Name: M.Rohith
3122 21 5001 085

# SSN College of Engineering

## Department of Computer Science and Engineering

## UCS2312 – Data Structures Lab

## II Year CSE - B Section ( III Semester)

## Academic Year 2022-23

**Staff Incharge: Dr.H. Shahul Hamead**

**Exercise-6: Exercises on Binary Search Trees**

**Aim:**

To implement C program in Data structures using the concept of Binary Search Trees.

Ex-no: 6 Name: M.Rohith
Date: 7-12-2022 3122 21 5001 085

**Pseudocode:**

6. Exercises on Binary Search Tree.

**Basic:**

⇒ Perform creation of a BST with 'n' no. of nodes and implement 3 traversals.
⇒ Perform searching of a node.
⇒ Print inorder successor & predecessor of node.
⇒ Deletion of a node from BST has 1 child.
⇒ Deletion of a leaf node from BST
⇒ Measure the height of a BST

**Advanced:**

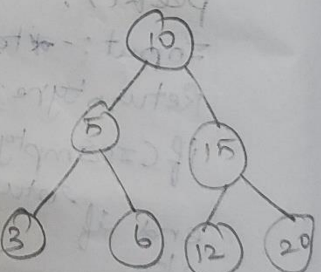
⇒ Deletion of a leaf node from BST

**Application**

⇒ Measure the height of a BST.

Algorithm:-

```
struct node
{
    int data;
    struct node* right_child;
    struct node *left_child;
};
```



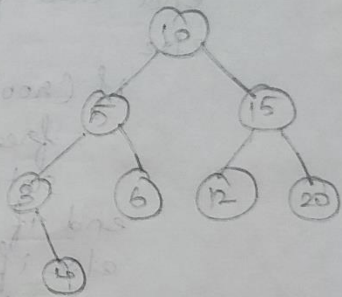

```
New node (),
Input: int x
Return: struct node * temp
temp = malloc(sizeof(struct (node));
temp → data = x
temp → left_child = NULL
temp → right_child = NULL
return temp
```

Name: M.Rohith
3122 21 5001 085

search ( ).

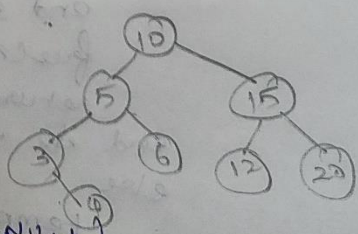
```
Input : struct node * root , int x
Return : struct node *
if(root== NULL || root -> data ==x)
    return root
end if
else if (root -> data < x)
    return search (root -> right_child, x)
end else if
else
    return search (root -> left_child, x)
end else
```

INSERT (4)



Insert ( )

```
Input: struct node * root, int x
Return type : struct node *
if (root ==NULL)
    return new_node (x)
else if (x > root -> data)
    root -> right_child = insert (root -> right_child, x)
end else if
else
    root -> left_child = insert (root -> left_child, x)
end else
return root
```

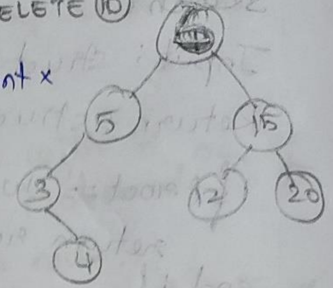


find_minimum ( )

```
if (root== NULL)
    return NULL
else if (root -> left_child != NULL)
    return find_minimum (root -> left_child)
return root
```

Name: M.Rohith
3122 21 5001 085

DELETE 10



delete()

Input: struct node * root, int x

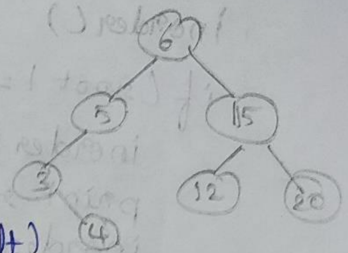
Return type: struct node *

```
if (root == NULL)
    return NULL
end if
else if (x > root -> data)
    root -> right_child = delete(root -> right_child, x)
end else if
else if (x < root -> data)
    root -> left_child = delete(root -> left_child, x)
end else if
else
    if (root -> left == NULL && root -> right == NULL)
        free(root)
        return NULL
    end if
    else if (root -> left_child == NULL || root -> right == NULL)
        if (root -> left == NULL)
            temp = root -> right
        end if
        else
            temp = root -> left
        end else
        free(root)
        return temp
    end else if
    else
        temp = find_minimum(root -> right)
        root -> data = temp -> data
        root -> right = delete(root -> right, temp -> data)
    end else
end else
return root
```

Ex-no: 6
Date: 7-12-2022

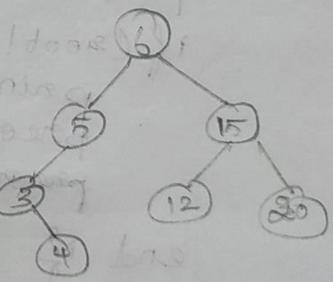
Name: M.Rohith
3122 21 5001 085

```
maxDepth( )
if (root==NULL)
    return 0
end if
else
    ldepth=maxDepth(root -> left)
    rdepth= maxDepth(root -> right)
    if ( ldepth>rdepth)
        return ldepth+1
    end if
    else
        return rdepth+1
    end else
end else .
```



maxDepth = 4

```
searchelement ( )
while(root != NULL)
    if (value > root-> data)
        root=root->right
    end if
    else if (value < root->data)
        root = root->left
    end else if
    else
        return 1
    end else
end while
return 0
```

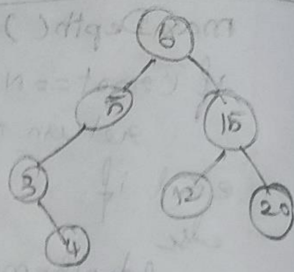


5 : found

Ex-no: 6
Date: 7-12-2022

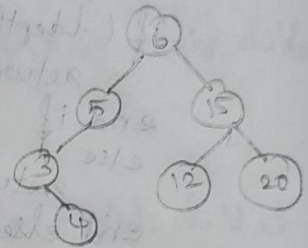
Name: M.Rohith
3122 21 5001 085

```
inorder()
 if(root != NULL)
    inorder(root->left)
    print root->data
    inorder(root->right)
end if
```



