**Data File Structures**

**Lab File**



Submitted By – Ajay Ravesh

Roll No - 53

**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

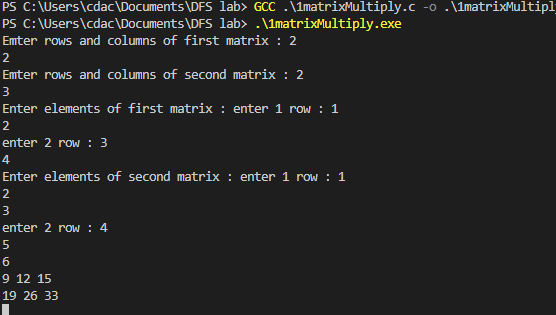
(ii) Preorder Traversal

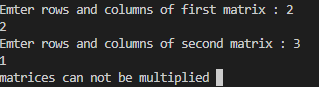
(iii) Inorder Traversal

(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. void mulMat(int mat1[][C1], int mat2[][C2]) {
9. int rslt[R1][C2];
10. printf("Multiplication of given two matrices is:\n\n");
11. for (int i = 0; i < R1; i++) {
12. for (int j = 0; j < C2; j++) {
13. rslt[i][j] = 0;
14. for (int k = 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. int main(void) {
23. int mat1[R1][C1] = {
24. {1, 1, 1, 1},
25. {2, 2, 2, 2},
26. {3, 3, 3, 3},
27. {4, 4, 4, 4}
28. };
29. int mat2[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. return 0;
44. }

**OUTPUT :**

****

****

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

#include <stdio.h>

#define N 4

void transpose(int A[][N], int B[][N])

{

    int i, j;

    for (i = 0; i < N; i++)

        for (j = 0; j < N; j++)

            B[i][j] = A[j][i];

}

int main()

{

    int A[N][N] = { {1, 1, 1, 1},

                    {2, 2, 2, 2},

                    {3, 3, 3, 3},

                    {4, 4, 4, 4}};

    int B[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]);

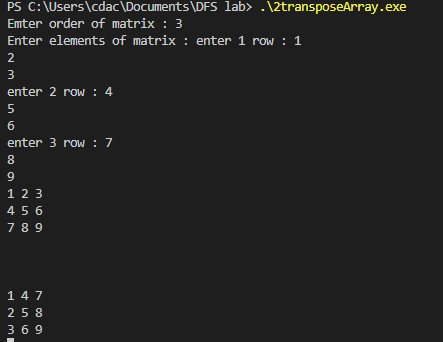
        printf("\n");

    }

    return 0;

}

**OUTPUT:**

****

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int linear\_search(int arr[], int length, int key)

{

    int i;

    for(i = 0; i<length; i++){

        if(arr[i] == key ){

            return i;

        }

    }

    return -1;

}

int main()

{

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

    int key;

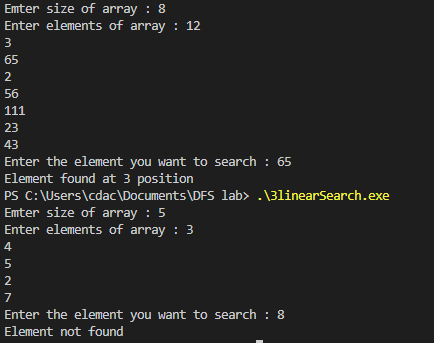
    printf("enter the value you want to search in the array : ");

    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>

void bubble\_sort(int arr[],int length)

{

    int i;

    int j;

    for(i =0; i < length; i++){

        for(j = 0; j<length-1; j++){

            if(arr[j] > arr[j+1]){

                int temp = arr[j];

                arr[j] = arr[j+1];

                arr[j+1] = temp;

            }

        }

    }

}

int main()

{

    int i;

    int arr[] = {2,6,1,5,45,21,32};

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

    bubble\_sort(arr,length);

