**Data File Structures**

**Lab File**



Submitted By – Shubham Goswami

Roll No - 14

**INDEX**

Q 1. Write a C Program to multiply two square matrices of the given order

Q 2. Write a C Program to implement Linear Search

Q 3. Write a C Program to implement Bubble Sort

Q 4. Write a C Program to transpose a square matrix of size nxn without using any auxiliary matrix

Q 5. Write a C Program to implement a Stack and its operations using an array

Q 6. Write a C Program to implement a Queue using an array

Q 7. Write a C Program to find factorial of a given number n using:

(i) Recursion

(ii) Iteration

Q 8. Write a C Program to find Fibonacci term of a given index n using

(i) Recursion

(ii) Iteration

Q 9. Write a C Program to implement Binary Search

Q 10. Write a C Program to implement Insertion Sort

Q 11. Write a C Program to implement Selection Sort

Q 12. Write a C Program to

(i) To create a Linked List of Integers

(ii) To insert an integer at the end of the list

(iii)To delete first element of the list

(iv)To reverse the links of the given Linked List

Q 13. Write a C Program to implement Merge Sort

Q 14. Write a C Program to implement Quick Sort

Q 15. Write a C Program to implement a Queue using two Stacks

Q 16. Write a C Program to

(i) To input a Binary Tree from the user

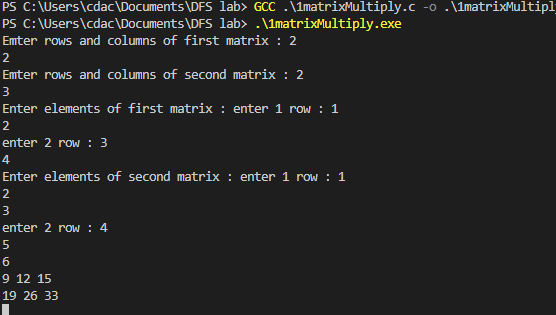
(ii) Preorder Traversal

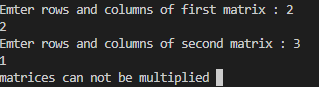
(iii)Inorder Traversal

(iv)Postorder Traversal

1. Write a program to show multiplication of 2 matrices in C.
2. #include<stdio.h>
3. #include<stdlib.h>
4. #define R1 4
5. #define C1 4
6. #define R2 4
7. #define C2 4
8. voidmulMat(intmat1[][C1], intmat2[][C2]) {
9. intrslt[R1][C2];
10. printf("Multiplication of given two matrices is:\n\n");
11. for (inti = 0; i<R1; i++) {
12. for (intj = 0; j<C2; j++) {
13. rslt[i][j] = 0;
14. for (intk = 0; k<R2; k++) {
15. rslt[i][j] += mat1[i][k] \* mat2[k][j];
16. }
17. printf("%d\t", rslt[i][j]);
18. }
19. printf("\n");
20. }
21. }
22. intmain(void) {
23. intmat1[R1][C1] = {
24. {1, 1, 1, 1},
25. {2, 2, 2, 2},
26. {3, 3, 3, 3},
27. {4, 4, 4, 4}
28. };
29. intmat2[R2][C2] = {
30. {1, 1, 1, 1},
31. {2, 2, 2, 2},
32. {3, 3, 3, 3},
33. {4, 4, 4, 4}
34. };
35. if (C1 != R2) {
36. printf("The number of columns in Matrix-1 must be equal to the number of rows in "
37. "Matrix-2\n");
38. printf("Please update MACROs value according to your array dimension in "
39. "#define section\n");
40. exit(EXIT\_FAILURE);
41. }
42. mulMat(mat1, mat2);
43. return0;
44. }

**OUTPUT :**

****

****

2. Write a Program in C to transpose a square matrix .

#include<stdio.h>

#define N 4

voidtranspose(intA[][N], intB[][N])

{

    inti, j;

    for (i = 0; i<N; i++)

        for (j = 0; j<N; j++)

            B[i][j] = A[j][i];

}

intmain()

{

    intA[N][N] = { {1, 1, 1, 1},

                    {2, 2, 2, 2},

                    {3, 3, 3, 3},

                    {4, 4, 4, 4}};

    intB[N][N], i, j;

    transpose(A, B);

    printf("Result matrix is \n");

    for (i = 0; i<N; i++)

    {

        for (j = 0; j<N; j++)

        printf("%d ", B[i][j]);

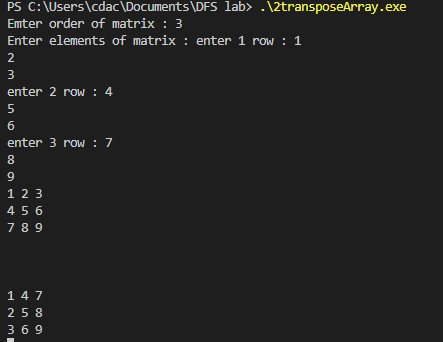
        printf("\n");

    }

    return0;

}

**OUTPUT:**

****

3. Write a program in C to implement Linear Search.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

intlinear\_search(intarr[], intlength, intkey)

{

    inti;

    for(i = 0; i<length; i++){

        if(arr[i] == key ){

            returni;

        }

    }

    return -1;

}

intmain()

{

    intarr[] = {1,5,12,14,76,32,54,2,15};

    intkey;

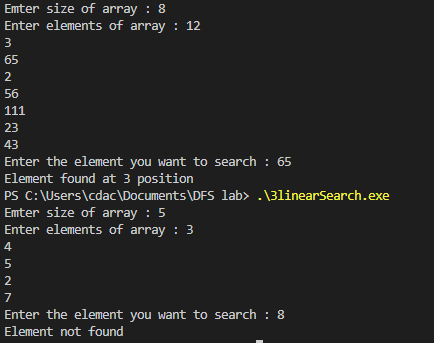
    printf("enter the value you want to search in the array : ");

    scanf("%d", &key);

    printf("%d",linear\_search(arr, sizeof(arr)/4, key));

}

**OUTPUT:**

****

4. write a program in C to implement Bubble Sort.