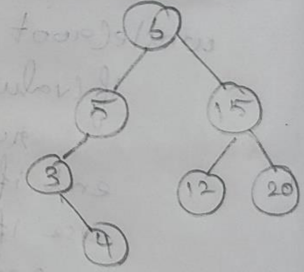
3, 4 5 6 12 15 20

```
postorder()
if (root==0)
    return
end if
postorder (root->left)
postorder (root->right)
print "root->data"
```



4 3 5 12 20 15 6

```
preorder ()
if(root! =NULL)
    print root->data
    preorder (root-> left)
    preorder (root -> right)
end if
```



6 5 3 4 15 12 20

```
main()
newnode(5)
INPUT x
insert(x)
INPUT val
search_element(val)
inorder()
preorder()
postorder()
delete()
height()
```

Ex-no: 6 Name: M.Rohith
Date: 7-12-2022 3122 21 5001 085

**Basic:**

- **Perform Creation of a BST with 'n' number of nodes and implement 3 traversals**
- **Perform searching of a node**
- **Print inorder successor and predecessor of a node**
- **Deletion of a node from BST has 1 child**
- **Deletion of a leaf node from BST**

**Program code:**

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

struct node
{
  int data;                          //node will store some data
  struct node *right_child;         // right child
  struct node *left_child;         // left child
};

//function to create a node
struct node* new_node(int x)
{
  struct node *temp;
  temp=malloc(sizeof(struct node));
  temp->data=x;
  temp->left_child=NULL;
  temp->right_child=NULL;
  return temp;
}

// searching operation
struct node* search(struct node * root, int x)
{
  if(root==NULL||root->data==x)      //if root->data is x then the
element is found
    return root;
```

Name: M.Rohith
3122 21 5001 085

```
  else if (x>root->data)              // x is greater, so we will
search the right subtree
    return search(root->right_child,x);
  else                                //x is smaller than the data,
so we will search the left subtree
    return search(root->left_child,x);
}

// A sample C function to check if a given node exists in a binary
search tree or not
int searchelement(struct node* root, int value)
{
    // while is used to traverse till the end of tree
    while (root != NULL){

        // checking condition and passing right subtree & recusing
        if (value > root->data)
            root = root->right_child;

        // checking condition and passing left subtree & recusing
        else if (value < root->data)
            root = root->left_child;
        else
            return 1; // if the value is found return 1
    }
    return 0;
}

// insertion
struct node* insert(struct node *root,int x)
{
  //searching for the place to insert
  if(root==NULL)
    return new_node(x);
  else if(x>root->data)            // x is greater. Should be
inserted to the right
    root->right_child=insert(root->right_child,x);
  else                             // x is smaller and should be
inserted to left
    root->left_child=insert(root->left_child,x);
  return root;
}
```

Name: M.Rohith
3122 21 5001 085

```
//function to find the minimum value in a node
struct node* find_minimum(struct node *root)
{
  if (root==NULL)
    return NULL;
  else if (root->left_child!=NULL)          // node with minimum
value will have no left child
    return find_minimum(root->left_child); // left most element will
be minimum
  return root;
}

struct node* find_maximum(struct node *root)
{
  if(root==NULL)
    return NULL;
  else if(root->right_child!=NULL)
    return find_maximum(root->right_child);
  return root;
}

// deletion
struct node* delete(struct node *root,int x)
{
  //searching for the item to be deleted
  if(root==NULL)
    return NULL;
  if(x>root->data)
    root->right_child=delete(root->right_child,x);
  else if(x<root->data)
    root->left_child=delete(root->left_child,x);
  else
  {
    //No Child node
    if(root->left_child==NULL && root->right_child==NULL)
    {
      free(root);
      return NULL;
    }

    //One Child node
```

Name: M.Rohith
3122 21 5001 085

```
    else if(root->left_child==NULL || root->right_child==NULL)
    {
      struct node *temp;
      if(root->left_child==NULL)
        temp=root->right_child;
      else
        temp=root->left_child;
      free(root);
      return temp;
    }

    //Two Children
    else
    {
      struct node *temp=find_minimum(root->right_child);
      root->data=temp->data;
      root->right_child=delete(root->right_child,temp->data);
    }
  }
  return root;
}

// Inorder Traversal
void inorder(struct node *root)
{
  if (root != NULL) // checking if the root is not null
  {
    inorder(root->left_child); // traversing left child
    printf("%d ",root->data); // printing data at root
    inorder(root->right_child); // traversing right child
  }
}

void postorder(struct node *root)
{
    if(root==0)
        return;
    postorder(root->left_child);
    postorder(root->right_child);
    printf("%d ",root->data);
}
```

Name: M.Rohith
3122 21 5001 085

```
void preorder(struct node *root)
{
    if(root==0)
        return;
    printf("%d ",root->data);
    preorder(root->left_child);
    preorder(root->right_child);
}

