value : 38
     Add at beginning
    6.  Add at location
    7.  Add at end
    8.  Deletion
    9.  Display
    10. Exit
      Enter your choice : 1
      Enter the value : 40
    7.  Add at beginning
    8.  Add at location
    9.  Add at end
    10. Deletion
    11. Display
    12. Exit
  Enter your choice : 5
The elements in single linked list: 33 38 40
```

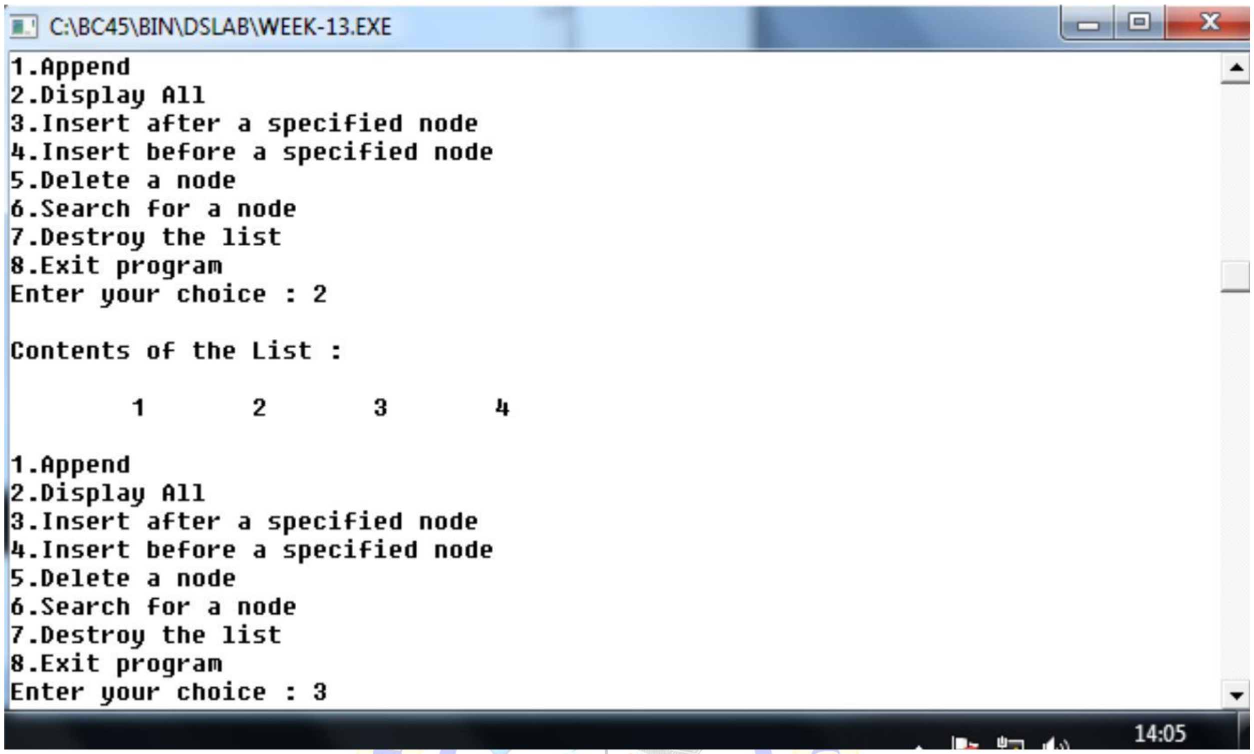
Observed Output;



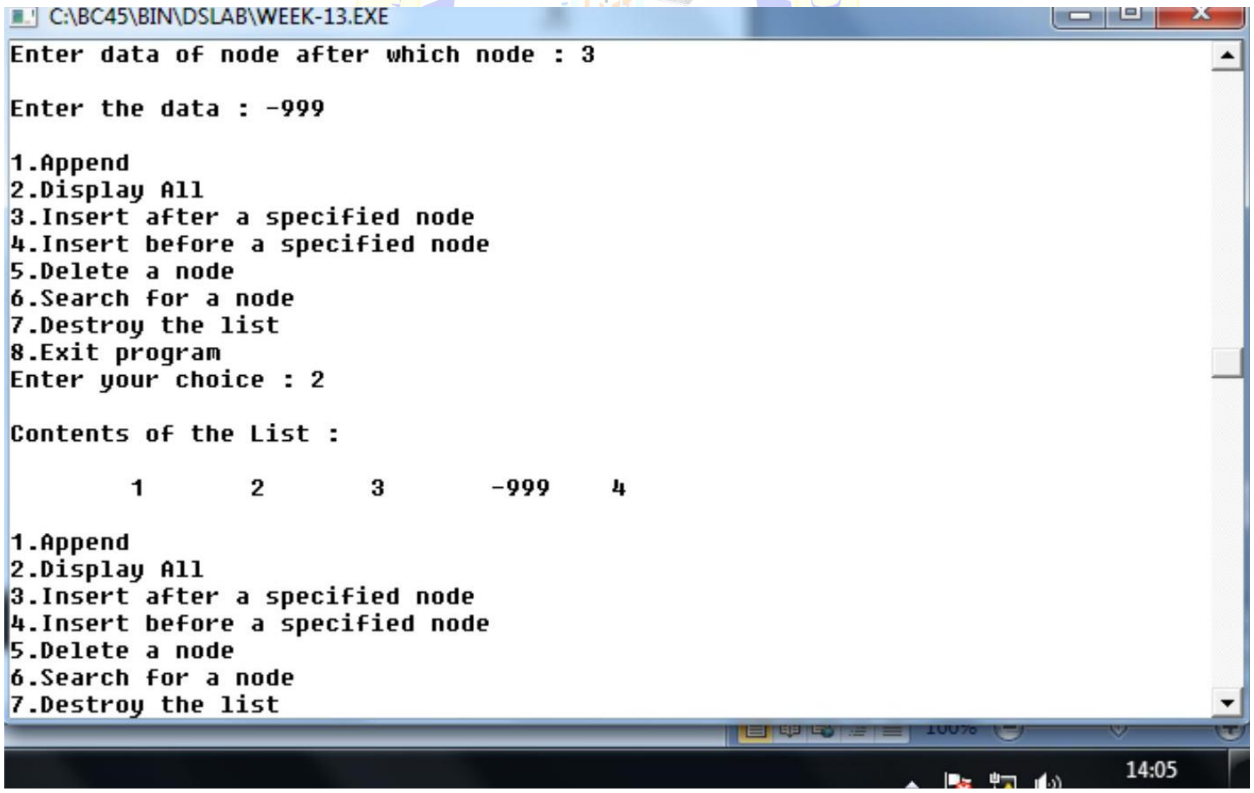
```
C:\BC45\BIN\DSLAB\WEEK-13.EXE