#include<stdio.h>

#include<stdlib.h>

voidbubble\_sort(intarr[],intlength)

{

    inti;

    intj;

    for(i =0; i<length; i++){

        for(j = 0; j<length-1; j++){

            if(arr[j] >arr[j+1]){

                inttemp = arr[j];

                arr[j] = arr[j+1];

                arr[j+1] = temp;

            }

        }

    }

}

intmain()

{

    inti;

    intarr[] = {2,6,1,5,45,21,32};

    intlength = sizeof(arr)/4;//use sizeof() in main function and not in user defined function.

    bubble\_sort(arr,length);

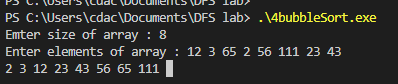
    for(i = 0; i<length; i++){

        printf("%d\t" , arr[i]);

    }

}

**OUTPUT:**



5. Write a program to implement a stack and its operations using an array in C.

// C program for array implementation of stack

#include<limits.h>

#include<stdio.h>

#include<stdlib.h>

// A structure to represent a stack

structStack {

    inttop;

    unsignedcapacity;

    int\* array;

};

// function to create a stack of given capacity. It initializes size of

// stack as 0

structStack\* createStack(unsignedcapacity)

{

    structStack\* stack = (structStack\*)malloc(sizeof(structStack));

    stack->capacity = capacity;

    stack->top = -1;

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

    returnstack;

}

// Stack is full when top is equal to the last index

intisFull(structStack\* stack)

{

    returnstack->top == stack->capacity - 1;

}

// Stack is empty when top is equal to -1

intisEmpty(structStack\* stack)

{

    returnstack->top == -1;

}

// Function to add an item to stack. It increases top by 1

voidpush(structStack\* stack, intitem)

{

    if (isFull(stack))

        return;

    stack->array[++stack->top] = item;

    printf("%d pushed to stack\n", item);

}

// Function to remove an item from stack. It decreases top by 1

intpop(structStack\* stack)

{

    if (isEmpty(stack))

        returnINT\_MIN;

    returnstack->array[stack->top--];

}

// Function to return the top from stack without removing it

intpeek(structStack\* stack)

{

    if (isEmpty(stack))

        returnINT\_MIN;

    returnstack->array[stack->top];

}

// Driver program to test above functions

intmain()

{

    structStack\* stack = createStack(100);

    push(stack, 10);

    push(stack, 20);

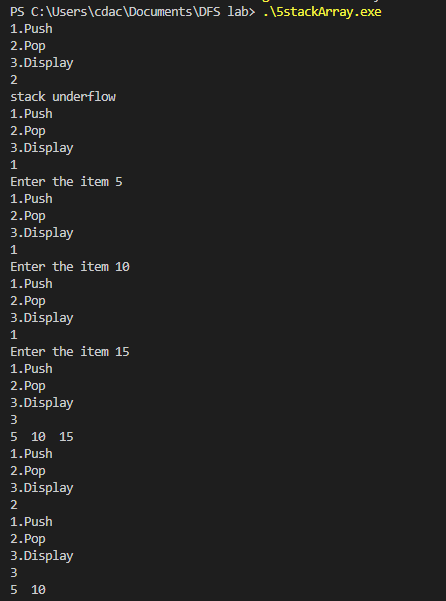
    push(stack, 30);

    printf("%d popped from stack\n", pop(stack));

    return0;

}

**OUTPUT:**



6. Write a program to implement a Queue and its operations using an array.

// C program for array implementation of queue

#include<limits.h>

#include<stdio.h>

#include<stdlib.h>

// A structure to represent a queue

structQueue {

    intfront, rear, size;

    unsignedcapacity;

    int\* array;

};

// function to create a queue

// of given capacity.

// It initializes size of queue as 0

structQueue\* createQueue(unsignedcapacity)

{

    structQueue\* queue = (structQueue\*)malloc(

        sizeof(structQueue));

    queue->capacity = capacity;

    queue->front = queue->size = 0;

    // This is important, see the enqueue

    queue->rear = capacity - 1;

    queue->array = (int\*)malloc(

        queue->capacity \* sizeof(int));

    returnqueue;

}

// Queue is full when size becomes

// equal to the capacity

intisFull(structQueue\* queue)

{

    return (queue->size == queue->capacity);

}

// Queue is empty when size is 0

intisEmpty(structQueue\* queue)

{

    return (queue->size == 0);

}

// Function to add an item to the queue.

// It changes rear and size

voidenqueue(structQueue\* queue, intitem)

{

    if (isFull(queue))

        return;

    queue->rear = (queue->rear + 1)

                % queue->capacity;

    queue->array[queue->rear] = item;

    queue->size = queue->size + 1;

    printf("%d enqueued to queue\n", item);

}

// Function to remove an item from queue.