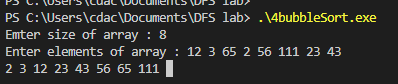
    for(i = 0; i < length; i++){

        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

struct Stack {

    int top;

    unsigned capacity;

    int\* array;

};

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

// stack as 0

struct Stack\* createStack(unsigned capacity)

{

    struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

    stack->capacity = capacity;

    stack->top = -1;

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

    return stack;

}

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

int isFull(struct Stack\* stack)

{

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

}

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

int isEmpty(struct Stack\* stack)

{

    return stack->top == -1;

}

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

void push(struct Stack\* stack, int item)

{

    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

int pop(struct Stack\* stack)

{

    if (isEmpty(stack))

        return INT\_MIN;

    return stack->array[stack->top--];

}

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

int peek(struct Stack\* stack)

{

    if (isEmpty(stack))

        return INT\_MIN;

    return stack->array[stack->top];

}

// Driver program to test above functions

int main()

{

    struct Stack\* stack = createStack(100);

    push(stack, 10);

    push(stack, 20);

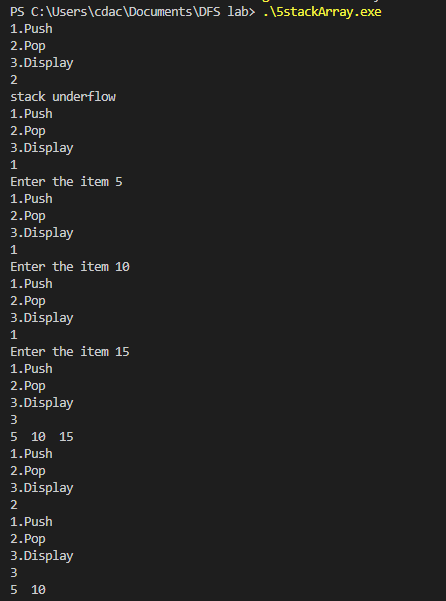
    push(stack, 30);

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

    return 0;

}

**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

struct Queue {

    int front, rear, size;

    unsigned capacity;

    int\* array;

};

// function to create a queue

// of given capacity.

// It initializes size of queue as 0

struct Queue\* createQueue(unsigned capacity)

{

    struct Queue\* queue = (struct Queue\*)malloc(

        sizeof(struct Queue));

    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));

    return queue;

}

// Queue is full when size becomes

// equal to the capacity

int isFull(struct Queue\* queue)

{

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

}

// Queue is empty when size is 0

int isEmpty(struct Queue\* queue)

{

    return (queue->size == 0);

}

// Function to add an item to the queue.

// It changes rear and size

void enqueue(struct Queue\* queue, int item)

{

    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

int dequeue(struct Queue\* queue)

{

    if (isEmpty(queue))

        return INT\_MIN;

    int item = queue->array[queue->front];

    queue->front = (queue->front + 1)

                % queue->capacity;

    queue->size = queue->size - 1;

    return item;

}

// Function to get front of queue

int front(struct Queue\* queue)

{

    if (isEmpty(queue))

        return INT\_MIN;

    return queue->array[queue->front];

}

// Function to get rear of queue

int rear(struct Queue\* queue)

{

    if (isEmpty(queue))

        return INT\_MIN;

    return queue->array[queue->rear];

}

// Driver program to test above functions./

int main()

{

    struct Queue\* queue = createQueue(1000);

    enqueue(queue, 10);

    enqueue(queue, 20);

    enqueue(queue, 30);

    enqueue(queue, 40);

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

        dequeue(queue));