int main()
{
  struct node
*root=NULL,*inorder_successor=NULL,*inorder_predecessor=NULL;
  int choice=0,x=0,height=0,n=0;
  do
  {
    printf("\n----------------Menu---------------\n");
    printf("\n1.Creation of a BST with ‘n’ number of nodes and
implement 3 traversals \n2.Searching of a node\n3.Printing the
inorder successor and inorder predecessor\n4.Delete of a node from
BST has 1 child\n5.Deletion of a leaf node from BST\n6.Exit\n");
    printf("\nEnter the choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("\nEnter the number of nodes you want to create: ");
        scanf("%d",&n);
        if(n==0)
          printf("\nInvalid number\n");
        else if(n==1)
        {
          printf("\nEnter the value of root node: ");
          scanf("%d",&x);
          root=new_node(x);
        }
        else
        {
          while(n>0)
          {
            printf("\nEnter the value to insert: ");
            scanf("%d",&x);
```

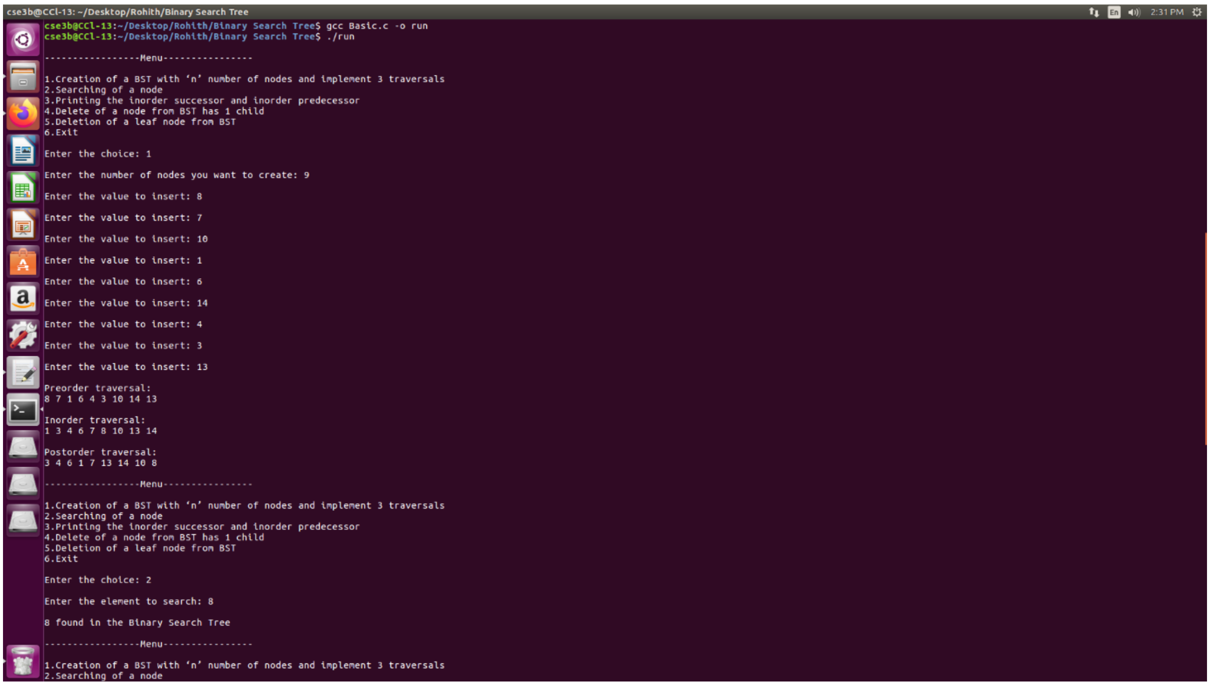
Name: M.Rohith
3122 21 5001 085

```
            root=insert(root,x);
            n--;
           }
        }
        printf("\nPreorder traversal:\n");
        preorder(root);
        printf("\n");
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        printf("\nPostorder traversal:\n");
        postorder(root);
        printf("\n");
        break;
      case 2:
  printf("\nEnter the element to search: ");
  scanf("%d",&x);
  n=searchelement(root,x);
  if(n==1)
    printf("\n%d found in the Binary Search Tree\n",x);
  else
    printf("\n%d found in the Binary Search Tree\n",x);
        break;
      case 3:
  inorder_predecessor=find_maximum(root->left_child);
  inorder_successor=find_minimum(root->right_child);
  printf("\nInorder successor = %d",inorder_successor->data);
  printf("\nInorder predecessor = %d\n",inorder_predecessor->data);
        break;
      case 4:
        printf("\nEnter the value to delete: ");
        scanf("%d",&x);
        root=delete(root,x);
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 5:
        printf("\nPreorder traversal:\n");
        printf("\nEnter the value to delete: ");
        scanf("%d",&x);
        root=delete(root,x);
```

Name: M.Rohith
3122 21 5001 085

