Name - Shantanu Deshpande
class - SYCSE
Roll no. - 7
Batch - S1
Sub. - D.S

## Practical No. 6

Aim :- Implementation of operation on BST.

Theory :-

### Binary search tree -

In computer science, a binary search tree (BST) is a binary data structure which has the following properties

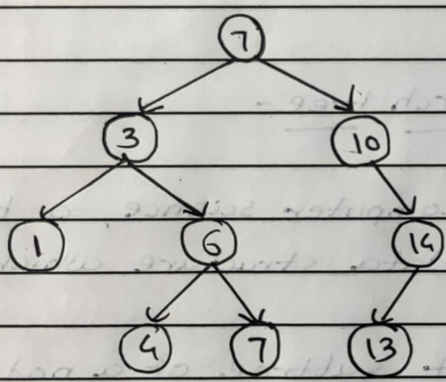
- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

from the above properties it naturally follows that :

- Each node (item in the tree) has a distinct key.

Generally, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records.

The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient.



**Operations** :

1. Insertion : Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an enternal node and add the value as its right or left chikld, depending on the node's value.

```
void InsertNode (Node * &treeNode, Node* newNode)
  {
    if (treeNode==NULL)
    treeNode = new Node;
        else if (new Node-->key < treeNode->key)
            InsertNode (treeNode->left, newNode);
        else
        InsertNode (treeNode->right, newNode);
    }
```

2. Searching:

We begin by enamining the root note. If the tree is null, the value we are searching for does not enist in the tree. Otherwise, if the equal value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search for the right subtree.

```
bool BinarySearchTree::Search (int val)
  {
    Node* nent = this->root();
    while (nent !=0)
    {
        if (val==nent->value())
        {
          return true;
        }
        else if (val < nent -> value() )
            nent = nent -> left();
        else if (val > nent -> value() )
            nent = nent -> right();
      }
      // not found
      return false;
    }
```

3. Decision: There are several cases tobe considered.

Deleting a leaf: Deleating a node with no children is easy, as we can simply remove it from the tree.

Deleting a node with one child: Delete it and replace it with its child.

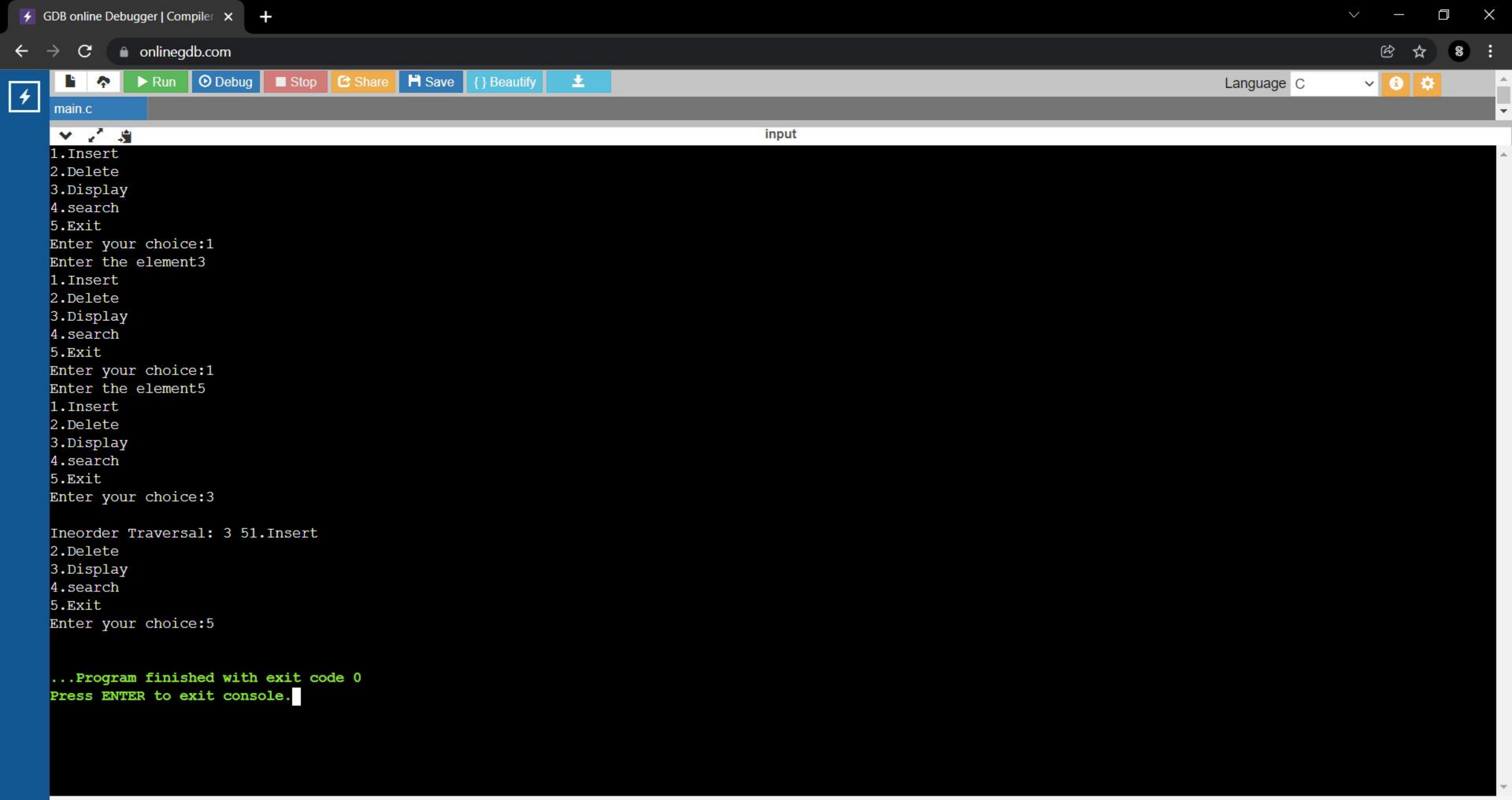
Deleting a node with two children: Call the node to be deleted "N". Do not delete N. Instead, chose either its in-order successor node "R". Replace the value of N with the value of R, then delete R. (Note: R itself have upto one child)

