**LAB RECORD ON DATA STRUCTURE**

***Submitted in partial fulfilment of the***

***award of the degree of***

***Master of Computer Application (MCA)***

****

**DEPARTMENT OF COMPUTER APPLICATIONS**

**P A AZIZ COLLEGE OF ENGINEERING & TECHNOLOGY**

**SEMESTER – 1**

**MCA BATCH – 2021-‘23**

**FACE SHEET**

**NAME:**

**BATCH:**

**SUBJECT:**

**NAME OF THE FACULTY SUPERVISOR:**

**P.A. AZIZ COLLEGE OF ENGINEERING AND TECHNOLOGY**

**GREEN HILLS, KARAKULAM PO, TRIVANDRUM, KERALA 695564**



**CERTIFICATE**

***This is to certify that Mr. / Ms.***  ***Of class Enrolment No has Satisfactorily completed the course in as***

***by the Kalam Technological University for Year (B.E.) semester of***

***Master of Computer Applications in the Academic year \_\_\_\_\_****- .*

***Date of Submission:-***

Faculty Name and Signature **Head of Department**

(Asst. Prof…………..) **(MCA)**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL NO.** | **PARTICULARS** | **PAGE NO.** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**PROGRAM – 1**

**AIM:** C program to implement Breadth First Search.

#include<stdio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {

for(i = 1; i <= n; i++)

if(a[v][i] && !visited[i])

q[++r] = i;

if(f <= r) {

visited[q[f]] = 1;

bfs(q[f++]);

}

}

void main() {

int v;

printf("\n Enter the number of vertices:");

scanf("%d", &n);

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

q[i] = 0;

visited[i] = 0;

}

printf("\n Enter graph data in matrix form:\n");

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

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

scanf("%d", &a[i][j]);

}

}

printf("\n Enter the starting vertex:");

scanf("%d", &v);

bfs(v);

printf("\n The node which are reachable are:\n");

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

if(visited[i])

printf("%d\t", i);

else {

printf("\n Bfs is not possible. Not all nodes are reachable");

break;

}

}

}

**PROGRAM – 2**

**AIM:** C program to implement Disjoint Set

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node{

struct node \*rep;

struct node \*next;

int data;

};

\*heads[50],\*tails[50];

static int countRoot=0;

void makeSet(int x){

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

new->rep=new;

new->next=NULL;

new->data=x;

heads[countRoot]=new;

tails[countRoot++]=new;

}

struct node\* find(int a){

int i; struct node \*tmp=(struct node \*)malloc(sizeof(struct node));

for(i=0;i<countRoot;i++){

tmp=heads[i];

while(tmp!=NULL){

if(tmp->data==a)

return tmp->rep;

tmp=tmp->next;

}

}

return NULL;}

void unionSets(int a,int b){

int i,pos,flag=0,j;

struct node \*tail2=(struct node \*)malloc(sizeof(struct node));

struct node \*rep1=find(a);

struct node \*rep2=find(b);

if(rep1==NULL||rep2==NULL){

printf("\nElement not present in the DS\n");

return;

}

if(rep1!=rep2){

for(j=0;j<countRoot;j++){

if(heads[j]==rep2){

pos=j;

flag=1;

countRoot-=1;

tail2=tails[j];

for(i=pos;i<countRoot;i++){

heads[i]=heads[i+1];

tails[i]=tails[i+1];

}}

if(flag==1)

break;}

for(j=0;j<countRoot;j++){

if(heads[j]==rep1){

tails[j]->next=rep2;

tails[j]=tail2;

break;

}}

while(rep2!=NULL){

rep2->rep=rep1;

rep2=rep2->next;

}}}

int search(int x){

int i;

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

for(i=0;i<countRoot;i++){

tmp=heads[i];

if(heads[i]->data==x)

return 1;

while(tmp!=NULL){

if(tmp->data==x)

return 1;

tmp=tmp->next;

} } return 0; }

void main(){

int choice,x,i,j,y,flag=0;

do{

printf("\n.......MENU.......\n\n1.Make Set\n2.Display set representatives\n3.Union\n4.Find Set\n5.Exit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice){

case 1:

printf("\nEnter new element : ");

scanf("%d",&x);

if(search(x)==1)

printf("\nElement already present in the disjoint set DS\n");

else

makeSet(x);

break;

case 2:

