NAME: Venkateswar L
BRANCH: Computer Science and Engineering SEC: F
ROLL NO.: 230701376

PROGRAM: Implementation Of Binary Search Tree

Write a C program to implement a Binary Search Tree and perform the following operations.

1. Insert
2. Delete
3. Search
4. Display

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

struct tree
{
int data;
struct tree *left;
struct tree *right;
}*root=NULL;

void insert();
void display();
void search();
void delete();
struct tree * inorder_succ(struct tree *);
```

NAME: Venkateswar L
BRANCH: Computer Science and Engineering SEC: F
ROLL NO.: 230701376

```
void insert()
{
   while (1)
   {
      struct tree *parent,*ptr=root;
      int value;
      int flag=0;
      printf("Enter the value to be inserted\n");
      scanf("%d",&value);

      while(ptr!=NULL && flag==0)
      {
         if(value<ptr->data)
         {
            parent=ptr;
            ptr=ptr->left;
         }
         else if(value>ptr->data)
         {
            parent=ptr;
            ptr=ptr->right;
         }
         else if(value==ptr->data)
```

```
            {
                printf("No duplicate value");
                flag=1;
            }
        }
        struct tree *newnode;
        newnode=malloc(sizeof(struct tree));
        newnode->data=value;

        if(parent==NULL)
        {
            root=newnode;
        }
        else if(value<parent->data)
        {
            parent->left=newnode;
        }
        else
        {
            parent->right=newnode;
        }
        printf("Insert more elements? 1/0: ");
        int k;
        scanf("%d",&k);

        if (k==1)

        continue;

        else

        break;

    }

}




void display(struct tree *ptr)

{

    if(ptr!=NULL)

    {

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

        display(ptr->left);

        display(ptr->right);

    }

}


void search()

{

    int flag=0;

    int value;
```

```
struct tree *parent,*ptr=root;
printf("Enter the Value to be searched\n");
scanf("%d",&value);

while(ptr!=NULL && flag==0)
{
    if(value<ptr->data)
    {
        parent=ptr;
        ptr=ptr->left;
    }
    else if(value>ptr->data)
    {
        parent=ptr;
        ptr=ptr->right;
    }
    else if(value==ptr->data)
    {
        flag=1;
        printf("%d is found",ptr->data);
    }
}
if(flag==0)
```

```
    printf("Value not found");

}




void delete(struct tree *ptr,int key)

{

    struct tree *parent=NULL;

    int flag=0;

    while(ptr!=NULL && flag==0)

    {

        if(key<ptr->data)

        {

            parent=ptr;

            ptr=ptr->left;

        }

        else if(key>ptr->data)

        {

            parent=ptr;

            ptr=ptr->right;

        }

        else if(key==ptr->data)

        {

        flag=1;
```

```
      }

   }

   if(flag==0)

   printf("Required Key does not exist");

   else

   {

      if(ptr->left==NULL && ptr->right==NULL )

      {

         if(parent==NULL)

         {

            root=NULL;

         }

         else if (key<parent->data)

            parent->left =NULL;

         else

            parent->right=NULL;

         free(ptr);

      }

      else if(ptr->left==NULL || ptr->right==NULL )

      {

         if(parent==NULL)

         {

            if(ptr->right==NULL)
```

```
                   root=ptr->left;

               else

                   root=ptr->right;

            }

        }

        else if(key<parent->data)

            {

               if (ptr->left!=NULL)

                   parent->left=ptr->left;

               else

                   parent->left=ptr->right;

            }

         else if(key>parent->data)

            {

                if (ptr->left!=NULL)

                   parent->right=ptr->left;

               else

                   parent->right=ptr->right;

            }

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

         {

            struct tree*new_ptr;

            new_ptr=inorder_succ(ptr->right);
            int save=new_ptr->data;

            delete(ptr,new_ptr->data);

            ptr->data=save;

        }

    }

}


struct tree *inorder_succ(struct tree *pt)

{

   while(pt->left!=NULL)

   {


      pt=pt->left;

   }

   return pt;

}


int main()

{

   int key;

   struct tree *ptr=NULL;

   int choice;

   while(1)
```

```
  {

    printf("Enter your choice:-\n1.Insert\n2.Delete\n3.Display\n4.Search\n");

    scanf("%d",&choice);

    switch(choice)

    {

      case 1:

      insert();

      break;


      case 2:

      printf("\nEnter the value to be deleted\n");

      scanf("%d",&key);

      ptr=root;

      delete(ptr,key);

      break;


      case 3:

      ptr=root;

      display(ptr);

      break;


      case 4:

      search();
```

```
            break;

        }

        printf("\nWant to continue? 1/0 ");

        int m;

        scanf("%d",&m);

        if (m==1)

        continue;

        else

        break;

    }

}
```