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 1

Enter the data : 1

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 1

Enter the data : 2
```



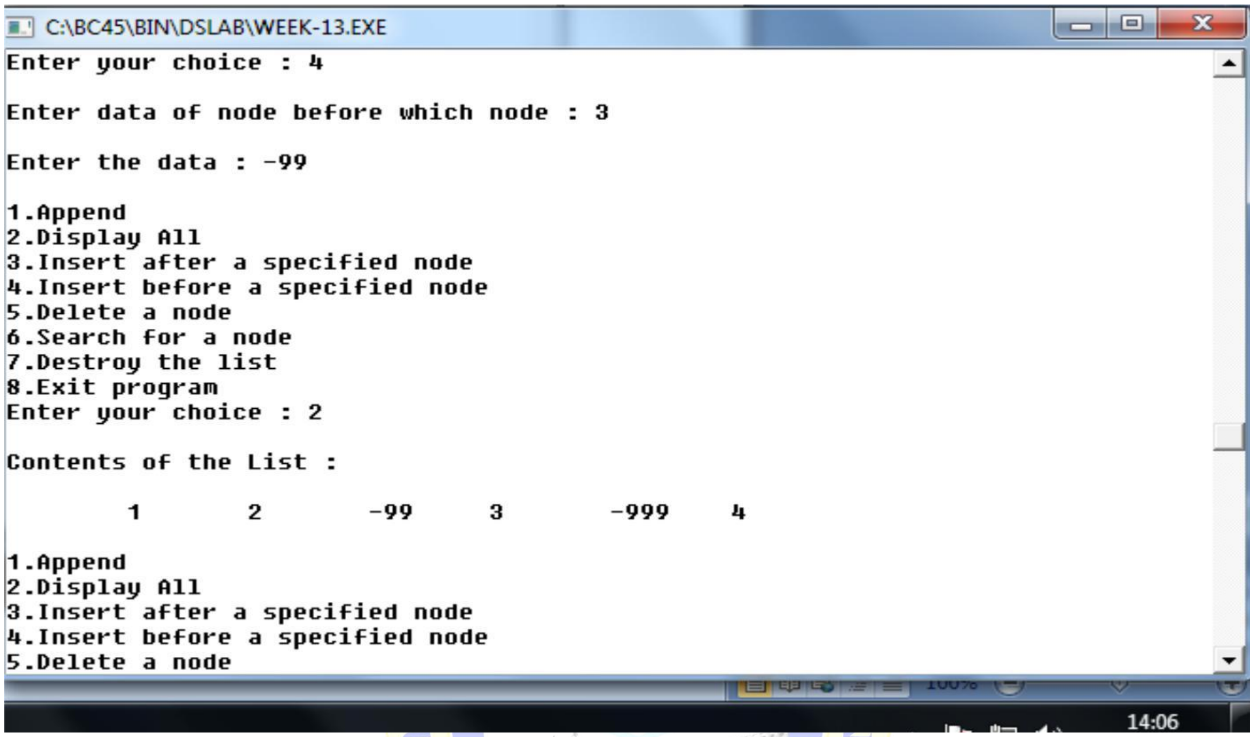
```
C:\BC45\BIN\DSLAB\WEEK-13.EXE
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 1