// It changes front and size

intdequeue(structQueue\* queue)

{

    if (isEmpty(queue))

        returnINT\_MIN;

    intitem = queue->array[queue->front];

    queue->front = (queue->front + 1)

                % queue->capacity;

    queue->size = queue->size - 1;

    returnitem;

}

// Function to get front of queue

intfront(structQueue\* queue)

{

    if (isEmpty(queue))

        returnINT\_MIN;

    returnqueue->array[queue->front];

}

// Function to get rear of queue

intrear(structQueue\* queue)

{

    if (isEmpty(queue))

        returnINT\_MIN;

    returnqueue->array[queue->rear];

}

// Driver program to test above functions./

intmain()

{

    structQueue\* queue = createQueue(1000);

    enqueue(queue, 10);

    enqueue(queue, 20);

    enqueue(queue, 30);

    enqueue(queue, 40);

    printf("%d dequeued from queue\n\n",

        dequeue(queue));

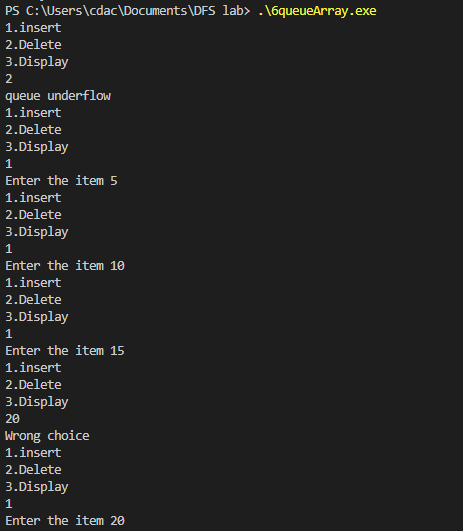
    printf("Front item is %d\n", front(queue));

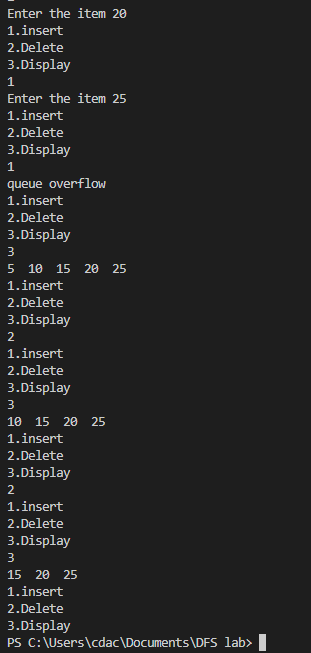
    printf("Rear item is %d\n", rear(queue));

    return0;

}

**OUTPUT:**

****

****

7. Write C program to find factorial of n using :

(i)Recursion

#include<stdio.h>

#include<stdlib.h>

intfactorial(intnum){

    if (num == 0) //base condition

    {

        return1;

    }

    intsmall\_ans = factorial(num-1);

    returnnum \* small\_ans;

    }

intmain()

{

    intnum;

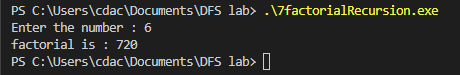
    printf("Enter the number to find the factorial : ");

    scanf("%d", &num);

    printf("%d",factorial(num));

}

**OUTPUT:**

****

(ii) Iteration

#include<stdio.h>

#include<stdlib.h>

intfactorial(intnum){

    intresult = 1;

    for (inti = 1; i<= num; i++){

        result = result \* i;

    }

    returnresult;

}

intmain()

{

    intnum;

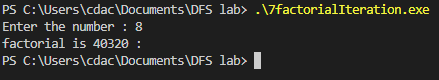
    printf("Enter the number to find the factorial : ");

    scanf("%d", &num);

    printf("%d",factorial(num));

}

**OUTPUT:**



8. Write C programs to compute nth Fibonacci using :

(i) Recursion

#include<stdio.h>

intfactorial(intn)

{

    // base case

    if (n == 0)

    {

        return1;

    }

    else

    {

        returnn \* factorial(n - 1);

    }

}

intfibbonacci(intn)

{

    if (n == 0)

    {

        return0;

    }

    elseif (n == 1)

    {

        return1;

    }

    else

    {

        return (fibbonacci(n - 1) + fibbonacci(n - 2));

    }

}

intmain()

{

    intn = 5;

    inti;

    printf("Factorial of %d: %d\n", n, factorial(n));

    printf("Fibbonacci of %d: ", n);

    for (i = 0; i<n; i++)

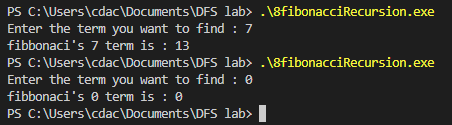
    {

        printf("%d ", fibbonacci(i));

    }

}

**OUTPUT:**



(ii) Iteration

#include<stdio.h>

#include<stdlib.h>

intfibonacci(intnum){

    if (num == 1)

    {

        return0;

    }

    elseif (num == 1 || num ==2 )

    {

        return1;

    }

    intresult = 1;

    for (inti = 1; i<= num; i++){

        result = result \* i;

    }

    returnresult;

}

intmain()

{

    intnum;

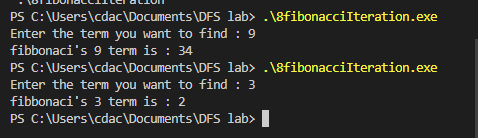
    printf("Enter the number to find the factorial : ");

    scanf("%d", &num);

    printf("%d",factorial(num));

}

**OUTPUT:**

****

9. Write a program in C to implement Binary Search.

#include<stdio.h>

#include<stdlib.h>

intbinary\_search(intarr[],intlength, intkey)

{

    intstart = 0;

    intend = length-1;

    while (start<= end){

        intmid = start+(end-start)/2;

        if (key == arr[mid])

        {

            returnmid;

        }

        elseif (key>= arr[mid])

        {

            start = mid+1;

        }

        else{

            end = mid-1;

        }

    }

    return -1;

}

intmain()

{

    intarr[] = {1,3,5,7,8,9,10,13,15,17};//array must be sorted

    intlength = sizeof(arr)/4;

    intkey;

    printf("enter the value you want to search in the array : ");

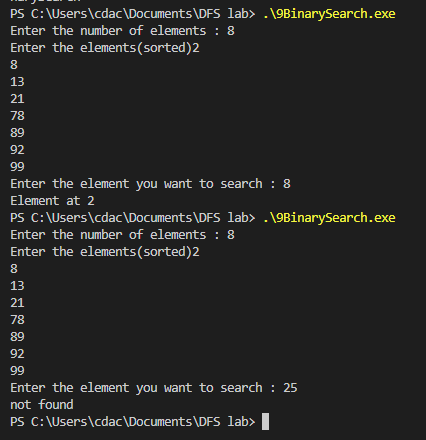
    scanf("%d", &key);

    intelement\_index = binary\_search(arr,length,key);

    printf("%d",element\_index);