```
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 6:
        exit(0);
      default:
        printf("\nInvalid choice\n");
    }
  } while (choice!=6);
  return 0;
}
```

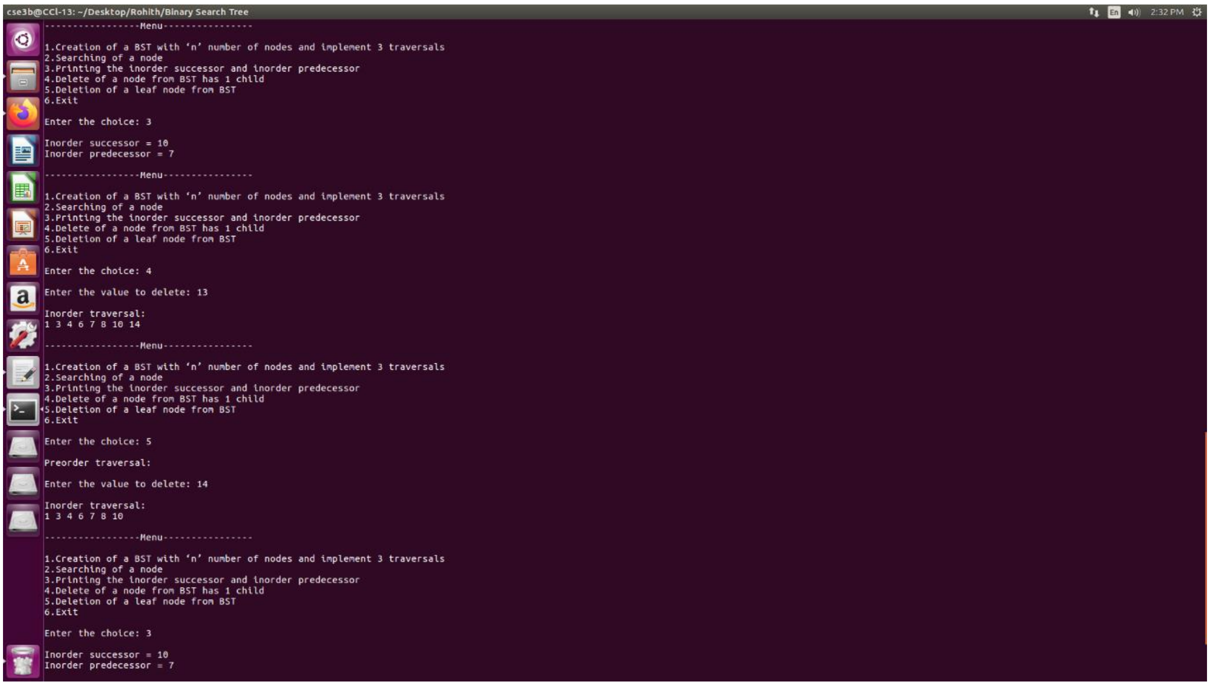
## <u>**Output:**</u>



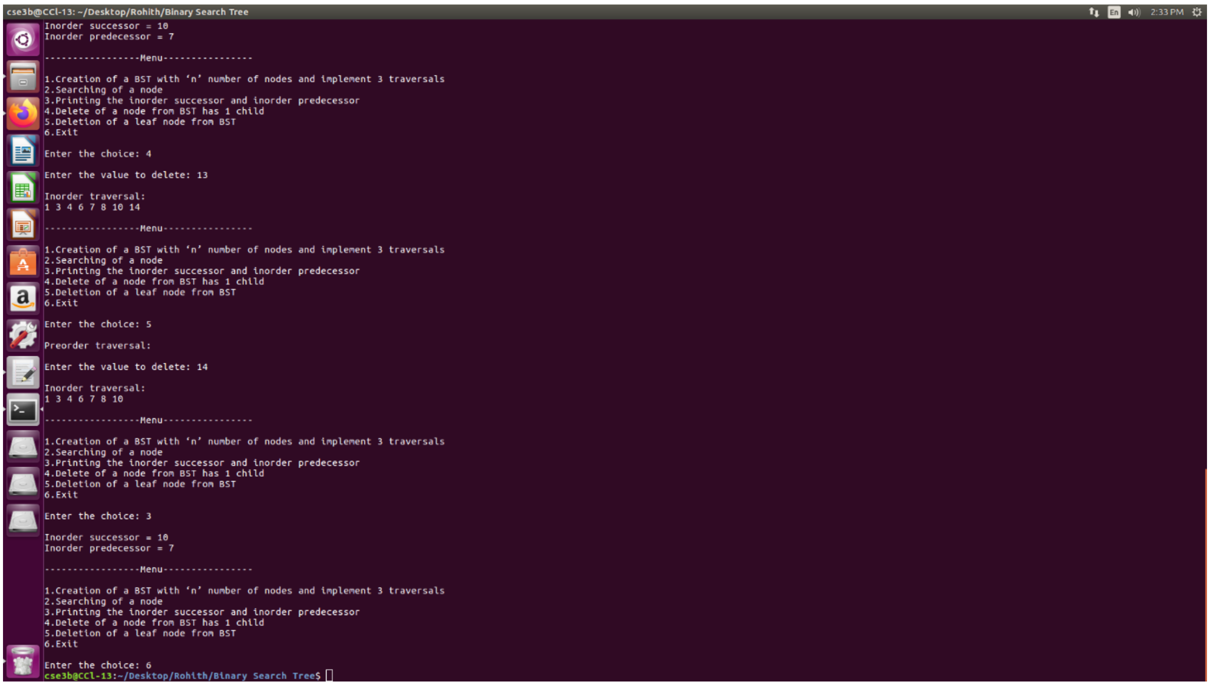
```
cse3b@CCl-13: ~/Desktop/Rohith/Binary Search Tree
cse3b@CCl-13:~/Desktop/Rohith/Binary Search Tree$ gcc Basic.c -o run
cse3b@CCl-13:~/Desktop/Rohith/Binary Search Tree$ ./run

----------------Menu---------------

1.Creation of a BST with ‘n’ number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 1

Enter the number of nodes you want to create: 9

Enter the value to insert: 8

Enter the value to insert: 7

Enter the value to insert: 10

Enter the value to insert: 1

Enter the value to insert: 6

Enter the value to insert: 14

Enter the value to insert: 4

Enter the value to insert: 3

Enter the value to insert: 13

Preorder traversal:
8 7 1 6 4 3 10 14 13

Inorder traversal:
1 3 4 6 7 8 10 13 14

Postorder traversal:
3 4 6 1 7 13 14 10 8

----------------Menu---------------

1.Creation of a BST with ‘n’ number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 2

Enter the element to search: 8

8 found in the Binary Search Tree

----------------Menu---------------

1.Creation of a BST with ‘n’ number of nodes and implement 3 traversals
2.Searching of a node
```

Ex-no: 6
Date: 7-12-2022

Name: M.Rohith
3122 21 5001 085



```
cse3b@CCL-13: ~/Desktop/Rohith/Binary Search Tree
----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 3

Inorder successor = 10
Inorder predecessor = 7

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 4

Enter the value to delete: 13

Inorder traversal:
1 3 4 6 7 8 10 14

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 5

Preorder traversal:

Enter the value to delete: 14

Inorder traversal:
1 3 4 6 7 8 10

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 3

Inorder successor = 10
Inorder predecessor = 7
```



```
cse3b@CCL-13: ~/Desktop/Rohith/Binary Search Tree
Inorder successor = 10
Inorder predecessor = 7

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 4

Enter the value to delete: 13

Inorder traversal:
1 3 4 6 7 8 10 14

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 5

Preorder traversal:

Enter the value to delete: 14

Inorder traversal:
1 3 4 6 7 8 10

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 3

Inorder successor = 10
Inorder predecessor = 7

----------------Menu---------------

1.Creation of a BST with 'n' number of nodes and implement 3 traversals
2.Searching of a node
3.Printing the inorder successor and inorder predecessor
4.Delete of a node from BST has 1 child
5.Deletion of a leaf node from BST
6.Exit