Conclusion: Thus we have implemented operations on BST.

Name-Shantanu Deshpande
Class- SYCSE
Roll No.-7
Batch-S1

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


struct node
{
int info;
struct node *left,*right;
}*root=NULL,*p,*q;

void insert(int);
void inorder(struct node *);
struct node * makenode(int a);
int search(int);
void delet(int);
struct node * parent(struct node *pr,struct node *n1,int no);

void main()
{
int ch =1;
while( ch != 5)
{
int ele,result;

printf("1.Insert \n2.Delete \n3.Display  \n4.search \n5.Exit \nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("Enter the element");
scanf("%d",&ele);
insert(ele);
break;
case 2:
printf("Enter the element to be deleted");
scanf("%d",&ele);
delet(ele);
break;
case 3:
printf("\nIneorder Traversal:");
inorder(root);

break;
case 4:
printf("\n Enter the element for search:");
scanf("%d",&ele);
result=search(ele);
if(result==1)
printf("Element is present");
else
printf("Element is not present");
break;
case 5:
break;
}
}
}
void insert(int x)
{
if(root==NULL)
root=makenode(x);
else
{
p=root;
while(p!=NULL)
{
q=p;
if(x<p->info)
p=q->left;
else
p=q->right;
}
if(x<=q->info)
q->left=makenode(x);
else
q->right=makenode(x);
}
}

struct node * makenode(int a)
{
struct node *s;
s=(struct node *)malloc(sizeof(struct node));
s->left=NULL;
s->right=NULL;
s->info=a;
return(s);
}

void inorder(struct node *t)
{
if(t==NULL)
return;
inorder(t->left);
printf(" %d",t->info);
inorder(t->right);
}


int search(int a)
{
if (root==NULL)
{
printf("Tree is empty");
return(0);
```

```
}
else
{
p=root;
while(p!=NULL)
{
 q=p;
 if(q->info==a)
  {
  return(1);
  }
 if(a<q->info)
  p=q->left;
 else
  p=q->right;

}
if(p==NULL)
return(0);
}
}

void delet(int x)
{
struct node *rp,*f,*s;
p=root;
q=NULL;
while(p!=NULL && p->info != x)
{
q=p;
p=(x<p->info)?p->left:p->right;
}
if(p==NULL)
{
printf("Element is not present");
return;
}
if(p->left==NULL)
    rp=p->right;
else
    if(p->right==NULL)
        rp=p->left;
    else
    {
        f=p;
        rp=p->right;
        s=rp->left;
     while(s!=NULL)
        {
         f=rp;
         rp=s;
         s=rp->left;
        }
        if(f!=p)
        {
            f->left=rp->right;
            rp->right=p->right;
        }
    rp->left=p->left;
    }

if(q==NULL)
    root=rp;
else
    if(p==q->left)
        q->left=rp;
    else q->right=rp;
        free(p);
}
```



```
1.Insert
2.Delete
3.Display
4.search
5.Exit
Enter your choice:1
Enter the element3
1.Insert
2.Delete
3.Display
4.search
5.Exit
Enter your choice:1
Enter the element5
1.Insert
2.Delete
3.Display
4.search
5.Exit
Enter your choice:3

Ineorder Traversal: 3 51.Insert
2.Delete
3.Display
4.search
5.Exit
Enter your choice:5


...Program finished with exit code 0
Press ENTER to exit console.
```