}

**OUTPUT:**



10. Write a program in C to implement Insertion sort.

#include<stdio.h>

#include<stdlib.h>

voidinsertion\_sort(intarr[],intlength)

{

    inti;//unsorted

    intj;//sorted

    for(i = 1; i<length; i++){

        j = i-1;

        inttemp = arr[i];//storing unsorted element in temp

        while (j>= 0&&arr[j] >temp )//shifting towards right

        {

            arr[j+1] = arr[j];//shifting sorted

            j--;

        }

        arr[j+1] = temp;//making unsorted element as sorted

    }

}

intmain()

{

    inti;

    intarr[] = {2,6,1,5,45,21,32};

    intlength = sizeof(arr)/4;//use sizeof() in main function and not in user defined function.

    insertion\_sort(arr,length);

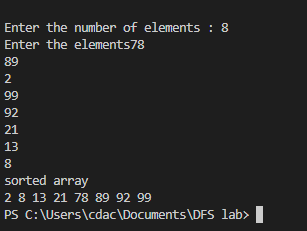
    for(i = 0; i<length; i++){

        printf("%d\t" , arr[i]);

    }

}

**OUTPUT:**

****

11 . Write a program in C to implement Selection Sort.

#include<stdio.h>

#include<stdlib.h>

voidselection\_sort(intarr[], intlength){

    inti;

    intj;

        //there will be n-1 rounds 0 to length-2

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

            intmin = arr[i];

            intcurrentMinIndex = i;

            for (j = i+1; j<length-1; j++) {//for checking the minimum in the complete array

                if (arr[j] <min){

                    min = arr[j];

                    currentMinIndex =j;

                }

            }

            //swapping

            inttemp = arr[i];

            arr[i] = min;

            arr[currentMinIndex] = temp;

        }

    }

intmain(){

        intarr[] = {3,1,4,2,6,9};

        intlength = sizeof(arr)/4;

        selection\_sort(arr,length);

        inti;

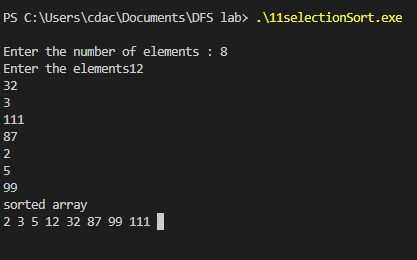
        for(i = 0; i<length; i++){

        printf("%d\t" , arr[i]);

    }

}

**OUTPUT:**

****

12. Write a program to implement linked list and its operations in C.

#include<stdio.h>

#include<stdlib.h>

structnode

{

    intnum;                        //Data of the node

    structnode \*nextptr;           //Address of the next node

}\*stnode;

voidcreateNodeList(intn); // function to create the list

voiddisplayList();         // function to display the list

intmain()

{

    intn;

        printf("\n\n Linked List : To create and display Singly Linked List :\n");

    printf(" Input the number of nodes : ");

    scanf("%d", &n);

    createNodeList(n);

    printf("\n Data entered in the list : \n");

    displayList();

    return0;

}

voidcreateNodeList(intn)

{

    structnode \*fnNode, \*tmp;

    intnum, i;

    stnode = (structnode\*)malloc(sizeof(structnode));

    if(stnode == NULL) //check whether the fnnode is NULL and if so no memory allocation

    {

        printf(" Memory can not be allocated.");

    }

    else

    {

// reads data for the node through keyboard

        printf(" Input data for node 1 : ");

        scanf("%d", &num);

        stnode->num = num;

        stnode->nextptr = NULL; // links the address field to NULL

        tmp = stnode;

// Creating n nodes and adding to linked list

        for(i=2; i<=n; i++)

        {

            fnNode = (structnode\*)malloc(sizeof(structnode));

            if(fnNode == NULL)

            {

                printf(" Memory can not be allocated.");

                break;

            }

            else

            {

                printf(" Input data for node %d : ", i);

                scanf(" %d", &num);

                fnNode->num = num;      // links the num field of fnNode with num

                fnNode->nextptr = NULL; // links the address field of fnNode with NULL

                tmp->nextptr = fnNode; // links previous node i.e.tmp to the fnNode

                tmp = tmp->nextptr;

            }

        }

    }

}

voiddisplayList()

{

    structnode \*tmp;

    if(stnode == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        tmp = stnode;

        while(tmp != NULL)

        {

            printf(" Data = %d\n", tmp->num);       // prints the data of current node

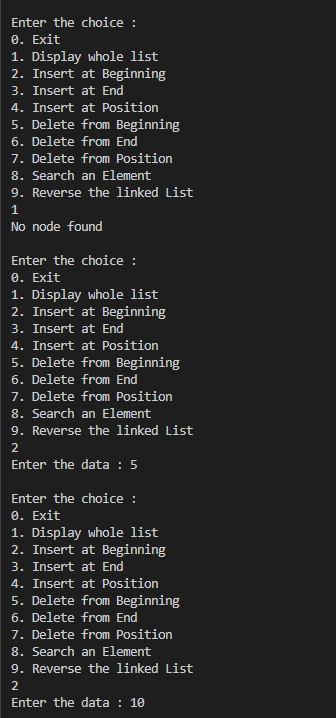
            tmp = tmp->nextptr;                     // advances the position of current node

