```
/*
Subject: DSA lab
Practical No: 04
Title: A C++ Program to implement Binary Search Tree and it's Operations
        a) Create BST
        b) Insert New Nodes
        c) Display - i. Preoreder, ii. Inorder, iii. Postorder
        d) Search a Key
        e) Delete a Node
        f) Find Max and Min Value in BST
*/

            //.........Header Files
#include <iostream>
using namespace std;

int level = 0;

struct Node
{
   int data;
   struct Node *left;
   struct Node *right;
}*Root, *Parent;

            //.........Function to Create Root of BST
void create_BST(int val)
{
   struct Node *Newnode;

   Newnode = new struct Node;

   Newnode->data = val;
   Newnode->left = NULL;
   Newnode->right = NULL;

   if(Root == NULL)
   {
      Root = Newnode;
      cout<<"\n\t Root "<<Root->data<<" of BST is Ready Now...!!!";
   }
   else
      cout<<"\n\t Root already exists....!!!";
}

            //.........Function to display BST in Preorder_BST(Data->Left->Right)
void Preorder_BST(struct Node *root)
{
   if(root != NULL)
   {
      cout<<" "<<root->data;
      Preorder_BST(root->left);
```

```
        Preorder_BST(root->right);
    }
}

          //.........Function to display BST in Inorder_BST(Left->Data->Right)
void Inorder_BST(struct Node *root)
{
    if(root != NULL)
    {
        Inorder_BST(root->left);
        cout<<" "<<root->data;
        Inorder_BST(root->right);
    }
}

          //.........Function to display BST in Postorder_BST(Left->Right->Data)
void Postorder_BST(struct Node *root)
{
    if(root != NULL)
    {
        Postorder_BST(root->left);
        Postorder_BST(root->right);
        cout<<" "<<root->data;
    }
}

              //.....Function to create New Node
struct Node* create_Node(int val)
{
    struct Node *Newnode;

    Newnode = new struct Node;

    Newnode->data = val;
    Newnode->left = NULL;
    Newnode->right = NULL;

    return Newnode;
}

                  //.....Function to Insert New Nodes in BST
void insert_BST(struct Node *current, struct Node *Newnode)
{
    if(current != NULL)
    {
        if(Newnode->data <= current->data)
        {
            if(current->left == NULL)
            {
                current->left = Newnode;
                cout<<"\n\nNewnode "<<Newnode->data<<" is inserted as Left Child of "<<current-
>data;
```

```
            }
            else
               insert_BST(current->left, Newnode);
         }
         else
         {
            if(current->right == NULL)
            {
               current->right = Newnode;
               cout<<"\n\nNewnode "<<Newnode->data<<" is inserted as Right Child of "<<current-
>data;
            }
            else
               insert_BST(current->right, Newnode);
         }
      }
   }


void search_Node(struct Node *root, int Key)
{
   if(root != NULL)
   {
      if(Key == root->data)
         cout<<"\n\t Key is found on level "<<level;
      else if(Key < root->data)
      {
         level++;
         search_Node(root->left, Key);
      }
      else
      {
         level++;
         search_Node(root->right, Key);
      }
   }
   else
      cout<<"\n\t Key not found...!!!";
}


void Min_Node()
{
   struct Node *p;

   p = Root;
   while(p->left != NULL)
     p = p->left;

   cout<<"\n\n Min Node/Value: "<<p->data;
}
```

```
void Max_Node()
{
   struct Node *p;

   p = Root;
   while(p->right != NULL)
     p = p->right;

   cout<<"\n\n Max Node/Value: "<<p->data;
}

struct Node* minValueNode(struct Node* node)
{
   struct Node* current = node;

   /* loop down to find the leftmost leaf */
   while (current && current->left != NULL)
      current = current->left;

   return current;
}

struct Node* deleteNode(struct Node *root, int Key)
{
   // base case
   if (root == NULL)
      return root;

   // If the key to be deleted is
   // smaller than the root's
   // key, then it lies in left subtree
   if (Key < root->data)
      root->left = deleteNode(root->left, Key);

   // If the key to be deleted is
   // greater than the root's
   // key, then it lies in right subtree
   else if (Key > root->data)
      root->right = deleteNode(root->right, Key);

   // if key is same as root's key, then This is the node
   // to be deleted
   else
   {
      // node has no child
      if (root->left==NULL and root->right==NULL)
         return NULL;

      // node with only one child or no child
      else if (root->left == NULL) {
         struct Node* temp = root->right;
         delete root;
```

```
            return temp;
        }
        else if (root->right == NULL) {
            struct Node* temp = root->left;
            delete root;
            return temp;
        }

        // node with two children: Get the inorder successor
        // (smallest in the right subtree)
        struct Node* temp = minValueNode(root->right);

        // Copy the inorder successor's content to this node
        root->data = temp->data;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

            //.........Main Function
int main()
{
    cout<<"\n\n ..... Binary Search Tree and it's Operations .....";

    struct Node *Newnode;
    int Key;

    Root = NULL;

    cout<<"\n\n 1. Creating BST .....";

    create_BST(25);

    cout<<"\n\n 2. Inserting New Nodes in BST .....";

    Newnode = create_Node(10);

    insert_BST(Root, Newnode);

    Newnode = create_Node(40);

    insert_BST(Root, Newnode);

    Newnode = create_Node(5);

    insert_BST(Root, Newnode);

    Newnode = create_Node(45);

    insert_BST(Root, Newnode);
```

```
    cout<<"\n\n 3. Display/Traversing BST .....";

    cout<<"\n\n Preorder Traversal: ";
    Preorder_BST(Root);

    cout<<"\n\n Inorder Traversal: ";
    Inorder_BST(Root);

    cout<<"\n\n Postorder Traversal: ";
    Postorder_BST(Root);

    cout<<"\n\n 4. Search a Key in BST .....";
    cout<<"\n\t Enter the Key to Search: ";
    cin>>Key;
    search_Node(Root, Key);

    cout<<"\n\n 5. Find MIN/MAX Node in BST .....";

    Min_Node();

    Max_Node();

    cout<<"\n\n 6. Delete a Key from BST .....";

    level = 0;
    Root = deleteNode(Root, 5);
    cout<<"\n\n After deleting 5 Inorder Traversal: ";
    Inorder_BST(Root);

    Root = deleteNode(Root, 40);
    cout<<"\n\n After deleting 40 Inorder Traversal: ";
    Inorder_BST(Root);

    Root = deleteNode(Root, 25);
    cout<<"\n\n After deleting 25 Inorder Traversal: ";
    Inorder_BST(Root);

    return 0;
}

/*

..... Binary Search Tree and it's Operations .....

 1. Creating BST .....
        Root 25 of BST is Ready Now...!!!

 2. Inserting New Nodes in BST .....

Newnode 10 is inserted as Left Child of 25
```

Newnode 40 is inserted as Right Child of 25

Newnode 5 is inserted as Left Child of 10

Newnode 45 is inserted as Right Child of 40

3. Display/Traversing BST .....

Preorder Traversal: 25 10 5 40 45

Inorder Traversal: 5 10 25 40 45

Postorder Traversal: 5 10 45 40 25

4. Search a Key in BST .....
Enter the Key to Search: 45

Key is found on level 2

5. Find MIN/MAX Node in BST .....

Min Node/Value: 5

Max Node/Value: 45

6. Delete a Key from BST .....

After deleting 5 Inorder Traversal: 10 25 40 45

After deleting 40 Inorder Traversal: 10 25 45

After deleting 25 Inorder Traversal: 10 45

...Program finished with exit code 0
Press ENTER to exit console.
*/