Meerut Institute of Engineering & Technology

N.H. 58, Delhi-Roorkee Highway, Baghpat Road Bypass Crossing,
Meerut-250005, UP(India)

Department of Computer Science & Engineering

B.Tech (Session 2019-2020)
Odd-Semester

DATA STRUCTURE LAB
(KCS-351)

L	T	P
0	0	2

Faculty Name

Ms. Shaili Singhal

MEERUT INSTITUTE OF ENGINEERING AND TECHNOLOGY, MEERUT

Computer Science & Engineering Department

B. Tech. 2nd Year/ 3rd Semester

Data Structure Lab (KCS-351)

L	T	P
0	0	2

1. Write a C Program to implement insertion and deletion operation in an array using function.
2. Write Program in C Using 2D array to implement addition and multiplication of two 2D arrays.
3. Write Program in C Using 2D array to implement transpose of 2D array.
4. Write Program in C for implementation of Linear Search.
5. Write Program in C for implementation of Binary Search.
6. Write a C Program to implement Bubble Sorting.
7. Write a C Program to implement Selection Sorting.
8. Write a C Program to implement Insertion Sorting.
9. Write Program in C for implementation of Stack Using Array (PUSH, POP & Traversing).
10. Write Program in C for implementation of Factorial using recursion.
11. Write Program in C for implementation of Tower of Hanoi using recursion.
12. Write Program in C for implementation of Merge sort.
13. Write Program in C for implementation of Quick Sort.
14. Write Program in C for implementation of simple Queue Using Array.
15. Write Program in C for implementation of Circular Queue Using Array.
16. Write a C program to implement operations of singly Linked List:
    - Creation, insertion, and deletion
17. Write a C program to implement operations of Doubly Linked List:
    - Creation, insertion, and deletion
18. Write Program in C for implementation of Stack Using Linked List (PUSH, POP & Traversing).
19. Write Program in C for implementation of simple Queue Using Linked List.
20. Write Program in C for implementation of Addition of two polynomials using Linked List.
21. Write a C program to implement binary tree traversal using Linked List.
22. Write Program in C for implementation of Insertion and Deletion in BST Using Linked List.
23. Write Program in C for implementation of Heap Sort.
24. Write Program in C for Graph Implementation of BFS & DFS Using Linked List.

Value Addition Programs:

1. Write Program in C for implementation of Circular Queue Using Linked List.
2. A graph is abstractly a collection of vertices which are labeled by non-negative integers, and a collection of edges. A graph called an undirected graph if we talk of merely the presence of an edge between vertices i and j, rather than its direction. For example, the following is a graph:

In this problem, you are given the edges in an undirected graph. An edge is a pair of non-negative integers (i, j) which indicates that the vertex i is connected to vertex j by an edge. Afterwards, you will be given a vertex number n. You have to output the list of vertices which are connected n by an edge, in the order in which the edges were input.

Input

You are given the following.

1. The first line contains an integer, E, between 1 and 1000.
2. This is followed by E lines, where each containing a pair of numbers i and j where i and j are both non-negative integers <=34,000. No edge will be listed more than once.
3. The last line contains a non-negative integer n <= 34,000. n is assured to be a vertex listed in one of the E lines in part (2).

Output

You have to output the list of nodes to which n has an edge, in the order in which the edges were input, one line for each vertex.

Sample Test Cases

	Input	Output
Test Case 1	4 1 2 2 3 3 4 4 5 4	3 5
Test Case 2	10 1 2 1 3 1 4 2 3 2 4 5 1 5 2 5 3 5 4 6 1 1	2 3 4 5 6

Program 1: Write a C Program to implement insertion and deletion operation in an array using function.

Algorithm (Insertion)

Let **LA** be a Linear Array (unordered) with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm where ITEM is inserted into the K^{th} position of LA −

1. Start
2. Set J = N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop

Algorithm (Deletion)

Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm to delete an element available at the K^{th} position of LA.

1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop

Program 2: Write Program in C using 2D array to implement addition and multiplication of two 2D arrays.

Matrix Multiplication Algorithm:

1. Start
2. Declare variables and initialize necessary variables
3. Enter the element of matrices by row wise using loops
4. Check the number of rows and column of first and second matrices
5. If number of rows of first matrix is equal to the number of columns of second matrix, go to step 6. Otherwise, print matrix multiplication is not possible and goes to step 3.
6. Multiply the matrices using nested loops.
7. Print the product in matrix form as console output.
8. Stop

Matrix Addition Algorithm:

Start the program

1. Read the total number of rows &columns for the matrix

2. Read the elements for A Matrix and store it in to two dimensional arrays.

3. Read the elements for B Matrix and store it into other two dimensional array.

4. Add the elements of 2 matrixes namely A& B through the nested for Loops and store the result into a resultant 2-dimesional array C.

5. Display the result Stop the program.

Program 3: Write Program in C using 2D array to implement transpose of 2D array.

Algorithm:

Start the program

1. Read the total number of rows &columns for the matrix

2. Read the elements for A Matrix and store it into two dimensional arrays.

3. Transpose of matrix A is computed 4. display the result.

Stop the program

Program 4 : Write Program in C for implementation of Linear Search and Binary Search.

Algorithm:

Start the program

1. Read the number of values (n).
2. Initialize the iterative variable i=0.
3. Repeat the steps 4&5until no. of values minus 1(n-1).
4. Read the elements in an array.
5. Increment the iterative variable.
6. Assign the first element of an array to the small and big variables.
7. Repeat the steps 8 to 10, until no of values minus 1 (n-1).
8. If the array element is greater than big element, assign the element to the big variable. Otherwise go to step 10.
9. If the array element is smaller than small element, assign the element to the small variable. Otherwise go to step 10.
10. Increment the iterative variable.
11. Display the biggest element in the list.
12. Display the smallest element in the list.