Enter the data : 3

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 1

Enter the data : 4
```


C:\BC45\BIN\DSLAB\WEEK-13.EXE

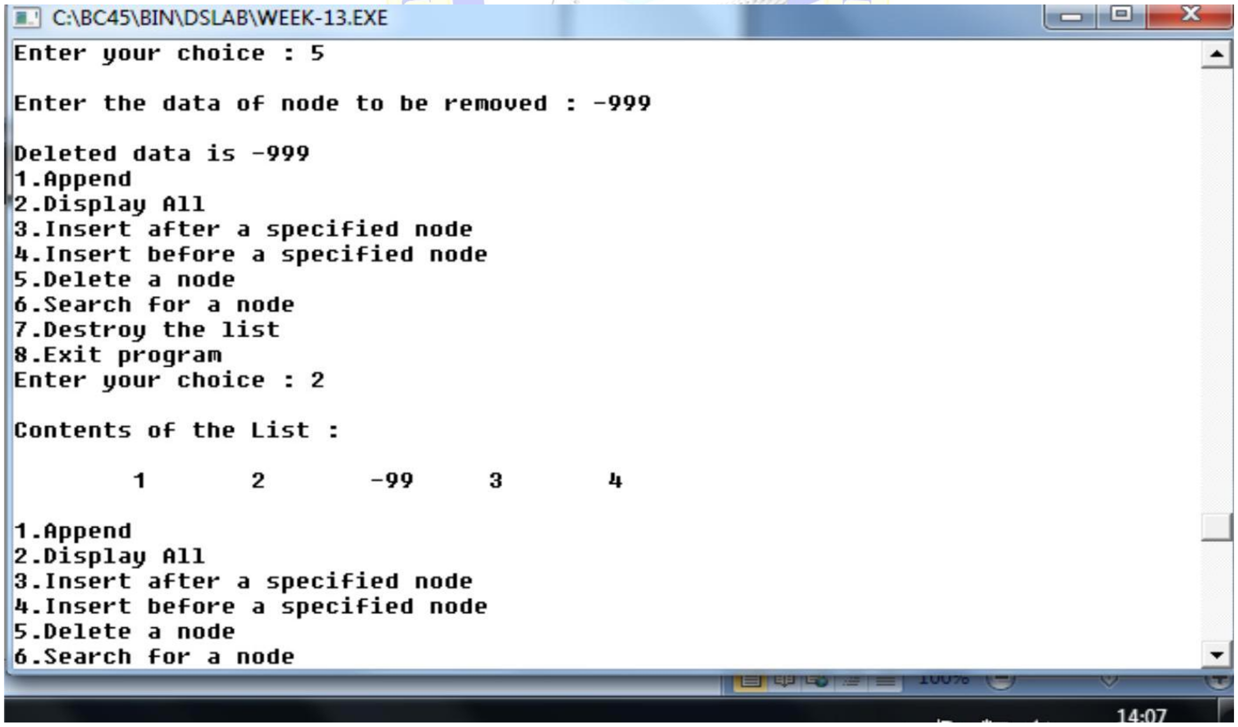
```
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 2

Contents of the List :