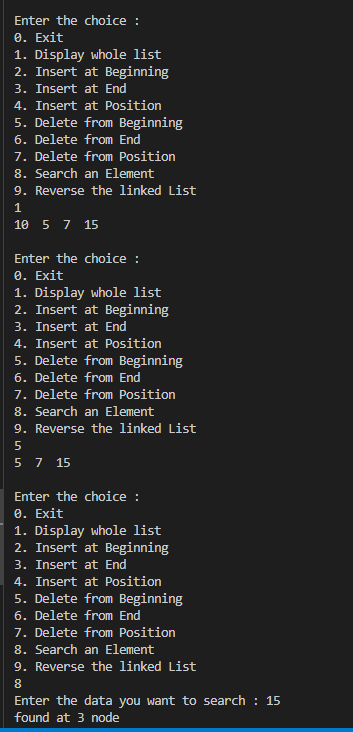
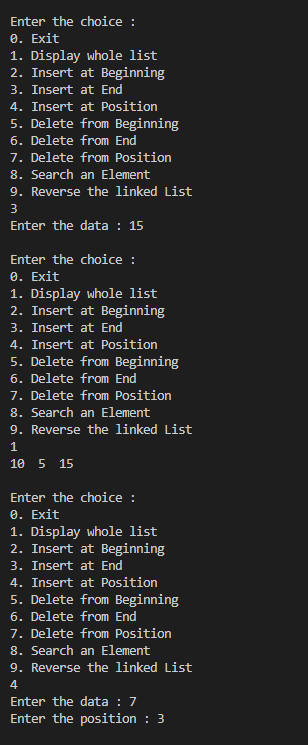
        }

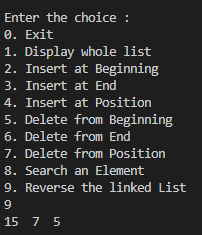
    }

}

**OUTPUT:**

****

****

****

13. Write a program in C to implement merge sort.

#include<stdio.h>

#include<stdlib.h>

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

voidmerge(intarr[], intl, intm, intr)

{

    inti, j, k;

    intn1 = m - l + 1;

    intn2 = r - m;

    //creating temporary arrays

    intL[n1], R[n2];

    //copying to arrays

    for (i = 0; i<n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j<n2; j++)

        R[j] = arr[m + 1 + j];

    //merging the arrays

    i = 0; // Initial index of first subarray

    j = 0; // Initial index of second subarray

    k = l; // Initial index of merged subarray

    while (i<n1&&j<n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        }

        else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    //copying leftout elements if any

    while (i<n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    //copying remaining elements if any

    while (j<n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

voidmergeSort(intarr[], intl, intr)

{

    if (l<r) {

        // Same as (l+r)/2, but avoids overflow for

        // large l and h

        intm = l + (r - l) / 2;

        // Sort first and second halves

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

voidprintArray(intA[], intsize)

{

    inti;

    for (i = 0; i<size; i++)

        printf("%d ", A[i]);

    printf("\n");

}

intmain()

{

    intarr[] = { 14,2,56,13,3,8,9,44 };

    intarr\_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");

    printArray(arr, arr\_size);

    mergeSort(arr, 0, arr\_size - 1);

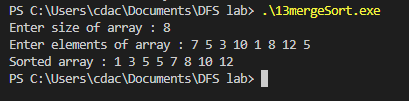
    printf("\nSorted array is \n");

    printArray(arr, arr\_size);

    return0;

}

**OUTPUT:**



14. Write a program in C to implement quick sort.

#include<stdio.h>

// function to swap elements

voidswap(int \*a, int \*b) {

  intt = \*a;

  \*a = \*b;

  \*b = t;

}

// function to find the partition position

intpartition(intarray[], intlow, inthigh) {

  // select the rightmost element as pivot

  intpivot = array[high];

  // pointer for greater element

  inti = (low - 1);

  // traverse each element of the array

  // compare them with the pivot

  for (intj = low; j<high; j++) {

    if (array[j] <= pivot) {

      // if element smaller than pivot is found

      // swap it with the greater element pointed by i

      i++;

      // swap element at i with element at j

      swap(&array[i], &array[j]);

    }

  }

  // swap the pivot element with the greater element at i

  swap(&array[i + 1], &array[high]);

  // return the partition point

  return (i + 1);

}

voidquickSort(intarray[], intlow, inthigh) {

  if (low<high) {

    // find the pivot element such that

    // elements smaller than pivot are on left of pivot

    // elements greater than pivot are on right of pivot

    intpi = partition(array, low, high);

    // recursive call on the left of pivot

    quickSort(array, low, pi - 1);

    // recursive call on the right of pivot

    quickSort(array, pi + 1, high);

  }

}

// function to print array elements

voidprintArray(intarray[], intsize) {

  for (inti = 0; i<size; ++i) {

    printf("%d  ", array[i]);

  }

  printf("\n");

}

// main function

intmain() {

  intdata[] = {1,9,2,8,3,7,4,6,0,5};

  intn = sizeof(data) / sizeof(data[0]);

  printf("Unsorted Array\n");

  printArray(data, n);

  // perform quicksort on data

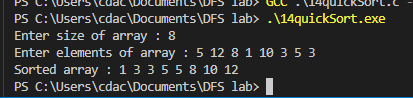
  quickSort(data, 0, n - 1);

  printf("Sorted array in ascending order: \n");

  printArray(data, n);

}

**OUTPUT:**

****

15. write a program in C to implement a queue using 2 stacks.