Stop the program

```
Procedure binary_search

   A ← sorted array

   n ← size of array

   x ← value to be searched


   Set lowerBound = 1

   Set upperBound = n


   while x not found

      if upperBound < lowerBound

         EXIT: x does not exists.


      set midPoint = lowerBound + ( upperBound - lowerBound ) / 2


      if A[midPoint] < x

         set lowerBound = midPoint + 1


      if A[midPoint] > x

         set upperBound = midPoint - 1


      if A[midPoint] = x

         EXIT: x found at location midPoint

   end while


end procedure
```

Program 5: Write a C Program to implement Bubble Sorting, Selection Sorting and Insertion Sorting.

```
INSERTION-SORT(A)
   for i = 1 to n
         key ← A [i]
         j ← i – 1
          while j > = 0 and A[j] > key
                 A[j+1] ← A[j]
                 j ← j – 1
         End while
         A[j+1] ← key
  End for
```

```
procedure selection sort

   list  : array of items

   n     : size of list


   for i = 1 to n - 1

   /* set current element as minimum*/

      min = i


      /* check the element to be minimum */


      for j = i+1 to n

         if list[j] < list[min] then

            min = j;

         end if

      end for


      /* swap the minimum element with the current element*/

      if indexMin != i  then

         swap list[min] and list[i]

      end if

   end for


end procedure
```

```
begin BubbleSort(list)


   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for


   return list


end BubbleSort
```

Program 6: Write Program in C for implementation of Stack Using Array (PUSH, POP & Traversing).

```
begin procedure push: stack, data


   if stack is full
      return null
   endif


   top ← top + 1
   stack[top] ← data


end procedure
begin procedure pop: stack


   if stack is empty
      return null
   endif


   data ← stack[top]
   top ← top - 1
   return data


end procedure
```

Program 7: Write Program in C for implementation of Factorial using recursion and Tower of Hanoi using recursion.

```
Step 1: Start
Step 2: Read number n
Step 3: Call factorial(n)
Step 4: Print factorial f
Step 5: Stop
factorial(n)
Step 1: If n==1 then return 1
Step 2: Else
f=n*factorial(n-1)
Step 3: Return f
```

```
START

Procedure Hanoi(disk, source, dest, aux)



   IF disk == 1, THEN

      move disk from source to dest

   ELSE

      Hanoi(disk - 1, source, aux, dest)     // Step 1

      move disk from source to dest          // Step 2

      Hanoi(disk - 1, aux, dest, source)     // Step 3

   END IF


END Procedure

STOP
```

Program 8: Write Program in C for implementation of Merge sort and Quick Sort.

```
MergeSort(arr[], l,  r)
If r > l
```

```
 1. Find the middle point to divide the array into two halves:
        middle m = (l+r)/2
 2. Call mergeSort for first half:
        Call mergeSort(arr, l, m)
 3. Call mergeSort for second half:
        Call mergeSort(arr, m+1, r)
 4. Merge the two halves sorted in step 2 and 3:
        Call merge(arr, l, m, r)
```

```
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
 partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];

    i = (low - 1)  // Index of smaller element

    for (j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
```

```
    return (i + 1)
}
```

Program 9: Write Program in C for implementation of simple Queue Using Array.

Algorithm for en-queue operation

```
procedure enqueue(data)


   if queue is full
      return overflow
   endif


   rear ← rear + 1
   queue[rear] ← data
   return true


end procedure
```

Algorithm for dequeue operation

```
procedure dequeue


   if queue is empty
      return underflow
   end if


   data = queue[front]
   front ← front + 1
   return true


end procedure
```

Program 10: Write Program in C for implementation of Circular Queue Using Array.

INSERT-ITEM(QUEUE, FRONT, REAR, MAX, COUNT, ITEM)

This algorithm is used to insert or add item

into circular queue.

1. If (COUNT = MAX) then
    a. Display “Queue overflow”;
    b. Return;
2. Otherwise
    a. If (REAR = MAX) then
        i. REAR := 1;
    b. Otherwise
        i. REAR := REAR + 1;
    c. QUEUE(REAR) := ITEM;
    d. COUNT := COUNT + 1;
3. Return;

REMOVE-ITEM(QUEUE, FRONT, REAR, COUNT, ITEM)

This algorithm is used to remove or delete item

from circular queue.

1. If (COUNT = 0) then
    a. Display “Queue underflow”;
    b. Return;
2. Otherwise
    a. ITEM := QUEUE(FRONT)l
    b. If (FRONT =MAX) then
        i. FRONT := 1;

c. Otherwise

i. FRONT := FRONT + 1;

d. COUNT := COUNT + 1;

3. Return;

EMPTY-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,EMPTY)

This is used to check queue is empty or not.

1. If(COUNT = 0) then

a. EMPTY := true;

2. Otherwise

a. EMPTY := false;

3. Return ;

FULL-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,FULL)

This algorithm is used to check queue is full or not.

1. If (COUNT = MAX) then

a. FULL := true;

2. Otherwise

a. FULL := false;

3. Return ;

Lab Outcome

1. Students will able to select appropriate data structures as applied to specified problem definition.

2. Students will able to implement operations like searching, insertion, and deletion, traversing mechanism etc. on various data structures.

3. Students will be able to implement linear and Non-Linear data structures.

4. Students will be able to implement appropriate sorting/searching technique for given problem.

5. Students will be able to design advance data structure using Non-linear data structure.

6. Students will be able to determine and analyze the complexity of given Algorithms.