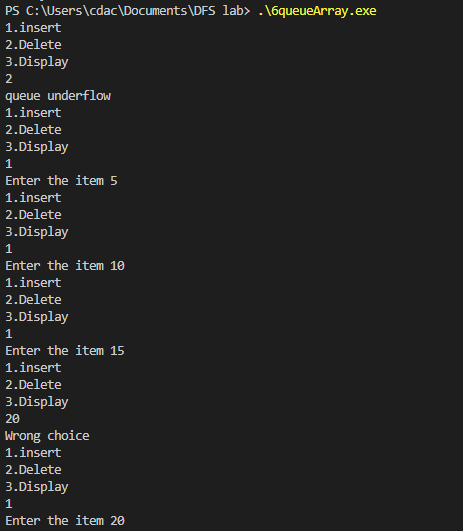
    printf("Front item is %d\n", front(queue));

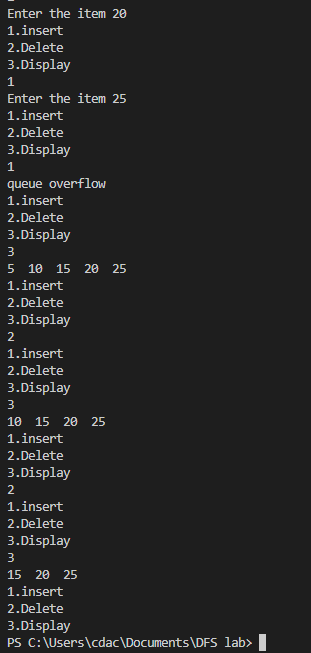
    printf("Rear item is %d\n", rear(queue));

    return 0;

}

**OUTPUT:**

****

****

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

(i)Recursion

#include <stdio.h>

#include <stdlib.h>

int factorial(int num){

    if (num == 0) //base condition

    {

        return 1;

    }

    int small\_ans = factorial(num-1);

    return num \* small\_ans;

    }

int main()

{

    int num;

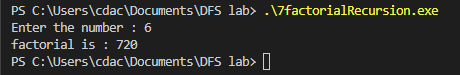
    printf("Enter the number to find the factorial : ");

    scanf("%d", &num);

    printf("%d",factorial(num));

}

**OUTPUT:**

****

(ii) Iteration

#include <stdio.h>

#include <stdlib.h>

int factorial(int num){

    int result = 1;

    for (int i = 1; i <= num; i++){

        result = result \* i;

    }

    return result;

}

int main()

{

    int num;

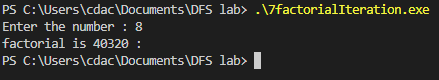
    printf("Enter the number to find the factorial : ");

    scanf("%d", &num);

    printf("%d",factorial(num));

}

**OUTPUT:**



8. Write C programs to compute nth Fibonacci using :

(i) Recursion

#include <stdio.h>

int factorial(int n)

{

    // base case

    if (n == 0)

    {

        return 1;

    }

    else

    {

        return n \* factorial(n - 1);

    }

}

int fibbonacci(int n)

{

    if (n == 0)

    {

        return 0;

    }

    else if (n == 1)

    {

        return 1;

    }

    else

    {

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

    }

}

int main()

{

    int n = 5;

    int i;

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

    printf("Fibbonacci of %d: ", n);

    for (i = 0; i < n; i++)

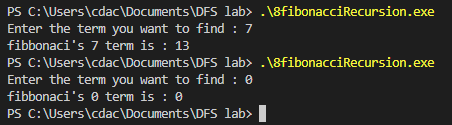
    {

        printf("%d ", fibbonacci(i));

    }

}

**OUTPUT:**



(ii) Iteration

#include <stdio.h>

#include <stdlib.h>

int fibonacci(int num){

    if (num == 1)

    {

        return 0;

    }

    else if (num == 1 || num ==2 )

    {

        return 1;

    }

    int result = 1;

    for (int i = 1; i <= num; i++){

        result = result \* i;

    }

    return result;

}

int main()

{

    int num;

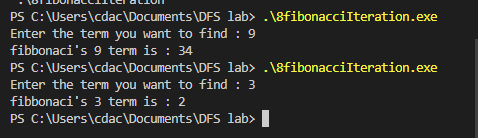
    printf("Enter the number to find the factorial : ");

    scanf("%d", &num);

    printf("%d",factorial(num));

