

AVL TREE

```
#include<stdio.h>
```

```
struct node
```

```
{
```

```
int data,ht;
```

```
struct node *left,*right;
```

```
}*root=NULL,*temp;
```

```
struct node *insert(struct node *,int);
```

```
struct node *Delete(struct node *,int);
```

```
void preorder(struct node *);
```

```
void inorder(struct node *);
```

```
void postorder(struct node *);
```

```
int height(struct node *);
```

```
struct node *rotateright(struct node *);
```

```
struct node *rotateleft(struct node *);
```

```
struct node *RR(struct node *);
```

```
struct node *LL(struct node *);
```

```
struct node *LR(struct node *);
```

```
struct node *RL(struct node *);
```

```
int BF(struct node *);
```

```
main()
```

```
{
```

```

int x,n,i,op;
do
{
    printf("\n1)Create:\n2)Insert:\n3)Delete:\n4)Print:\n5)Quit:");
    printf("\n\nEnter Your Choice:");
    scanf("%d",&op);
    switch(op)
    {
        case 1:printf("\nEnter no. of elements:");
            scanf("%d",&n);
            printf("\nEnter tree data:");
            root=NULL;
            for(i=0;i<n;i++)
            {
                scanf("%d",&x);
                root=insert(root,x);
            }
            break;

        case 2:
            printf("\nEnter a data:");
            scanf("%d",&x);
            root=insert(root,x);
            break;

        case 3:
            printf("\nEnter a data:");
            scanf("%d",&x);
            root=Delete(root,x);
            break;

        case 4:

```

```

printf("\nPreordersequence:\n");
    preorder(root);
    printf("\nInordersequence:\n");
    inorder(root);
    printf("\n");
}
}while(op!=5);
}

```

```

struct node* insert(struct node *root,int x)
{
    if(root==NULL)
    {
        temp =(struct node*)malloc(sizeof(struct node));
        temp->data=x;
        temp->left=NULL;
        temp->right=NULL;
return (temp);
    }
}

```

```

else if(x< root->data) /* insert in left subtree*/

```

```

{
    root->left=insert(root->left,x);

```

```

    if(BF(root)==2)

```

```

{
    if(x < root->left->data)
        root=LL(root);
    else
        root=LR(root);
}

```

```

    }
    else if(x > root->data) /* insert in right subtree*/
    {
        root->right=insert(root->right,x);

        if(BF(root)==-2)
        {
            if(x>root->right->data)
                root=RR(root);
            else
                root=RL(root);
        }
    }

    root->ht=height(root);
    return(root);
}

```

```

int height(struct node *root)
{
    int lh,rh;
    if(root==NULL)
        return(0);

    if(root->left==NULL)
        lh=0;
    else
        lh=1+root->left->ht;

```

```

    if(root->right==NULL)
        rh=0;
    else
        rh=1+root->right->ht;

    if(lh>rh)
        return(lh);

    return(rh);
}

```

```

struct node * rotateright(struct node *x)
{
    struct node *y;

    y=x->left;
    x->left=NULL;
    y->right=x;

    x->ht=height(x);
    y->ht=height(y);

    return(y);
}

```

```

struct node *rotateleft(struct node *x)
{
    struct node *y;
    y=x->right;

```

```
x->right=NULL;

y->left=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}
```

```
struct node *RR(struct node *root)
{
    root=rotateleft(root);
    return(root);
}
```

```
struct node *LL(struct node *root)
{
    root=rotateright(root);
    return(root);
}
```

```
struct node *LR(struct node *root)
{
    root->left=rotateleft(root->left);
    root=rotateright(root);
    return(root);
}
```

```
struct node *RL(struct node *root)
{
    root->right=rotateright(root->right);
```

```
        root=rotateleft(root);  
        return(root);  
    }
```

```
int BF(struct node *root)  
{  
    int lh,rh;  
  
    if(root==NULL)  
        return(0);  
  
    if(root->left==NULL)  
        lh=0;  
    else  
        lh=1+root->left->ht;  
  
    if(root->right==NULL)  
        rh=0;  
    else  
        rh=1+root->right->ht;  
  
    return(lh-rh);  
}
```

```
void preorder(struct node *root)  
{  
    if(root!=NULL)  
    {
```

```

        printf("%d(Bf=%d)\t",root->data,BF(root));
        preorder(root->left);
        preorder(root->right);
    }
}

