**CODE:**

/\*

INORDER=LEft->Parent->Right

PREORDER=Parent->Left->Right

POSTORDER=Left->Right->Parent

\*/

#include<stdio.h>

#include<malloc.h>

typedef struct node

{

struct node \*left;

int data;

struct node \*right;

}NODE;

NODE \*root,\*tptr,\*prev;

void preOrder(NODE \*print)

{

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

if(print->left!=NULL)

preOrder(print->left);

if(print->right!=NULL)

preOrder(print->right);

}

void inorder(NODE \*print)

{

if(print->left)

inorder(print->left);

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

if(print->right)

inorder(print->right);

}

void inorderrev(NODE \*print)

{

if(print->right)

inorder(print->right);

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

if(print->left)

inorder(print->left);

}

void postorder(NODE \*print)

{

if(print->left)

postorder(print->left);

if(print->right)

postorder(print->right);

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

}

void addChild(int givenData)

{

NODE \*newnode;

newnode=(NODE\*)malloc(sizeof(NODE));

newnode->data=givenData;

newnode->left=newnode->right=NULL;

if(root==NULL)

root=newnode;

else

{

tptr=root;

while(tptr)

{

prev=tptr;

if(tptr->data<givenData)

tptr=tptr->right;

else if(tptr->data>givenData)

tptr=tptr->left;

else

break;

}

if(tptr==NULL)

{

if(prev->data<givenData)

prev->right=newnode;

else

prev->left=newnode;

}

}

}

void deleten(int givenData)

{

tptr=root;

while(tptr->data!=givenData && tptr)

{

prev=tptr;

if(tptr->data>givenData)

{

tptr=tptr->left;

}

else //if(tptr->data<givenData)

{

tptr=tptr->right;

}

}

if(tptr->data==givenData)

{

if(tptr->left==NULL&&tptr->right==NULL&&tptr!=NULL)

{

if(prev->data>givenData)

{

prev->left=NULL;

}

else

{

prev->right=NULL;

}

free(tptr);

}

else if((tptr->left!=NULL||tptr->right!=NULL)&&(tptr!=NULL))

{

if(tptr==prev->right)

{

if(tptr->left!=NULL)

{

prev->right=tptr->left;

tptr->left=NULL;

free(tptr);

}

else if(tptr->right!=NULL)

{

prev->right=tptr->right;

tptr->right=NULL;

free(tptr);

}

}

else //(tptr==prev->left)

{

if(tptr->left!=NULL)

{

prev->left=tptr->left;

tptr->left=NULL;

free(tptr);

}

else if(tptr->right!=NULL)

{

prev->left=tptr->right;

tptr->right=NULL;

free(tptr);

}

}

}

}

postorder(root);

}

/\*else if(tptr->left!=NULL&&tptr->right!=NULL)

{

case for deleting a root node

Find the smallest element in the right side

Find the largest element in the left side

Make it is a root node

Delete older root node

}\*/

int main()

{

int num,num2;

do

{

scanf("%d",&num);

if(num==-1)

{

break;

}

else

{

addChild(num);

}

}while(1);

printf("\nPreOrder Display:");

preOrder(root);

printf("\nInOrder Display:");

inorder(root);

printf("\nPostOrder Display:");

postorder(root);

printf("\nEnter a number to delete:\n");

scanf("%d",&num2);

printf("\nAfter Deletion POST Order display:");

deleten(num2);

printf("\nReversal InOrder Display:");

inorderrev(root);//reverse display of inorder traversal

return 0;

}

**OUTPUT:**