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

printf("%d ",heads[i]->data);

break;

case 3:

printf("\nEnter first element : ");

scanf("%d",&x);

printf("\nEnter second element : ");

scanf("%d",&y);

unionSets(x,y);

break;

case 4:

printf("\nEnter the element");

scanf("%d",&x);

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

rep=find(x);

if(rep==NULL)

printf("\nElement not present in the DS\n");

else

printf("\nThe representative of %d is %d\n",x,rep->data);

break;

case 5:

exit(0);

default:

printf("\nWrong choice\n");

break;

}

}

while(1);

};

**PROGRAM – 3**

**AIM:** C program to implement Stack Using linked List

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100

int stack[MAX];

int top=-1;

int max=100;

bool isEmpty(){

if(top<=-1){

return 1;

}

else{

return 0;

}

}

bool isFull(){

if(top>=max){

return 1;

}

else{

return 0;

}

}

void push(){

int value;

if(isFull()){

printf("Stack is full\n");

}

else{

printf("Enter push item : ");

scanf("%d",&value);

top=top+1;

stack[top]=value;

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

}

}

void pop(){

if(isEmpty()){

printf("stack is empty");

}

else{

printf("%d is poped out : ",stack[top]);

top=top-1;

}

}

void display(){

int i;

if(!isEmpty()){

for(i=0;i<=top;i++){

printf("stack[%d] = %d \n",i+1,stack[i]);

}

}

else{

printf("stack is empty");

}

}

void peek(){

if(!isEmpty()){

printf("top element is : %d",stack[top]);

}

else{

printf("stack is empty");

}

}

int main() {

int opt;

do{

printf("\n..................\n");

printf("1)PUSH\n");

printf("2)POP\n");

printf("3)PEEK\n");

printf("4)DISPLAY\n");

printf("5)QUIET");

printf("\n..................\n");

printf("choose your option : ");

scanf("%d",&opt);

switch(opt){

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

display();

break;

case 5:

exit(0);

default:

printf("\nInvalid Operation! Try Again... \n");

}

}while(1);

return 0;

}

**PROGRAM – 4**

**AIM:** C program to implement  BitString

#include<stdio.h>

#include<stdlib.h>

int superSet[20],superSetSize=0,

setA[20],setASize=0,

setB[20],setBSize=0,

bitStringA[20],bitStringB[20],bitStringUnion[20],bitStringIntersection[20],bitStringDifference[20];

int isBitStringReady=0;

//method declarations

void getSet(int arr[],int setSize);

void processChoice(int choice);

void printAllSets();

void printSet(int arr[],int size);

void generateAndPrintBitStrings();

void generateBitString(int arr[],int size,int bitStringArray[]);

int search(int arr[],int arrSize,int elem);

int checkBitStringStatus();

void setUnion(int arr1[],int arr2[]);

void setIntersection(int arr1[],int arr2[]);

void setDifference(int arr1[],int arr2[]);

void printSetOperationResult(int arr[]);

void printBitStringAsSet(int arr[]);

int checkBitStringStatus();

int main(){

    int choice=0;

    system("cls");

    do{

        printf("\n============================\n");

        printf("1.Enter Universal Set\n");

        printf("2.Enter Set A\n");

        printf("3.Enter Set B\n");

        printf("4.Generate Bit Strings\n");

        printf("5.Union\n");

        printf("6.Intersection\n");

        printf("7.Difference\n");

        printf("8.Print Sets\n");

        printf("9.Exit\n");

        printf("============================\n");

        printf("Enter Choice:");

        scanf("%d",&choice);

        printf("============================\n");

        processChoice(choice);

    }while(choice!=9);

    return 0;

}

void processChoice(int choice){

    switch(choice){

        case 1:

            printf("Enter Super Set Size:");

            scanf("%d",&superSetSize);

            getSet(superSet,superSetSize);

            break;

        case 2:

            printf("Enter Set A Size:");

            scanf("%d",&setASize);

            getSet(setA,setASize);

            break;

        case 3:

            printf("Enter Set B Size:");

            scanf("%d",&setBSize);

            getSet(setB,setBSize);

             break;

        case 4:

            printf("Generating bit strings\n");

            generateAndPrintBitStrings();

             break;

        case 5:

            printf("Set union\n");

            if(checkBitStringStatus()==1)

