1: Write a C Program to perform following operations on a binary search tree

* DISPLAY
* SEARCH
* INSERT
* DELETE

Program:

#include<stdio.h>

#include<stdlib.h>

# define MAX 10

struct node \*insert(struct node \*ptr, int ikey);

struct node \*search(struct node \*ptr, int skey);

struct node \*del(struct node \*ptr, int dkey);

void preorder(struct node \*ptr);

void inorder(struct node \*ptr);

void postorder(struct node \*ptr);

struct node{

struct node \*lchild;

int info;

struct node \*rchild;

};

int main(){

struct node \*root = NULL, \*ptr;

int ch, data;

while(1){

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

printf("\n1. Insert \n2. Search \n3. Delete \n4. Display\n5. Quit");

printf("\nEnter Choice: ");

scanf("%d",&ch);

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

switch(ch){

case 1: printf("\nEnter element to be inserted: ");

scanf("%d", &data);

root = insert(root,data);

break;

case 2: printf("\nEnter element to be searched: ");

scanf("%d", &data);

ptr = search(root,data);

if(ptr== NULL)

printf("\nKey not found");

else

printf("\nKey found");

break;

case 3: printf("\nEnter element to be deleted: ");

scanf("%d", &data);

root = del(root,data);

break;

case 4: printf("\nPreorder: ");

preorder(root);

printf("\nInorder: ");

inorder(root);

printf("\nPostorder: ");

postorder(root);

break;

case 5: exit(1);

default: printf("\nWRONG CHOICE");

break;

}

}

}

struct node \*insert(struct node \*ptr, int ikey){

if(ptr==NULL){

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

ptr->info = ikey;

ptr->lchild = NULL;

ptr->rchild = NULL;

}

else if(ikey < ptr->info)

ptr->lchild = insert(ptr->lchild, ikey);

else if(ikey > ptr->info)

ptr->rchild = insert(ptr->rchild, ikey);

else

printf("\nDuplicate Key");

return ptr;

}

struct node \*search(struct node \*ptr, int skey){

if(ptr == NULL){

printf("\nKey not found");

return NULL;

}

else if(skey < ptr->info)

return search(ptr->lchild, skey);

else if(skey > ptr->info)

return search(ptr->rchild, skey);

else

return ptr;

}

struct node \*del(struct node \*ptr, int dkey){

struct node \*temp, \*succ;

if(ptr == NULL){

printf("\nKey not present in tree");

return ptr;

}

if(dkey < ptr->info)

ptr->lchild = del(ptr->lchild, dkey);

else if(dkey > ptr->info)

ptr->rchild = del(ptr->rchild, dkey);

else{

if(ptr->lchild != NULL && ptr->rchild != NULL){

succ = ptr->rchild;

while(succ->lchild != NULL)

succ = succ->lchild;

ptr->info = succ->info;

ptr->rchild = del(ptr->rchild, succ->info);

}

else{

temp = ptr;

if(ptr->lchild != NULL)

ptr = ptr->lchild;

else if(ptr->rchild != NULL)

ptr = ptr->rchild;

else

ptr = NULL;

free(temp);

}

}

return ptr;

}

void preorder(struct node \*ptr){

if(ptr == NULL)

return;

printf(" %d", ptr->info);

preorder(ptr->lchild);

preorder(ptr->rchild);

}

void inorder(struct node \*ptr){

if(ptr == NULL)

return;

inorder(ptr->lchild);

printf("%d ", ptr->info);

inorder(ptr->rchild);

}

void postorder(struct node \*ptr){

if(ptr == NULL)

return;

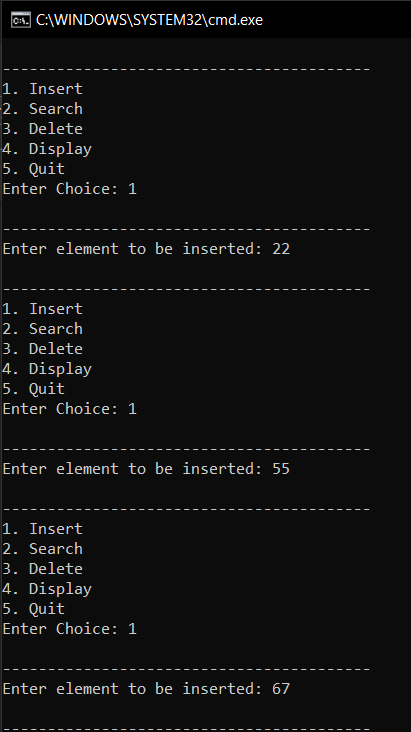
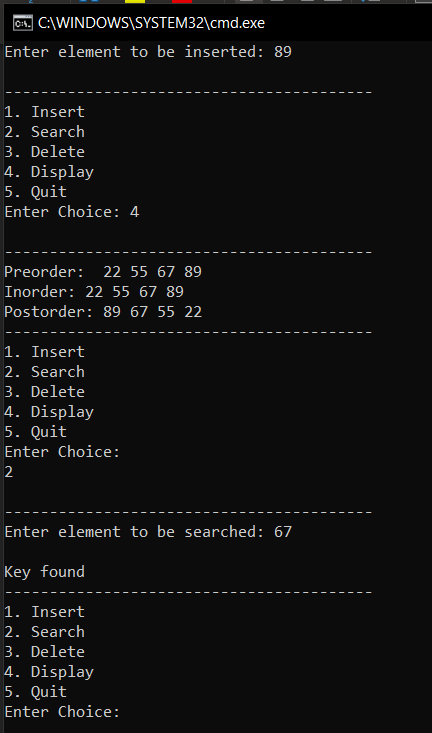
postorder(ptr->lchild);