}

**OUTPUT:**

****

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

#include <stdio.h>

#include <stdlib.h>

int binary\_search(int arr[],int length, int key)

{

    int start = 0;

    int end = length-1;

    while (start <= end){

        int mid = start+(end-start)/2;

        if (key == arr[mid])

        {

            return mid;

        }

        else if (key >= arr[mid])

        {

            start = mid+1;

        }

        else{

            end = mid-1;

        }

    }

    return -1;

}

int main()

{

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

    int length = sizeof(arr)/4;

    int key;

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

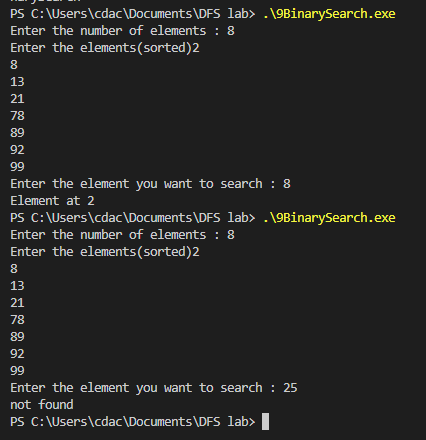
    scanf("%d", &key);

    int element\_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>

void insertion\_sort(int arr[],int length)

{

    int i;//unsorted

    int j;//sorted

    for(i = 1; i < length; i++){

        j = i-1;

        int temp = 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

    }

}

int main()

{

    int i;

    int arr[] = {2,6,1,5,45,21,32};

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

    insertion\_sort(arr,length);

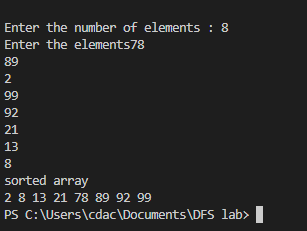
    for(i = 0; i < length; i++){

        printf("%d \t" , arr[i]);

    }

}

**OUTPUT:**

****

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

#include <stdio.h>

#include <stdlib.h>

void selection\_sort(int arr[], int length){

    int i;

    int j;

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

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

            int min = arr[i];

            int currentMinIndex = 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

            int temp = arr[i];

            arr[i] = min;

            arr[currentMinIndex] = temp;

        }

    }

int main(){

        int arr[] = {3,1,4,2,6,9};

        int length = sizeof(arr)/4;

        selection\_sort(arr,length);

        int i;

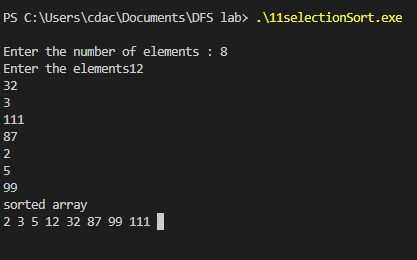
        for(i = 0; i < length; i++){

        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>

struct node

{

    int num;                        //Data of the node

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

}\*stnode;

void createNodeList(int n); // function to create the list

void displayList();         // function to display the list

int main()

{

    int n;

        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();

    return 0;

}

void createNodeList(int n)

{

    struct node \*fnNode, \*tmp;

    int num, i;

    stnode = (struct node \*)malloc(sizeof(struct node));

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

            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;

            }

        }

    }

}

void displayList()

{

    struct node \*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

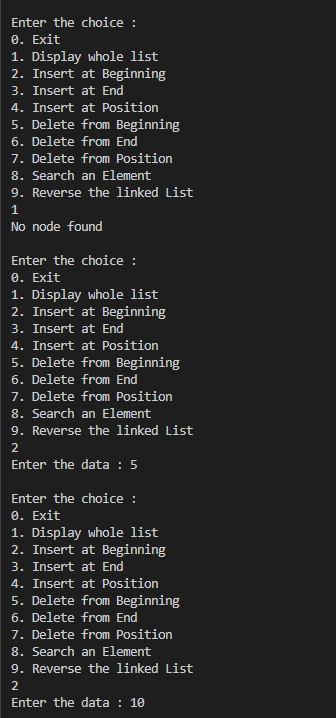
            tmp = tmp->nextptr;                     // advances the position of current node