            {

                setUnion(bitStringA,bitStringB);

                printSetOperationResult(bitStringUnion);

            }

             break;

        case 6:

            printf("Set Intersection\n");

            if(checkBitStringStatus()==1)

            {

                setIntersection(bitStringA,bitStringB);

                printSetOperationResult(bitStringIntersection);

            }

             break;

        case 7:

            printf("Set Difference\n");

            if(checkBitStringStatus()==1)

            {

                setDifference(bitStringA,bitStringB);

                printSetOperationResult(bitStringDifference);

            }

             break;

        case 8:

            printAllSets();

             break;

    }

}

void printSetOperationResult(int arr[]){

    printf("\nUnion Operation (bit string):");

    printSet(arr,superSetSize);

    printf("\nOperation Result:");

    printBitStringAsSet(arr);

}

void printBitStringAsSet(int arr[]){

    int isFirstOutputDoneFlag=0;

    int i=0;

    printf("{");

    for( i=0;i<superSetSize;i++){

        if(arr[i]==1){

            if(i!=0 && isFirstOutputDoneFlag==1){

                printf(",");

            }

            printf("%d",superSet[i]);

            isFirstOutputDoneFlag=1;

        }

    }

    printf("}");

}

void setUnion(int arr1[],int arr2[]){

    int i;

    for(i=0;i<superSetSize;i++){

        bitStringUnion[i]=arr1[i]|arr2[i];

    }

}

void setIntersection(int arr1[],int arr2[]){

    int i;

    for(i=0;i<superSetSize;i++){

        bitStringIntersection[i]=arr1[i]&arr2[i];

    }

}

void setDifference(int arr1[],int arr2[]){

    int i;

    for(i=0;i<superSetSize;i++){

        printf("%d",!arr2[i]) ;

        bitStringDifference[i]=arr1[i]&(!arr2[i]);

    }

}

int checkBitStringStatus(){

    if(isBitStringReady==0){

        printf("\n Generate Bit String first!");

        return 0;

    }

    return 1;

}

void getSet(int arr[],int setSize){

    int i;

    printf("\nEnter set\n");

    for(i=0;i<setSize;i++){

        scanf("%d",&arr[i]);

    }

}

void printAllSets(){

    printf("\nSuper Set:");

    printSet(superSet,superSetSize);

    printf("Set A:");

    printSet(setA,setASize);

    printf("Set B:");

    printSet(setB,setBSize);

}

void printSet(int arr[],int size){

    int i;

    printf("{");

    for(i=0;i<size;i++){

        printf("%d",arr[i]);

        if(i!=size-1){

            printf(",");

        }

    }

    printf("}\n");

}

void generateAndPrintBitStrings(){

    int i;

    for(i=0;i<superSetSize;i++){

        bitStringA[i]=0;

        bitStringB[i]=0;

        bitStringUnion[i]=0;

        bitStringIntersection[i]=0;

        bitStringDifference[i]=0;

    }

    //generate setA bitString

    generateBitString(setA,setASize,bitStringA);

    //generate setB bitString

    generateBitString(setB,setBSize,bitStringB);

    //print the bit strings

    printf("\nSet A Bit String representation : ");

    printSet(bitStringA,superSetSize);

    printf("\nSet B Bit String representation : ");

    printSet(bitStringB,superSetSize);

    isBitStringReady=1;

}

void generateBitString(int arr[],int size,int bitStringArray[]){

    int i;

    for(i=0;i<size;i++){

        int pos=search(superSet,superSetSize,arr[i]);

        if(pos>=0){

            bitStringArray[pos]=1;

        }

    }

}

int search(int arr[],int arrSize,int elem){

    int i;

    for(i=0;i<arrSize;i++){

        if(arr[i]==elem)

            return i;

    }

    return -1;

}

**PROGRAM – 5**

**AIM:** C program to implement DFS (Depth First Search)

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 6

//vertex count

int vertex\_count =0;