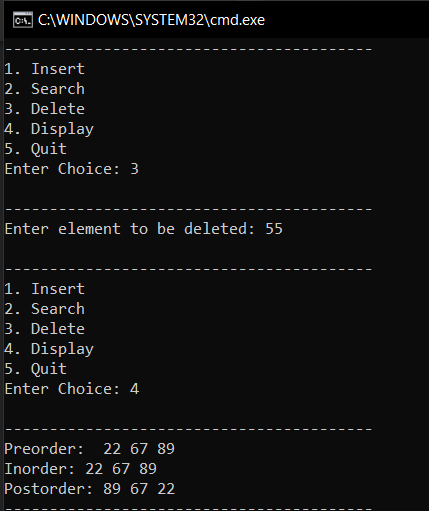
postorder(ptr->rchild);

printf("%d ", ptr->info);

}

OUTPUT:



2. Write a program to implement following traversals in a binary tree

* In order
* Pre Order
* Post Order
* Level Order

Program:

#include<stdio.h>

#include<stdlib.h>

# define MAX 10

struct node \*tree[MAX];

int front=-1, rear=-1;

struct node \*insert(struct node \*ptr, int item);

void preorder(struct node \*ptr);

void inorder(struct node \*ptr);

void postorder(struct node \*ptr);

void level\_trav(struct node \*root);

void insert\_tree(struct node \*item);

struct node \*del\_queue();

int queue\_empty();

struct node{

struct node \*lchild;

int info;

struct node \*rchild;

};

int main(){

struct node \*root = NULL;

int i, k;

printf("Enter values:\n");

for(i=0; i<10; i++){

scanf("%d", &k);

root = insert(root, k);

}

printf("\nPreorder: ");preorder(root);

printf("\nInorder: ");inorder(root);

printf("\nPostorder: ");postorder(root);

printf("\nLevel order: ");level\_trav(root);

}

struct node \*insert(struct node \*ptr, int item){

if(ptr==NULL){

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

ptr->info = item;

ptr->lchild = NULL;

ptr->rchild = NULL;

}

else if(item < ptr->info)

ptr->lchild = insert(ptr->lchild, item);

else if(item > ptr->info)

ptr->rchild = insert(ptr->rchild, item);

else

printf("\nDuplicate Key");

return ptr;

}

void preorder(struct node \*ptr){

if(ptr == NULL)

return;

printf("%d ", ptr->info);

preorder(ptr->lchild);

preorder(ptr->rchild);

}

void inorder(struct node \*ptr){

if(ptr == NULL)

return;

inorder(ptr->lchild);

printf("%d ", ptr->info);

inorder(ptr->rchild);

}

void postorder(struct node \*ptr){

if(ptr == NULL)

return;

postorder(ptr->lchild);

postorder(ptr->rchild);

printf("%d ", ptr->info);

}

void insert\_tree(struct node \*item){

if(rear == MAX-1){

printf("Queue Overflow\n");

return;

}

if(front == -1)

front = 0;

rear = rear + 1;

tree[rear] = item;

}

struct node \*del\_queue(){

struct node \*item;

if(queue\_empty()){

printf("Queue Underflow\n");

return 0;

}

item = tree[front];

front = front + 1;

return item;

}

int queue\_empty(){

if(front == -1 || front == rear+1)

return 1;

else

return 0;

}

void level\_trav(struct node \*root){

struct node \*ptr = root;

if(ptr == NULL){

printf("Tree is empty\n");

return;

}

insert\_tree(ptr);

while(!queue\_empty()){

ptr = del\_queue();

printf("%d ", ptr->info);

if(ptr->lchild != NULL)

insert\_tree(ptr->lchild);

if(ptr->rchild != NULL)

insert\_tree(ptr->rchild);

}

printf("\n");

}