#include<stdio.h>

#include<stdlib.h>

/\* Functions and variables used \*/

voidpush1(int);

voidpush2(int);

intpop1();

intpop2();

voidenqueue();

voiddequeue();

voiddisplay();

voidcreate();

intstack1[100], stack2[100];

inttop1 = -1, top2 = -1;

intcount = 0;

/\* Main Function \*/

intmain()

{

    intchoice;

    printf("\nQUEUE USING STACKS IMPLEMENTATION\n\n");

    printf("\n1.ENQUEUE");

    printf("\n2.DEQUEUE");

    printf("\n3.DISPLAY");

    printf("\n4.EXIT");

    printf("\n");

    create();

    while (1)

    {

        printf("\nEnter your choice : ");

        scanf("%d", &choice);

        switch (choice)

        {

        case1:

            enqueue();

            break;

        case2:

            dequeue();

            break;

        case3:

            display();

            break;

        case4:

            exit(0);

        default:

            printf("\nInvalid Choice\n");

        }

    }

}

/\* Function to initialize top of two stacks\*/

voidcreate()

{

    top1 = top2 = -1;

}

/\* Function to push an element to stack \*/

voidpush1(intelement)

{

    stack1[++top1] = element; // Pushing the element to stack1

}

/\* Function to pop element from stack \*/

intpop1()

{

    return (stack1[top1--]); // Pop element from stack1

}

/\* Function to push an element on to stack \*/

voidpush2(intelement)

{

    stack2[++top2] = element; // Pushing the element to stack2

}

/\* Function to pop an element from stack \*/

intpop2()

{

    return (stack2[top2--]); // pop element from stack2

}

/\* Function to enqueue an element into the queue using stack \*/

voidenqueue()

{

    intdata, i;

    printf("Enter the data : ");

    scanf("%d", &data);

    push1(data); // Push data from stack to the queue

    count++;

}

/\* Function to dequeue an element from the queue using stack \*/

voiddequeue()

{

    inti;

    for (i = 0; i<= count; i++)

    {

        push2(pop1()); // Pop elements from stack1 and push them to stack2

    }

    pop2(); // Pop the element from stack2 which is the element to be dequeued

    count--;

    for (i = 0; i<= count; i++)

    {

        push1(pop2()); // Push back all the elements from stack2 to stack1

    }

}

/\*Function to display the elements in the queue\*/

voiddisplay()

{

    inti;

    if (top1 == -1)

    {

        printf("\nEMPTY QUEUE\n");

    }

    else

    {

        printf("\nQUEUE ELEMENTS : ");

        for (i = 0; i<= top1; i++)

        {

            printf(" %d ", stack1[i]);

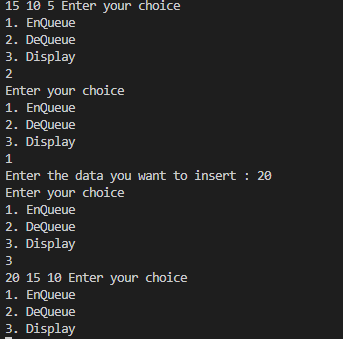
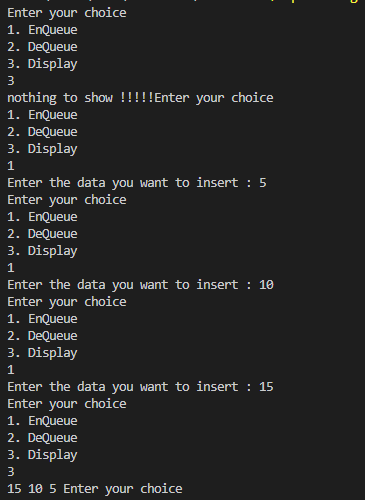
        }

        printf("\n");

    }

}

**OUTPUT:**



16. Write a program in C to show Tree traversal.

// C program for different tree traversals

#include<stdio.h>

#include<stdlib.h>

/\* A binary tree node has data, pointer to left child

and a pointer to right child \*/

structnode

{

    intdata;

    structnode \*left;

    structnode \*right;

};

/\* Helper function that allocates a new node with the

given data and NULL left and right pointers. \*/

structnode \*newNode(intdata)

{

    structnode \*node = (structnode\*)malloc(sizeof(structnode));

    node->data = data;

    node->left = NULL;

    node->right = NULL;

    return (node);

}

/\* Given a binary tree, print its nodes according to the

"bottom-up" postorder traversal. \*/

voidprintPostorder(structnode \*node)

{

    if (node == NULL)

        return;

    // first recur on left subtree

    printPostorder(node->left);

    // then recur on right subtree

    printPostorder(node->right);

    // now deal with the node

    printf("%d ", node->data);

}

/\* Given a binary tree, print its nodes in inorder\*/

voidprintInorder(structnode \*node)

{

    if (node == NULL)

        return;

    /\* first recur on left child \*/

    printInorder(node->left);

    /\* then print the data of node \*/

    printf("%d ", node->data);

    /\* now recur on right child \*/

    printInorder(node->right);

}

/\* Given a binary tree, print its nodes in preorder\*/

voidprintPreorder(structnode \*node)

{

    if (node == NULL)

        return;

    /\* first print data of node \*/

    printf("%d ", node->data);

    /\* then recur on left subtree \*/

    printPreorder(node->left);

    /\* now recur on right subtree \*/

    printPreorder(node->right);

}

/\* Driver program to test above functions\*/

intmain()

{

    structnode \*root = newNode(1);

    root->left = newNode(2);

    root->right = newNode(3);

    root->left->left = newNode(4);

    root->left->right = newNode(5);

    printf("\nPreorder traversal of binary tree is \n");

    printPreorder(root);

    printf("\nInorder traversal of binary tree is \n");

    printInorder(root);