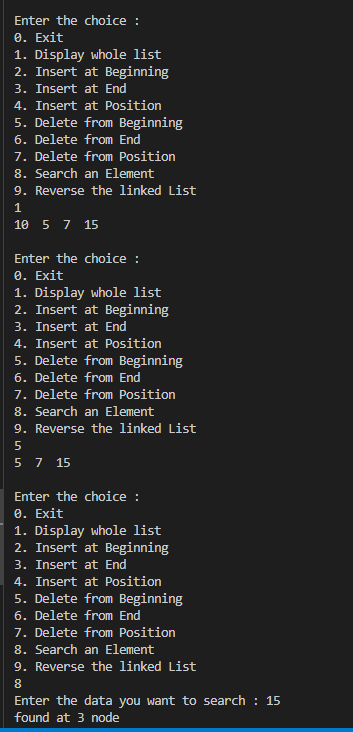
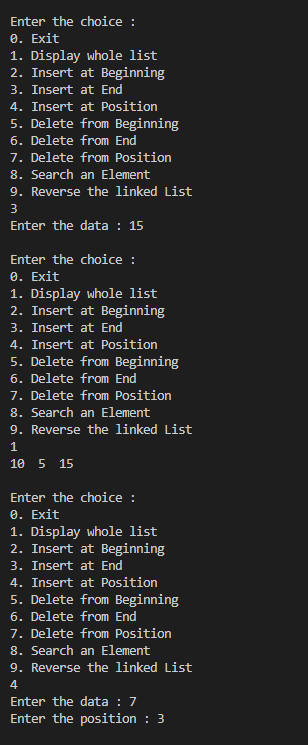
        }

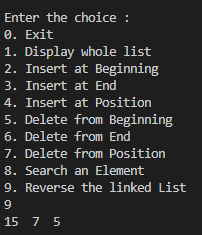
    }

}

**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]

void merge(int arr[], int l, int m, int r)

{

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

    //creating temporary arrays

    int L[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++;

    }

}

void mergeSort(int arr[], int l, int r)

{

    if (l < r) {

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

        // large l and h

        int m = l + (r - l) / 2;

        // Sort first and second halves

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

void printArray(int A[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        printf("%d ", A[i]);

    printf("\n");

}

int main()

{

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

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

    printf("Given array is \n");

    printArray(arr, arr\_size);

    mergeSort(arr, 0, arr\_size - 1);

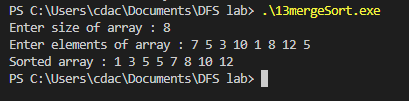
    printf("\nSorted array is \n");

    printArray(arr, arr\_size);

    return 0;

}

**OUTPUT:**



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

#include <stdio.h>

// function to swap elements

void swap(int \*a, int \*b) {

  int t = \*a;

  \*a = \*b;

  \*b = t;

}

// function to find the partition position

int partition(int array[], int low, int high) {

  // select the rightmost element as pivot

  int pivot = array[high];

  // pointer for greater element

  int i = (low - 1);

  // traverse each element of the array

  // compare them with the pivot

  for (int j = 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);

}

void quickSort(int array[], int low, int high) {

  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

    int pi = 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

void printArray(int array[], int size) {

  for (int i = 0; i < size; ++i) {

    printf("%d  ", array[i]);

  }

  printf("\n");

}

// main function

int main() {

  int data[] = {1,9,2,8,3,7,4,6,0,5};

  int n = sizeof(data) / sizeof(data[0]);

  printf("Unsorted Array\n");

  printArray(data, n);