        1       2       3       4

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 3
```


C:\BC45\BIN\DSLAB\WEEK-13.EXE

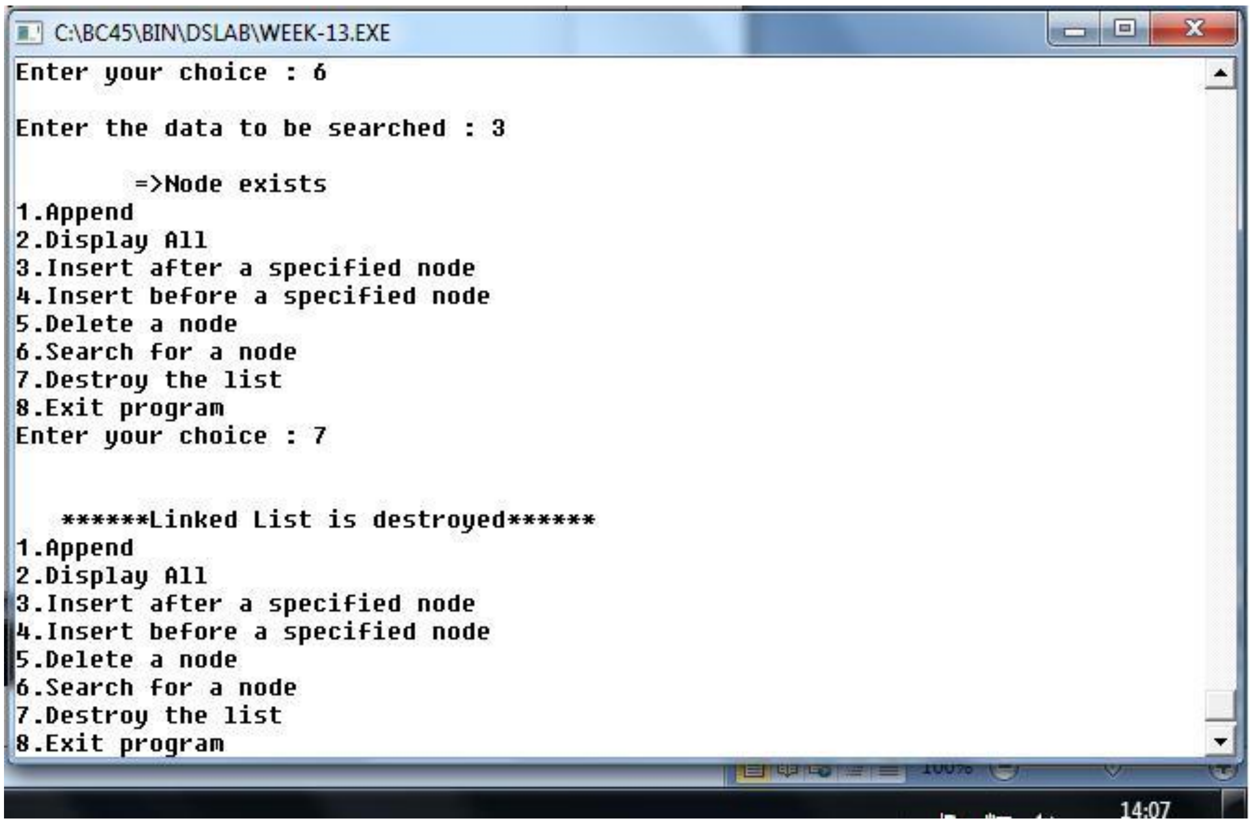
```
Enter data of node after which node : 3

Enter the data : -999

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 2

Contents of the List :

        1       2       3       -999    4

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
```



```
Enter your choice : 4

Enter data of node before which node : 3

Enter the data : -99

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 2

Contents of the List :

        1       2       -99     3       -999    4

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
```



```
Enter your choice : 5

Enter the data of node to be removed : -999

Deleted data is -999
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 2

Contents of the List :

        1       2       -99     3       4

1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
```


C:\BC45\BIN\DSLAB\WEEK-13.EXE

```
Enter your choice : 6

Enter the data to be searched : 3

        =>Node exists
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
Enter your choice : 7


   ******Linked List is destroyed******
1.Append
2.Display All
3.Insert after a specified node
4.Insert before a specified node
5.Delete a node
6.Search for a node
7.Destroy the list
8.Exit program
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Viva questions:

1.Describe the steps to insert data into a singly linked list.

2. Explain how to reverse singly link list.

3.Define circular linked list.

4.Define circular linked list.

5. Describe linked list.

6. Linked lists demystified

7.How to represent a linked list node?

8. How do you traverse a linked list?

9. How to insert a node at the beginning of the list?

10.How to insert a node at the end of the list?

11.How to insert a node in a random location in a list?

12.How to delete a node at a specific location?