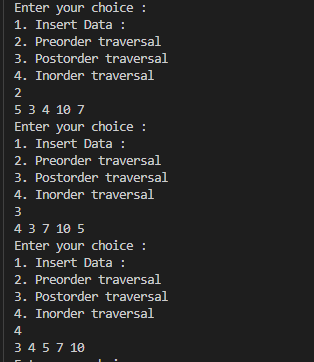
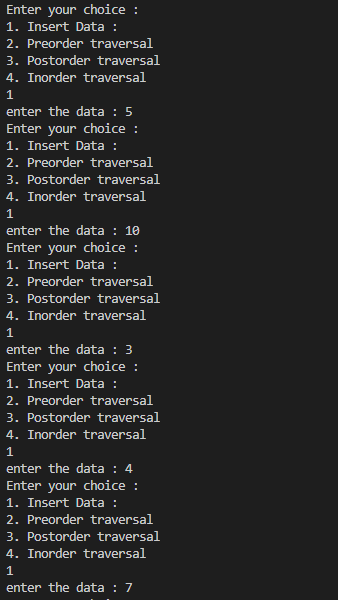
    printf("\nPostorder traversal of binary tree is \n");

    printPostorder(root);

    getchar();

    return0;}

**OUTPUT:**

****

**Q.BFS TRAVERSAL**

**#define MAXSIZE 10**

**#define INF 1000**

**#include<stdio.h>**

**#include<stdlib.h>**

**int main()**

**{**

**int i,j;//for looping**

**int n;//number of nodes.**

**int visitedNodes[MAXSIZE];//for marking which nodes are already part of the MST**

**int adj[MAXSIZE][MAXSIZE];**

**int frontier[MAXSIZE];**

**int frontierSize=0;**

**int visitOrder[MAXSIZE];**

**int visitedCount=0;**

**int level[MAXSIZE];**

**int currentLevel=0;**

**printf("Enter the number of nodes: ");**

**scanf("%d",&n);**

**//initialize visitedNodes**

**i=0;**

**while(i<n){**

**visitedNodes[i++]=0;**

**}**

**//initialize level**

**i=0;**

**while(i<n){**

**level[i++]=0;**

**}**

**//initialize frontier**

**i=0;**

**while(i<n){**

**frontier[i++]=0;**

**}**

**//initialize adjacency matrix adj**

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

**for(j=0;j<n;j++){**

**adj[i][j]=0;**

**}**

**}**

**//taking input for edges**

**printf("Enter node1 and node2(-1 -1 to stop): \n");**

**int inLoop=1;**

**while(inLoop==1){**

**int v1,v2;**

**scanf("%d %d",&v1,&v2);**

**if(v1>=0)**

**{**

**//register this edge in cost matrix**

**adj[v1][v2]=1;**

**adj[v2][v1]=1;**

**}**

**else{**

**inLoop=0;**

**}**

**}**

**//BFS**

**frontier[0]=1;**

**frontierSize=1;**

**visitedNodes[0]=1;**

**visitOrder[visitedCount++]=0;**

**level[0]=0;**

**while(frontierSize>0)**

**{**

**int nextFrontier[n];**

**currentLevel++;**

**frontierSize=0;**

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

**if(frontier[i]==0) continue;**

**for(j=0;j<n;j++){**

**if(visitedNodes[j]==1) continue;**

**if(adj[i][j]==1){**

**visitOrder[visitedCount++]=j;**

**level[j]=currentLevel;**

**nextFrontier[frontierSize++]=j;**

**visitedNodes[j]=1;**

**}**

**}**

**}**

**//copy nextFrontier to frontier**

**i=0;**

**while(i<n){**

**frontier[i++]=0;**

**}**

**for(i=0;i<frontierSize;i++){**

**frontier[nextFrontier[i]]=1;**

**}**

**}**

**//print order of traversal**

**printf("node \t level\n");**

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

**printf("%d \t %d\n",visitOrder[i],level[i]);**

**}**

**}**

**Q.DFS TRAVERSAL.**

**#define MAX 20//maximum number of nodes**

**#include<stdio.h>**

**void DFS(int,int);**

**int visited[MAX];//to mark which nodes have been visited**

**int adj[MAX][MAX];**

**int main(){**

**int i,j;**

**int n;//number of nodes**

**printf("Enter number of nodes: ");**

**scanf("%d",&n);**

**//initially mark all nodes as not visited.**

**i=0;**

**while(i<n){**

**visited[i++]=0;**

**}**

**i=0;**

**//initialize the adjacency matrix with zero in every cell**

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

**for(j=0;j<n;j++){**

**adj[i++][j++]=0;**

**}**

**}**

**//taking input for edges**

**printf("Enter node1 and node2(-1 -1 to stop): \n");**

**int inLoop=1;**

**while(inLoop==1){**

**int v1,v2;**

**scanf("%d %d",&v1,&v2);**

**if(v1>=0)**

**{**

**//register this edge in cost matrix**

**adj[v1][v2]=1;**

**adj[v2][v1]=1;**

**}**

**else{**

**inLoop=0;**

**}**

**}**

**//DFS logic**

**//the following loop selects every one by one and calls DFS for this node.**

**//advantage of this mechanism is that the entire graph can be explored even if the graph is not connected.**

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

**//pick i only if it is not visited.**

**if(visited[i]==0){**

**DFS(i,n);**

**}**

**}**

**}**

**void DFS(int v, int n){**

**int k;**

**visited[v]=1;**

**printf("%d--->",v);**

**for(k=0;k<n;k++){**

**if(adj[v][k]==1 && visited[k]==0){**

**DFS(k,n);**

**}**

**}**

**printf("backtraced---->");**

**}**

**Q.KRUSKAL**

**#define MAXSIZE 10**

**#define INF 1000**

**#include<stdio.h>**

**#include<stdlib.h>**

**int main()**

**{**

**int i,j;//for looping**

**int n;//number of nodes.**

**int visitedNodes[MAXSIZE];//for marking which nodes are already part of the MST**

**int adj[MAXSIZE][MAXSIZE];**

**int MST[MAXSIZE-1][2];**

**int MSTedgeCount=0;**

**int MSTweight=0;**

**printf("Enter the number of nodes: ");**

**scanf("%d",&n);**

**//initialize visitedNodes**

**i=0;**

**while(i<n){**

**visitedNodes[i++]=0;**

**}**

**//initialize cost matrix adj**

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