Enter the choice: 6
cse3b@CCL-13:~/Desktop/Rohith/Binary Search Tree$
```

Ex-no: 6 Name: M.Rohith
Date: 7-12-2022 3122 21 5001 085

## Application - Measure the height of a BST

## Program code:

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

struct node
{
  int data;                          //node will store some data
  struct node *right_child;         // right child
  struct node *left_child;         // left child
};

//function to create a node
struct node* new_node(int x)
{
  struct node *temp;
  temp=malloc(sizeof(struct node));
  temp->data=x;
  temp->left_child=NULL;
  temp->right_child=NULL;
  return temp;
}

// searching operation
struct node* search(struct node * root, int x)
{
  if(root==NULL||root->data==x)      //if root->data is x then the
element is found
    return root;
  else if (x>root->data)             // x is greater, so we will
search the right subtree
    return search(root->right_child,x);
  else                               //x is smaller than the data,
so we will search the left subtree
    return search(root->left_child,x);
}

// insertion
struct node* insert(struct node *root,int x)
{
```

```
  //searching for the place to insert
  if(root==NULL)
    return new_node(x);
  else if(x>root->data)           // x is greater. Should be
inserted to the right
    root->right_child=insert(root->right_child,x);
  else                           // x is smaller and should be
inserted to left
    root->left_child=insert(root->left_child,x);
  return root;
}

//function to find the minimum value in a node
struct node* find_minimum(struct node *root)
{
  if (root==NULL)
    return NULL;
  else if (root->left_child!=NULL)          // node with minimum
value will have no left child
    return find_minimum(root->left_child); // left most element will
be minimum
  return root;
}

// deletion
struct node* delete(struct node *root,int x)
{
  //searching for the item to be deleted
  if(root==NULL)
    return NULL;
  if(x>root->data)
    root->right_child=delete(root->right_child,x);
  else if(x<root->data)
    root->left_child=delete(root->left_child,x);
  else
  {
    //No Child node
    if(root->left_child==NULL && root->right_child==NULL)
    {
      free(root);
      return NULL;
    }
```

Name: M.Rohith
3122 21 5001 085

```
    //One Child node
    else if(root->left_child==NULL || root->right_child==NULL)
    {
      struct node *temp;
      if(root->left_child==NULL)
        temp=root->right_child;
      else
        temp=root->left_child;
      free(root);
      return temp;
    }

    //Two Children
    else
    {
      struct node *temp=find_minimum(root->right_child);
      root->data=temp->data;
      root->right_child=delete(root->right_child,temp->data);
    }
  }
  return root;
}

/* Compute the "maxDepth" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
int maxDepth(struct node* root)
{
    if (root==NULL)
        return 0;
    else
    {
      /* compute the depth of each subtree */
      int lDepth=maxDepth(root->left_child);
      int rDepth=maxDepth(root->right_child);

      /* use the larger one */
      if (lDepth>rDepth)
        return (lDepth+1);
      else
        return (rDepth+1);
```

Name: M.Rohith
3122 21 5001 085

```
    }
}

// Inorder Traversal
void inorder(struct node *root)
{
  if (root != NULL) // checking if the root is not null
  {
    inorder(root->left_child); // traversing left child
    printf("%d ",root->data); // printing data at root
    inorder(root->right_child); // traversing right child
  }
}

void postorder(struct node *root)
{
    if(root==0)
        return;
    postorder(root->left_child);
    postorder(root->right_child);
    printf("%d ",root->data);
}

void preorder(struct node *root)
{
    if(root==0)
        return;
    printf("%d ",root->data);
    preorder(root->left_child);
    preorder(root->right_child);
}

int main()
{
  struct node *root=NULL;
  int choice=0,x=0,height=0,n=0;
  do
  {
    printf("\n----------------Menu----------------\n");
    printf("\n1.Create a root node\n2.Insert an element\n3.Delete an
element\n4.Inorder display\n5.Preorder display\n6.Postorder
display\n7.Height of Tree\n8.Exit\n");
```

Name: M.Rohith
3122 21 5001 085

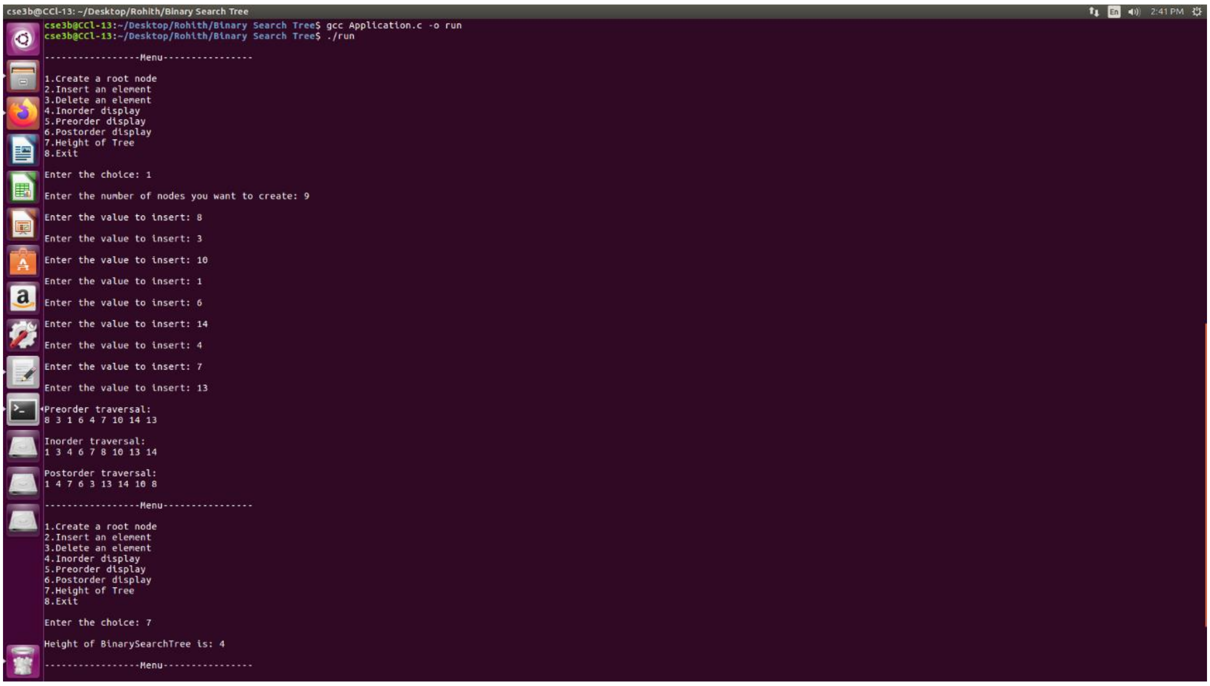
```
    printf("\nEnter the choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("\nEnter the number of nodes you want to create: ");
        scanf("%d",&n);
        if(n==0)
          printf("\nInvalid number\n");
        else if(n==1)
        {
          printf("\nEnter the value of root node: ");
          scanf("%d",&x);
          root=new_node(x);
        }
        else
        {
          while(n>0)
          {
            printf("\nEnter the value to insert: ");
            scanf("%d",&x);
            root=insert(root,x);
            n--;
           }
        }
        printf("\nPreorder traversal:\n");
        preorder(root);
        printf("\n");
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        printf("\nPostorder traversal:\n");
        postorder(root);
        printf("\n");
        break;
      case 2:
        printf("\nEnter the value to insert: ");
        scanf("%d",&x);
        root=insert(root,x);
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
```