  // perform quicksort on data

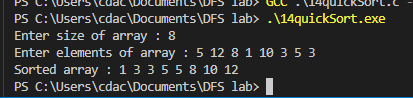
  quickSort(data, 0, n - 1);

  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 \*/

void push1(int);

void push2(int);

int pop1();

int pop2();

void enqueue();

void dequeue();

void display();

void create();

int stack1[100], stack2[100];

int top1 = -1, top2 = -1;

int count = 0;

/\* Main Function \*/

int main()

{

    int choice;

    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)

        {

        case 1:

            enqueue();

            break;

        case 2:

            dequeue();

            break;

        case 3:

            display();

            break;

        case 4:

            exit(0);

        default:

            printf("\nInvalid Choice\n");

        }

    }

}

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

void create()

{

    top1 = top2 = -1;

}

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

void push1(int element)

{

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

}

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

int pop1()

{

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

}

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

void push2(int element)

{

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

}

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

int pop2()

{

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

}

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

void enqueue()

{

    int data, 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 \*/

void dequeue()

{

    int i;

    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\*/

void display()

{

    int i;

    if (top1 == -1)

    {

        printf("\nEMPTY QUEUE\n");

    }

    else

    {

        printf("\nQUEUE ELEMENTS : ");

        for (i = 0; i <= top1; i++)

        {

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

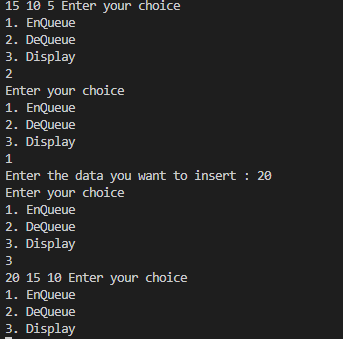
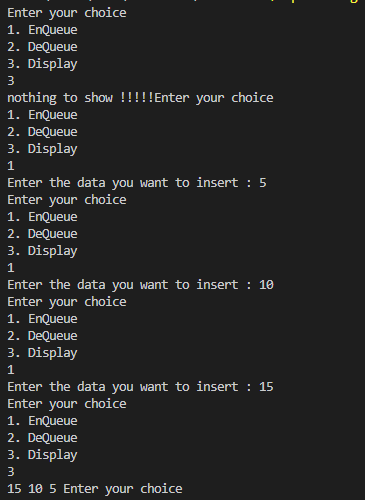
        }

        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 \*/

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

};

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

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

struct node \*newNode(int data)

{

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

    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. \*/

void printPostorder(struct node \*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\*/

void printInorder(struct node \*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\*/

void printPreorder(struct node \*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\*/

int main()

{

    struct node \*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);

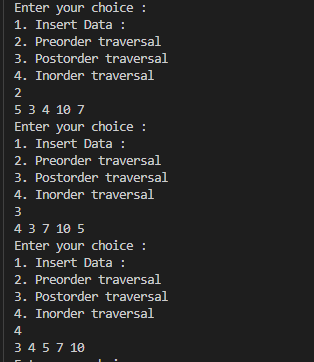
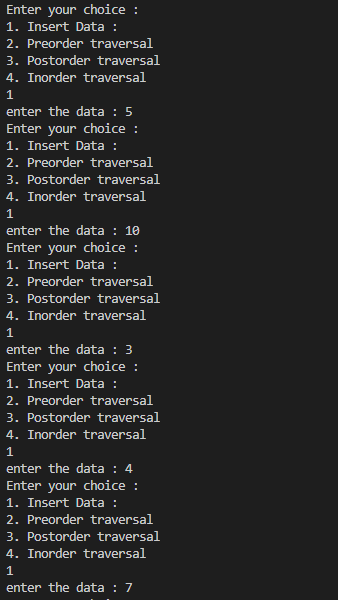
    printf("\nPostorder traversal of binary tree is \n");

    printPostorder(root);

    getchar();

    return 0;}

**OUTPUT:**

****