**for(j=0;j<n;j++){**

**adj[i][j]=INF;**

**}**

**}**

**//taking input for edges**

**printf("Enter node1, node2 and edge-weight(-1 -1 -1 to stop): \n");**

**int inLoop=1;**

**while(inLoop==1){**

**int v1,v2,w;**

**scanf("%d %d %d",&v1,&v2,&w);**

**if(w>=0)**

**{**

**//register this edge in cost matrix**

**adj[v1][v2]=w;**

**adj[v2][v1]=w;**

**}**

**else{**

**inLoop=0;**

**}**

**}**

**//Kruskal's algorithm starts here.**

**while(MSTedgeCount<n-1)**

**{**

**int smallestWeight=INF;**

**int node1,node2;**

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

**for(j=0;j<n;j++)**

**{**

**if(visitedNodes[i]==1 && visitedNodes[j]==1) continue;**

**if(adj[i][j]<smallestWeight){**

**smallestWeight=adj[i][j];**

**node1=i;**

**node2=j;**

**}**

**}**

**}**

**//we have the next smallest edge. Add it to MST**

**MST[MSTedgeCount][0]=node1;**

**MST[MSTedgeCount][1]=node2;**

**MSTedgeCount++;**

**MSTweight+=smallestWeight;**

**//updated visitedNodes**

**visitedNodes[node1]=1;**

**visitedNodes[node2]=1;**

**}**

**printf("Edges in the MST are as follows.\n");**

**for(i=0;i<MSTedgeCount;i++){**

**printf("%d----%d\n",MST[i][0],MST[i][1]);**

**}**

**printf("The weight of MST is: %d\n",MSTweight);**

**}**

**Q.MST\_PRIMS**

**#define MAXSIZE 10**

**#define INF 1000**

**#include<stdio.h>**

**#include<stdlib.h>**

**int main()**

**{**

**int i,j;//for looping**

**int n;//number of nodes.**

**int visitedNodes[MAXSIZE];//for marking which nodes are already part of the MST**

**int adj[MAXSIZE][MAXSIZE];**

**int MST[MAXSIZE-1][2];**

**int MSTedgeCount=0;**

**int MSTweight=0;**

**printf("Enter the number of nodes: ");**

**scanf("%d",&n);**

**//initialize visitedNodes**

**i=0;**

**while(i<n){**

**visitedNodes[i++]=0;**

**}**

**//initialize cost matrix adj**

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

**for(j=0;j<n;j++){**

**adj[i][j]=INF;**

**}**

**}**

**//taking input for edges**

**printf("Enter node1, node2 and edge-weight(-1 -1 -1 to stop): \n");**

**int inLoop=1;**

**while(inLoop==1){**

**int v1,v2,w;**

**scanf("%d %d %d",&v1,&v2,&w);**

**if(w>=0)**

**{**

**//register this edge in cost matrix**

**adj[v1][v2]=w;**

**adj[v2][v1]=w;**

**}**

**else{**

**inLoop=0;**

**}**

**}**

**//Prim's algorithm starts here.**

**visitedNodes[0]=1;**

**while(MSTedgeCount<n-1){**

**int smallestEdgeWeight = INF;**

**int node1,node2;**

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

**if(visitedNodes[i]==0) continue;**

**for(j=0;j<n;j++)**

**{**

**//check if j is already visited or not**

**if(visitedNodes[j]==1)continue;**

**if(adj[i][j]<smallestEdgeWeight){**

**smallestEdgeWeight=adj[i][j];**

**node1=i;**

**node2=j;**

**}**

**}**

**}**

**//now you have the node which is at the minimum distance from already discovered tree.**

**//add this node to tree**

**MST[MSTedgeCount][0]=node1;**

**MST[MSTedgeCount][1]=node2;**

**MSTedgeCount++;**

**MSTweight+=smallestEdgeWeight;**

**visitedNodes[node2]=1;**

**}**

**printf("Edges in the MST are as follows.\n");**

**for(i=0;i<MSTedgeCount;i++){**

**printf("%d----%d\n",MST[i][0],MST[i][1]);**

**}**

**printf("The weight of MST is: %d\n",MSTweight);**

**}**

**Q.FLOYD WARSALL**

**#define MAXSIZE 10**

**#define INF 1000**

**#include<stdio.h>**

**#include<stdlib.h>**

**int main()**

**{**

**int i,j,k;//for looping**

**int n;//number of nodes.**

**int adj[MAXSIZE][MAXSIZE];**

**printf("Enter the number of nodes: ");**

**scanf("%d",&n);**

**//initialize cost matrix adj**

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

**for(j=0;j<n;j++){**

**if(i==j){**

**adj[i][j]=0;**

**}**

**else{**

**adj[i][j]=INF;**

**}**

**}**

**}**

**//taking input for edges**

**printf("Enter node1, node2 and edge-weight(-1 -1 -1 to stop): \n");**

**int inLoop=1;**

**while(inLoop==1){**

**int v1,v2,w;**

**scanf("%d %d %d",&v1,&v2,&w);**

**if(v1>-1 && v2>-1)**

**{**

**//register this edge in cost matrix**

**adj[v1][v2]=w;**

**}**

**else{**

**inLoop=0;**

**}**

**}**

**printf("adj matrix is as follows\n");**

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

**for(j=0;j<n;j++){**

**printf("%d\t",adj[i][j]);**

**}**

**printf("\n");**

**}**

**printf("\n");**

**//applying floyd warshall algorithm**

**for(k=0;k<n;k++){**

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

**for(j=0;j<n;j++){**

**if(i==j) continue;**

**if(adj[i][j]>adj[i][k]+adj[k][j]){**

**adj[i][j]=adj[i][k]+adj[k][j];**

**}**

**}**

**}**

**}**

**printf("All pairs shortest path matrix is as follows.\n");**

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

**for(j=0;j<n;j++){**

**printf("%d\t",adj[i][j]);**

**}**

**printf("\n");**

**}**

**}**