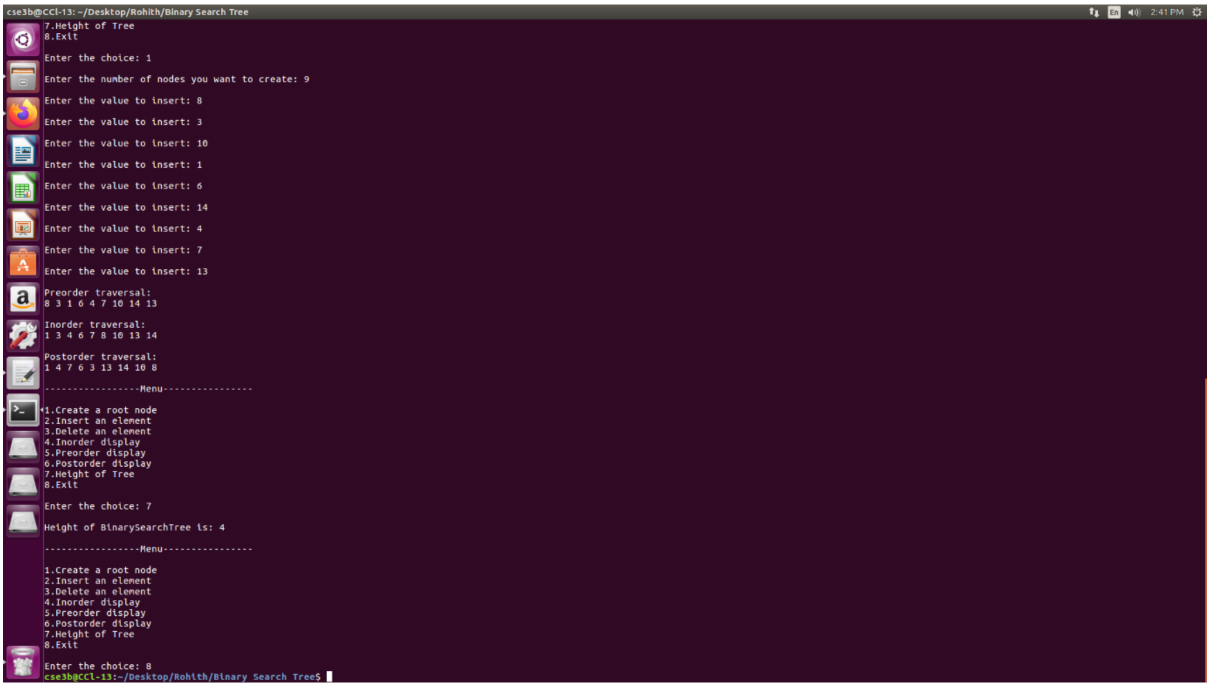
Name: M.Rohith
3122 21 5001 085

```
        break;
      case 3:
        printf("\nEnter the value to delete: ");
        scanf("%d",&x);
        root=delete(root,x);
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 4:
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 5:
        printf("\nPreorder traversal:\n");
        preorder(root);
        printf("\n");
        break;
      case 6:
        printf("\nPostorder traversal:\n");
        postorder(root);
        printf("\n");
        break;
      case 7:
        height=maxDepth(root);
        printf("\nHeight of BinarySearchTree is: %d\n",height);
        break;
      case 8:
        exit(0);
      default:
        printf("\nInvalid choice\n");
    }
  } while (choice!=8);
  return 0;
}
```

Name: M.Rohith
3122 21 5001 085


## **Output:**



Ex-no: 6 Name: M.Rohith
Date: 7-12-2022 3122 21 5001 085

## Advanced - Deletion of a node which has two children

## Program code:

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

struct node
{
  int data;                            //node will store some data
  struct node *right_child;          // right child
  struct node *left_child;         // left child
};

//function to create a node
struct node* new_node(int x)
{
  struct node *temp;
  temp=malloc(sizeof(struct node));
  temp->data=x;
  temp->left_child=NULL;
  temp->right_child=NULL;
  return temp;
}

// searching operation
struct node* search(struct node * root, int x)
{
  if(root==NULL||root->data==x)     //if root->data is x then the
element is found
    return root;
  else if (x>root->data)             // x is greater, so we will
search the right subtree
    return search(root->right_child,x);
  else                               //x is smaller than the data,
so we will search the left subtree
    return search(root->left_child,x);
}

// insertion
```

Name: M.Rohith
3122 21 5001 085

```
struct node* insert(struct node *root,int x)
{
  //searching for the place to insert
  if(root==NULL)
    return new_node(x);
  else if(x>root->data)           // x is greater. Should be
inserted to the right
    root->right_child=insert(root->right_child,x);
  else                            // x is smaller and should be
inserted to left
    root->left_child=insert(root->left_child,x);
  return root;
}