// vertex definitions

struct vertex{

char data;

bool visited;

};

//array of vertices

struct vertex \*graph[MAX];

//adjacency matrix

int adj\_matrix[MAX][MAX];

//stack

int stack[MAX];

int top = -1;

void push(int data){

stack[++top]=data;

}

int pop(){

return stack[top--];

}

int peek(){

return stack[top];

}

bool is\_stack\_empty(){

return top == -1;

}

//add vertex to the vertex list

void add\_vertex(char data){

struct vertex \*new = (struct vertex\*)malloc(sizeof(struct vertex));

new->data = data;

new->visited = false;

graph[vertex\_count]=new;

vertex\_count++;

}

//add edge to edge array

void add\_edge(int start,int end){

adj\_matrix[start][end]=1;

adj\_matrix[end][start]=1;

}

// to return adjacent vertex

int adj\_vertex(int vertex\_get){

int i;

for(i=0;i<vertex\_count;i++){

if(adj\_matrix[vertex\_get][i] == 1 && graph[i]->visited == false){

return i;

}

}

return -1;

}

// to display vertex value

void display\_vertex(int pos){

printf("%c",graph[pos]->data);

}

void dfs(){

int i;

int unvisited;

printf("\n|||||||||||||||||||||||||||||||\n");

graph[0]->visited =true;

display\_vertex(0);

push(0);

while(!is\_stack\_empty()){

int unvisited = adj\_vertex(peek());

if(unvisited == -1){

pop();

}

else{

graph[unvisited]->visited = true;

display\_vertex(unvisited);

push(unvisited);

}

}

printf("\n|||||||||||||||||||||||||||||||\n");

for(i=0;i<vertex\_count;i++){

graph[i]->visited = false;

}

}

void show(){

int i;

printf("\n.................................\n");

for(i=0;i<vertex\_count;i++){

printf("Edge postion of '%c' is %d\n",graph[i]->data,i);

}

printf(".................................\n");

}

int main(){

int opt;

char data;

int edge\_1,edge\_2;

int i, j;

for(i = 0; i < MAX; i++) // set adjacency

for(j = 0; j < MAX; j++) // matrix to 0

adj\_matrix[i][j] = 0;

do{

printf("\n1)Add vertex \n2)Create edge \n3)Traversal \n0)Exit \nChoose option :: ");

scanf("%d",&opt);

switch(opt){

case 1:

printf("\nEnter data to be added to vertex : ");

scanf(" %c", &data);

add\_vertex(data);

break;

case 2:

show();

printf("\nEnter edge starting : ");

scanf("%d",&edge\_1);

printf("\nEnter edge ending : ");

scanf("%d",&edge\_2);

if(vertex\_count-1 < edge\_1 || vertex\_count-1 < edge\_2){

printf("\nThere is no vertex !!\n");

}

else{

add\_edge(edge\_1,edge\_2);

}

break;

case 3:

dfs();

break;

default:

printf("\nInvalid option try again !! ...");

}

}while(opt!=0);

return 0;}

**PROGRAM – 6**

**AIM:** C program to implement kruskal ‘s Algorithm

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int parent[MAX];

int find(int i){

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j){

if(i!=j){

parent[j]=i;

return 1;

}

return 0;

}

int main(){

int vertex\_count=0;

int row,column;

int cost\_matrix[MAX][MAX];

int edge\_count=0,count=1;

int sum\_cost=0,min\_cost;

int row\_no,column\_no,edge1,edge2;

printf("Implementation of Kruskal's algorithm\n\n");

printf("Total no of vertex :: ");

scanf("%d",&vertex\_count);

//Get edge weight matrix from user

for(row=1;row<=vertex\_count;row++){

for(column=1;column<=vertex\_count;column++){

scanf("%d",&cost\_matrix[row][column]);

if(cost\_matrix[row][column] == 0){

cost\_matrix[row][column] = 999;

}

}

}

edge\_count = vertex\_count-1;

while(count <= edge\_count){

for(row=1,min\_cost=999;row<=vertex\_count;row++){

for(column=1;column<=vertex\_count;column++){

if(cost\_matrix[row][column] < min\_cost){

min\_cost = cost\_matrix[row][column];

edge1 = row\_no = row;

edge2 = column\_no = column;