```

```

void inorder(struct node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d(Bf=%d)\t",root->data,BF(root));
        inorder(root->right);
    }
}

```

```

void postorder(struct node *root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d(Bf=%d)\t",root->data,BF(root));
    }
}

```

```

struct node *Delete(struct node *root,int x)
{

```



```

struct node *p;
if(root==NULL)

{
    return NULL;
}
else if(x> root->data)/* insert in right subtree*/
{
    root->right=Delete(root->right,x);
    if(BF(root)==2)
    if(BF(root->left)>=0)
    root=LL(root);
    else
    root=LR(root);
}
else if(x<root->data)
{
    root->left=Delete(root->left,x);
    if(BF(root)==-2)/Rebalanceduring windup/
    if(BF(root->right)<=0)
        root=RR(root);
    else
        root=RL(root);
}
else
{
    /data to be deleted is found/
    if(root->right!=NULL)
    { /delete its inordersuccesor/
        p=root->right;
        while(p->left!= NULL)

```

```

    p=p->left;

    root->data=p->data;

    root->right=Delete(root->right,p->data);

if(BF(root)==2)/Rebalanceduring windup/
    if(BF(root->left)>=0)

        root=LL(root);

    else

        root=LR(root);
}

else

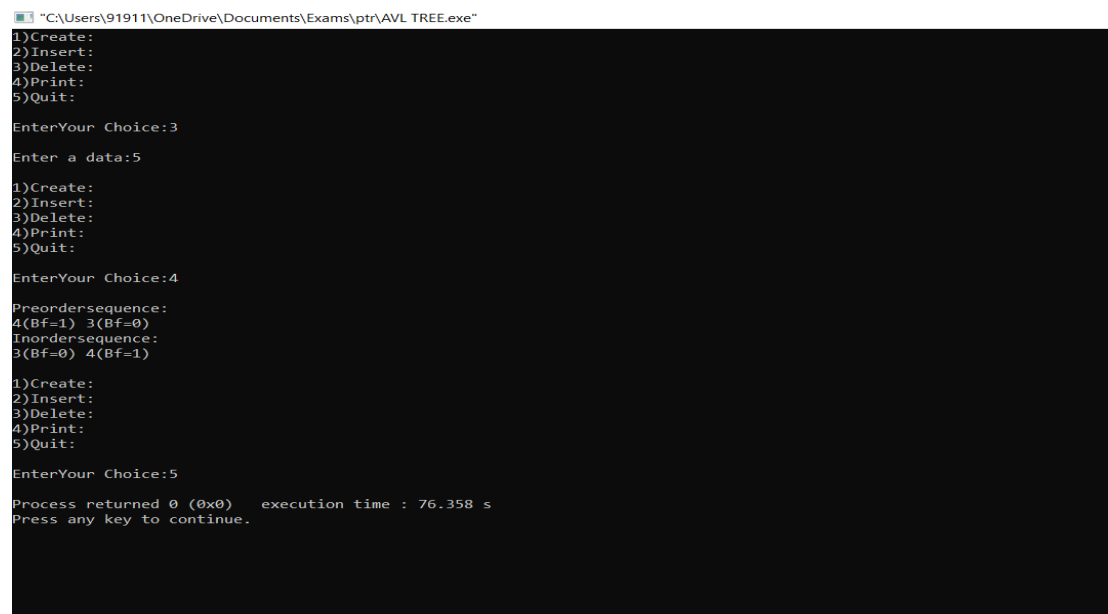
    return(root->left);
}

root->ht=height(root);

return(root);
}

```

OUTPUT:



```

"C:\Users\91911\OneDrive\Documents\Exams\ptr\AVL TREE.exe"
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

EnterYour Choice:3

Enter a data:5

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

EnterYour Choice:4

Preordersequence:
4(Bf=1) 3(Bf=0)
Inordersequence:
3(Bf=0) 4(Bf=1)

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

EnterYour Choice:5

Process returned 0 (0x0) execution time : 76.358 s
Press any key to continue.

```