//function to find the minimum value in a node
struct node* find_minimum(struct node *root)
{
  if (root==NULL)
    return NULL;
  else if (root->left_child!=NULL)          // node with minimum
value will have no left child
    return find_minimum(root->left_child); // left most element will
be minimum
  return root;
}

// deletion
struct node* delete(struct node *root,int x)
{
  //searching for the item to be deleted
  if(root==NULL)
    return NULL;
  if(x>root->data)
    root->right_child=delete(root->right_child,x);
  else if(x<root->data)
    root->left_child=delete(root->left_child,x);
  else
  {
    //No Child node
    if(root->left_child==NULL && root->right_child==NULL)
    {
      free(root);
```

Name: M.Rohith
3122 21 5001 085

```
      return NULL;
    }

    //One Child node
    else if(root->left_child==NULL || root->right_child==NULL)
    {
      struct node *temp;
      if(root->left_child==NULL)
        temp=root->right_child;
      else
        temp=root->left_child;
      free(root);
      return temp;
    }

    //Two Children
    else
    {
      struct node *temp=find_minimum(root->right_child);
      root->data=temp->data;
      root->right_child=delete(root->right_child,temp->data);
    }
  }
  return root;
}

/* Compute the "maxDepth" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
int maxDepth(struct node* root)
{
    if (root==NULL)
        return 0;
    else
    {
      /* compute the depth of each subtree */
      int lDepth=maxDepth(root->left_child);
      int rDepth=maxDepth(root->right_child);

      /* use the larger one */
      if (lDepth>rDepth)
        return (lDepth+1);
```

Name: M.Rohith
3122 21 5001 085

```
      else
        return (rDepth+1);
    }
}

// Inorder Traversal
void inorder(struct node *root)
{
  if (root != NULL) // checking if the root is not null
  {
    inorder(root->left_child); // traversing left child
    printf("%d ",root->data); // printing data at root
    inorder(root->right_child); // traversing right child
  }
}

void postorder(struct node *root)
{
    if(root==0)
        return;
    postorder(root->left_child);
    postorder(root->right_child);
    printf("%d ",root->data);
}

void preorder(struct node *root)
{
    if(root==0)
        return;
    printf("%d ",root->data);
    preorder(root->left_child);
    preorder(root->right_child);
}

int main()
{
  struct node *root=NULL;
  int choice=0,x=0,height=0,n=0;
  do
  {
    printf("\n----------------Menu----------------\n");
```

Name: M.Rohith
3122 21 5001 085

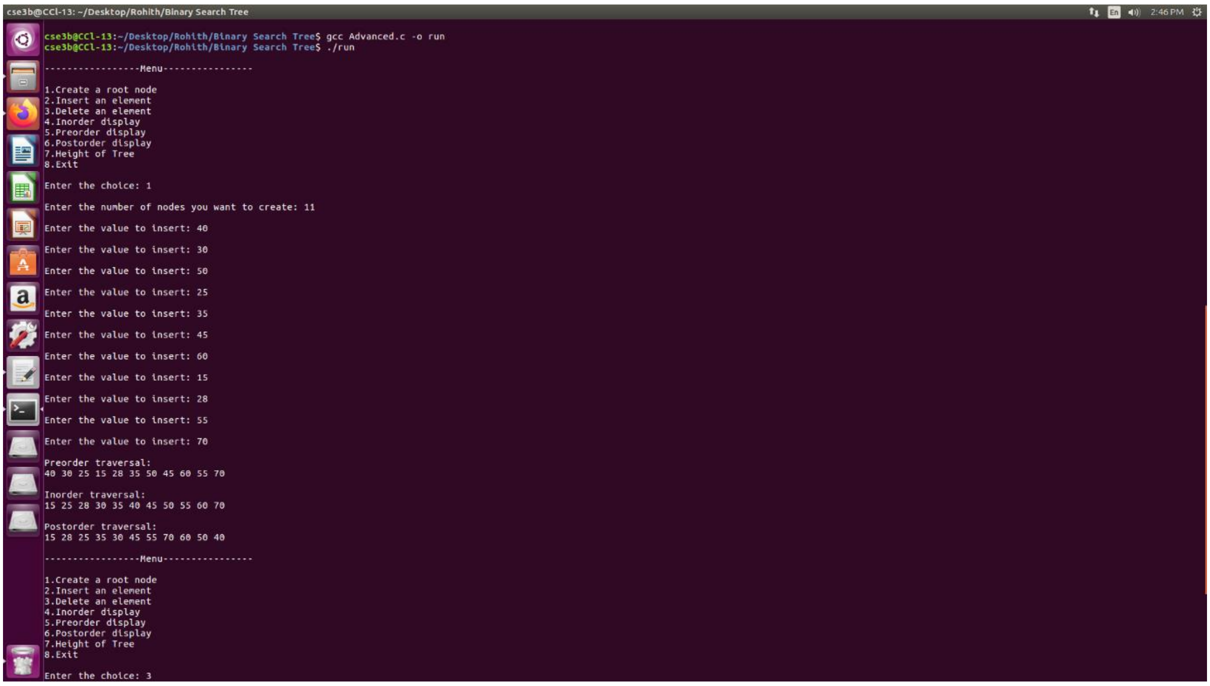
```
    printf("\n1.Create a root node\n2.Insert an element\n3.Delete an
element\n4.Inorder display\n5.Preorder display\n6.Postorder
display\n7.Height of Tree\n8.Exit\n");
    printf("\nEnter the choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("\nEnter the number of nodes you want to create: ");
        scanf("%d",&n);
        if(n==0)
          printf("\nInvalid number\n");
        else if(n==1)
        {
          printf("\nEnter the value of root node: ");
          scanf("%d",&x);
          root=new_node(x);
        }
        else
        {
          while(n>0)
          {
            printf("\nEnter the value to insert: ");
            scanf("%d",&x);
            root=insert(root,x);
            n--;
           }
        }
        printf("\nPreorder traversal:\n");
        preorder(root);
        printf("\n");
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        printf("\nPostorder traversal:\n");
        postorder(root);
        printf("\n");
        break;
      case 2:
        printf("\nEnter the value to insert: ");
        scanf("%d",&x);
        root=insert(root,x);
```

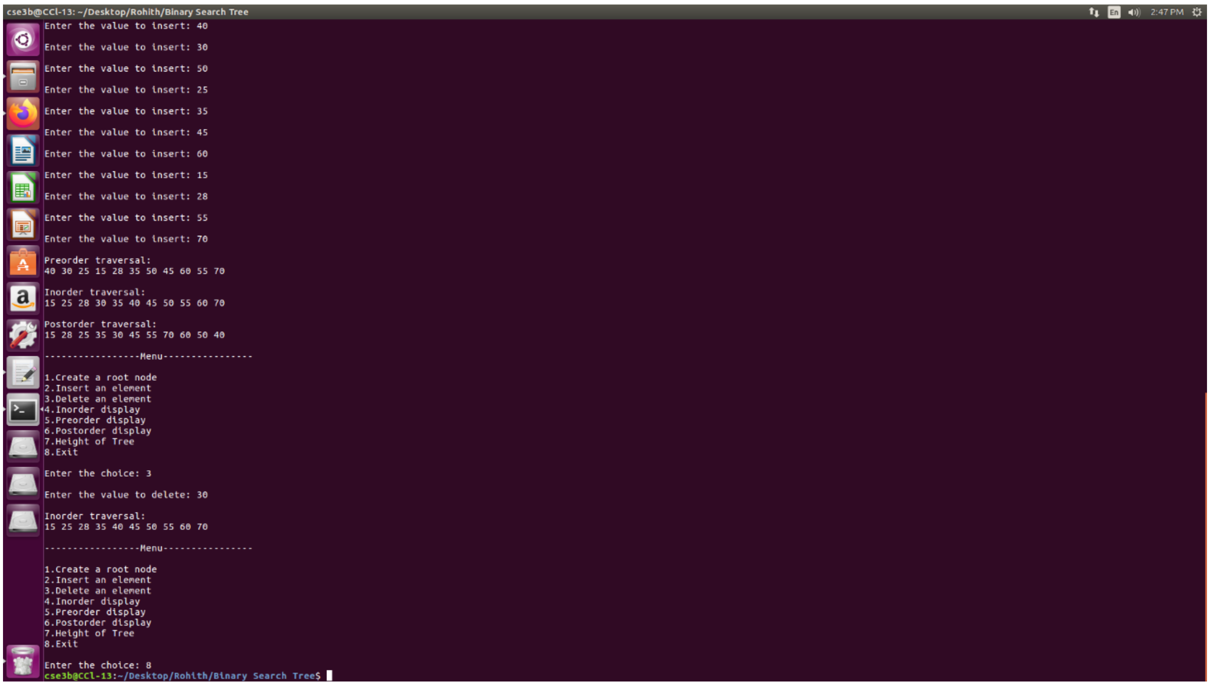
Name: M.Rohith
3122 21 5001 085