}

}

}

row\_no = find(row\_no);

column\_no = find(column\_no);

if(uni(row\_no,column\_no)){

printf("\nEdge %d is (%d -> %d) with cost : %d ",count++,edge1,edge2,min\_cost);

sum\_cost = sum\_cost + min\_cost;

}

cost\_matrix[edge1][edge2] = cost\_matrix[edge2][edge1] = 999;

}

printf("\n Minimum cost=%d",sum\_cost);

return 0;

}

**PROGRAM – 7**

**AIM:** C program to implement Prims Algorithm

#include<stdio.h>

#define MAX 10

int main(){

int vertex\_array[MAX],counter;

int vertex\_count=0;

int row,column;

int cost\_matrix[MAX][MAX];

int visited[MAX]={0};

int edge\_count=0,count=1;

int sum\_cost=0,min\_cost=0;

int row\_no,column\_no,vertex1,vertex2;

printf("Total no of vertex :: ");

scanf("%d",&vertex\_count);

printf("\n-- Enter vertex -- \n\n");

for(counter=1;counter<=vertex\_count;counter++){

printf("vertex[%d] :: ",counter);

scanf("%d",&vertex\_array[counter]);

}

printf("\n--- Enter Cost matrix of size %d x %d ---\n\n",vertex\_count,vertex\_count);

printf("\n\t-- format is --\n");

for(row=1;row<=vertex\_count;row++){

for(column=1;column<=vertex\_count;column++){

printf("x ");

}

printf("\n");

}

printf("\n-- MATRIX --\n\n");

//Get edge weight matrix from user

for(row=1;row<=vertex\_count;row++){

for(column=1;column<=vertex\_count;column++){

scanf("%d",&cost\_matrix[row][column]);

if(cost\_matrix[row][column] == 0){

cost\_matrix[row][column] = 999;

}

}

}

printf("\n");

visited[1]=1;

edge\_count = vertex\_count-1;

while(count <= edge\_count){

for(row=1,min\_cost=999;row<=vertex\_count;row++){

for(column=1;column<=vertex\_count;column++){

if(cost\_matrix[row][column] < min\_cost){

if(visited[row] != 0){

min\_cost = cost\_matrix[row][column];

vertex1 = row\_no = row;

vertex2 = column\_no = column;

}

}

}

}

if(visited[row\_no] == 0 || visited[column\_no] ==0){

printf("\nEdge %d is (%d -> %d) with cost : %d ",count++,vertex\_array[vertex1],vertex\_array[vertex2],min\_cost);

sum\_cost = sum\_cost + min\_cost;

visited[column\_no]=1;

}

cost\_matrix[vertex1][vertex2] = cost\_matrix[vertex2][vertex1] = 999;

}

printf("\n\nMinimum cost=%d",sum\_cost);

return 0;

}

**PROGRAM – 8**

**AIM:** C program to implement Circular Queue

#include<stdio.h>

#define MAX 10

void enqueue();

void dequeue();

void search();

void display();

int c,queue[MAX],item,front=-1,rear=-1,i,n,flag=0;

void main(){

do

{

printf("\n1)Insertion\n2)Deletion\n3)Search\n4)Display\n5)Exit\n");

scanf("%d",&c);

switch(c)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

search();

break;

case 4:

display();

break;

case 5:

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

break;

default:

printf("\n enter a valied choice 1 or 2 or 3 or 4 or 5");

}

}while(c!=5);

}

void enqueue()

{

if((rear+1)%MAX==front){

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

}

else

{

if(rear==-1)

front=0;

printf("\nEnter the element to insert : ");

scanf("%d",&item);

rear=(rear+1)%MAX;

queue[rear]=item;

}

}

void dequeue(){

if(front==-1){

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

}

else if(front==rear){

printf("\nThe deleted element is:%d\n",queue[front]);

front=rear=-1;

}

else{

printf("\nThe deleted element is:%d\n",queue[front]);

front=(front+1)%MAX;

