//Implementation of binary search tree

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left,*right;
};
struct node *root=NULL;
void insert()
{
struct node *cur,*parent=NULL;
int n;
struct node *temp = (struct node*)malloc(sizeof(struct node));
printf("enter a number");
scanf("%d",&n);
temp->data=n;
temp->left=NULL;
temp->right=NULL;
if(root==NULL)
{
root=temp;
}
else
{
cur=root;
while(cur)
{
parent=cur;
if(temp->data>cur->data)
{
cur=cur->right;
}
else
{
cur=cur->left;
}
}
if(temp->data>parent->data)
{
parent->right=temp;
}
else
{
parent->left=temp;
}
}
}

void delet()
{
struct node *cur=root,*parent=NULL;
int n;
printf("enter a node data which u want to delete");
```

```
scanf("%d",&n);
if(root==NULL)
{
          printf("Tree is empty");
          return;
}
else
{
while(cur!=NULL)
{
          if(cur->data==n)
          {
                    break;
          }
          else
          {
                    parent=cur;
                    if(n>cur->data)
                              {
                              cur=cur->right;
                              }
                    else
                              {
                              cur=cur->left;
                              }
          }
}
if(cur==NULL)
{
          printf("Invalid data node.Try again");
          return;
}
}
//Leaf Node
if( cur->left == NULL && cur->right == NULL)
   {
      if(parent->left == cur)
      {
      parent->left = NULL;
      }
      else
      {
      parent->right = NULL;
                    }
          return;
          }
//Node with single child

if((cur->left == NULL && cur->right != NULL)|| (cur->left != NULL&& cur->right == NULL))
 {
     if(cur->left == NULL && cur->right != NULL)
     {
        if(parent->left == cur)
        {
          parent->left = cur->right;
```

```
            }
           else
           {
             parent->right = cur->right;

           }
       }
       else // left child present, no right child
       {
          if(parent->left == cur)
           {
             parent->left = cur->left;

           }
           else
           {
             parent->right = cur->left;

           }
       }
     return;
}

//Nodes have 2 child nodes
if (cur->left != NULL && cur->right != NULL)
{
             struct node *t1,*t2;
             t1=cur->right;
             if(t1->left==NULL && t1->right==NULL)
             {
             cur->data=t1->data;
             cur->right=NULL;
              //delete t1;
             }
             else
             {
                          if((cur->right)->left != NULL)
                              {
                                 struct node *rcur;
                                 struct node *rcurp;
                                 rcurp = cur->right;
                                 rcur = (cur->right)->left;
                                 while(rcur->left != NULL)
                                 {
                                    rcurp = rcur;
                                    rcur = rcur->left;
                                 }
                                      cur->data = rcur->data;
                                 //delete rcur;
                                 rcurp->left = NULL;
                             }
                             else
                             {
                                struct node *tmp;
                                tmp = cur->right;
                                cur->data = tmp->data;
                                cur->right = tmp->right;
                                //delete tmp;
                            }

            }
            return;
}
}

void search(struct node *root,int key)
{
if(root==NULL)
{
printf("Tree is empty/Element not found");
}
else if(root->data==key)
{
printf("element is found");
}
else if(root->data<key)
{
search(root->right,key);
}
else
{
search(root->left,key);
}
}
void preorder(struct node *t)
{
if(t != NULL)
   {
      printf("%d ",t->data);
      if(t->left) preorder(t->left);
      if(t->right) preorder(t->right);
   }
   else return;
}

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

```
        if(t->right) postorder(t->right);
        printf("%d ",t->data);

    }
    else return;
}

int main()
{
int ch,n;
while(1)
    {
      printf("\n");
      printf(" Binary Search Tree Operations\n ");
      printf(" ---------------------------- ");
      printf(" \n1. Insertion/Creation ");
      printf(" \n2. Pre-Order Traversal ");
      printf(" \n3. In-Order Traversal ");
      printf(" \n4. Post-Order Traversal ");
      printf(" \n5. Delete ");
      printf(" \n6. Search ");
      printf(" \n7. Exit ");
      printf(" \nEnter your choice : ");
      scanf("%d",&ch);
       switch(ch)
      {
         case 1 : insert();
                     break;
         case 2 : preorder(root);
                     break;
         case 3 : inorder(root);
                     break;
         case 4 : postorder(root);
                     break;
         case 5 : delet();
                     break;
         case 6: printf("enter an element to be search");
                             scanf("%d",&n);
                             search(root,n);
                             break;
         case 7: return 0;
         default: printf("Selct valid option");
                     break;
        }
      }
 }
```