```
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 3:
        printf("\nEnter the value to delete: ");
        scanf("%d",&x);
        root=delete(root,x);
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 4:
        printf("\nInorder traversal:\n");
        inorder(root);
        printf("\n");
        break;
      case 5:
        printf("\nPreorder traversal:\n");
        preorder(root);
        printf("\n");
        break;
      case 6:
        printf("\nPostorder traversal:\n");
        postorder(root);
        printf("\n");
        break;
      case 7:
        height=maxDepth(root);
        printf("\nHeight of BinarySearchTree is: %d\n",height);
        break;
      case 8:
        exit(0);
      default:
        printf("\nInvalid choice\n");
    }
  } while (choice!=8);
  return 0;
}
```

Ex-no: 6 Name: M.Rohith
Date: 7-12-2022 3122 21 5001 085

## Output:


```
cse3b@CCl-13:~/Desktop/Rohith/Binary Search Tree$ gcc Advanced.c -o run
cse3b@CCl-13:~/Desktop/Rohith/Binary Search Tree$ ./run

----------------Menu----------------

1.Create a root node
2.Insert an element
3.Delete an element
4.Inorder display
5.Preorder display
6.Postorder display
7.Height of Tree
8.Exit

Enter the choice: 1

Enter the number of nodes you want to create: 11

Enter the value to insert: 40

Enter the value to insert: 30

Enter the value to insert: 50

Enter the value to insert: 25

Enter the value to insert: 35

Enter the value to insert: 45

Enter the value to insert: 60

Enter the value to insert: 15

Enter the value to insert: 28

Enter the value to insert: 55

Enter the value to insert: 70

Preorder traversal:
40 30 25 15 28 35 50 45 60 55 70

Inorder traversal:
15 25 28 30 35 40 45 50 55 60 70

Postorder traversal:
15 28 25 35 30 45 55 70 60 50 40

----------------Menu----------------

1.Create a root node
2.Insert an element
3.Delete an element
4.Inorder display
5.Preorder display
6.Postorder display
7.Height of Tree
8.Exit

Enter the choice: 3
```


```
Enter the value to insert: 40

Enter the value to insert: 30

Enter the value to insert: 50

Enter the value to insert: 25

Enter the value to insert: 35

Enter the value to insert: 45

Enter the value to insert: 60

Enter the value to insert: 15

Enter the value to insert: 28

Enter the value to insert: 55

Enter the value to insert: 70

Preorder traversal:
40 30 25 15 28 35 50 45 60 55 70

Inorder traversal:
15 25 28 30 35 40 45 50 55 60 70

Postorder traversal:
15 28 25 35 30 45 55 70 60 50 40

----------------Menu----------------

1.Create a root node
2.Insert an element
3.Delete an element
4.Inorder display
5.Preorder display
6.Postorder display
7.Height of Tree
8.Exit

Enter the choice: 3

Enter the value to delete: 30

Inorder traversal:
15 25 28 35 40 45 50 55 60 70

----------------Menu----------------

1.Create a root node
2.Insert an element
3.Delete an element
4.Inorder display
5.Preorder display
6.Postorder display
7.Height of Tree
8.Exit

Enter the choice: 8
cse3b@CCl-13:~/Desktop/Rohith/Binary Search Tree$
```

## Result:

Hence C program using Binary Search trees data structure has been implemented to perform various operations.