}

}

void search(){

printf("\nEnter the value to search\n");

scanf("%d",&n);

if(front==-1){

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

}

else{

int front\_pos=front,rear\_pos=rear;

if(front\_pos>rear\_pos){

while(front\_pos>rear\_pos)

{

if(queue[front\_pos]==n){

flag=1;

break;

}

front\_pos=((front\_pos+1))%MAX;

}

}

if(front\_pos<=rear\_pos){

for(i=front\_pos;i<=rear\_pos;i++){

if(queue[i]==n){

flag=1;

break;

}

}

}

}

if(flag==1)

printf("\nElement is found ");

else

printf("\nElement is not found\n");

}

void display(){

if(front==-1){

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

}

else{

printf("\n.............................\n");

int front\_pos=front,rear\_pos=rear;

if(front\_pos>rear\_pos){

while(front\_pos>rear\_pos)

{

printf(" %d ",queue[front\_pos]);

front\_pos=((front\_pos+1))%MAX;

}

}

if(front\_pos<=rear\_pos){

for(i=front\_pos;i<=rear\_pos;i++)

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

}

printf("\n.............................\n");

}

}

**PROGRAM – 9**

**AIM:** C program to implement Binary Search Tree

#include<stdio.h>

#include<stdlib.h>

struct node{

struct node \*left;

struct node \*right;

int data;

};

struct node \*root;

struct node\* newNode(value){

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

newnode->data = value;

newnode->left=NULL;

newnode->right=NULL;

return newnode;

}

struct node\* insert(struct node\* root,int value) {

if(root == NULL){

return newNode(value);

}

else if(value == root->data){

printf("Same data can't be stored");

}

else if(value>root->data){

root->right = insert(root->right,value);

}

else if(value<root->data){

root->left = insert(root->left,value);

}

return root;

}

// Inorder traversal

void inorderTraversal(struct node\* root) {

if (root == NULL) return;

inorderTraversal(root->left);

printf("%d ->", root->data);

inorderTraversal(root->right);

}

// Preorder traversal

void preorderTraversal(struct node\* root) {

if (root == NULL) return;

printf("%d ->", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

// Postorder traversal

void postorderTraversal(struct node\* root) {

if (root == NULL) return;

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d ->", root->data);

}

struct node\* search(struct node\* root, int key) {

if (root == NULL)

printf("\nNot FOUND!\n");

else if (root->data == key)

printf("\nFOUND!\n");

else{

if (root->data < key)

return search(root->right, key);

return search(root->left, key);

}

}

struct node\* minValueNode(struct node\* node){

struct node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key){

if (root == NULL)

return root;

if (key < root->data)

root->left = deleteNode(root->left, key);

else if (key > root->data)

root->right = deleteNode(root->right, key);

else {

// node with only one child or no child

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// node with two children:

// Get the inorder successor

// (smallest in the right subtree)

struct node\* temp = minValueNode(root->right);

// Copy the inorder

// successor's content to this node

root->data = temp->data;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->data);

}

return root;

}

int main(){

int opt;

int value,searchv,key;

do{

printf("\n1)Create Root Node \n2)Insert Node\n3)Search\n");

printf("4)inorderTraversal \n5)preorderTraversal \n6)postorderTraversal \n7)Delete \n8)Quiet \n");

printf("Choose Option :: ");

scanf("%d",&opt);

switch(opt){

case 1:

printf("\nEnter a number : ");

scanf("%d",&value);

root = newNode(value);

break;

case 2:

printf("\nEnter a number : ");

scanf("%d",&value);

root = insert(root,value);

break;

case 3:

printf("\nEnter a number : ");

scanf("%d",&searchv);

search(root,searchv);

break;

case 4:

printf("\n..................................\n");

inorderTraversal(root);

printf("\n..................................\n");

break;

case 5:

printf("\n..................................\n");

preorderTraversal(root);

printf("\n..................................\n");

break;

case 6:

printf("\n..................................\n");

postorderTraversal(root);

printf("\n..................................\n");

break;

case 7:

printf("\nEnter a number to be deleted : ");

scanf("%d",&key);

deleteNode(root,key);

break;

defualt:

printf("Invalid option!");

}

}while(opt!=8);

return 0;

}