**BINARY SEARCH TREE**

**LAB # 10**



**Data Structures & Algorithms**

Submitted by: **Shah Raza**

Registration No: **18PWCSE1658**

Class Section: **B**

“On my honor, as a student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to: **Dr. Khurram Shehzad Khattak**

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Lab Objectives:**

Objectives of this lab are as follows:

* Implement basic operations on Binary Search Tree

**Task # 1:**

Implement Binary Search Tree with the mentioned features.

**Code:**

#include <iostream>

#include<stdlib.h>

using namespace std;

struct Bst

{

int data;

Bst \*left;

Bst \*right;

};

Bst \*createnode(int data)

{

Bst \*Node = new Bst;

Node->data=data;

Node->left=NULL;

Node->right=NULL;

return Node;

}

Bst \*insertnode(Bst \*Root,int data)

{

if(Root==NULL)

{

Root=createnode(data);

}

else if (data <= Root->data)

{

Root->left=insertnode(Root->left,data);

}

else

{

Root->right=insertnode(Root->right,data);

}

return Root;

}

void inOrder(Bst\* Root)

{

if (Root != NULL)

{

inOrder(Root->left);

cout<<Root->data<<" ";

inOrder(Root->right);

}

}

void preOrder(Bst\* Root)

{

if (Root != NULL)

{

cout<<Root->data<<" ";

preOrder(Root->left);

preOrder(Root->right);

}

}

void postOrder(Bst\* Root)

{

if (Root != NULL)

{

postOrder(Root->left);

postOrder(Root->right);

cout<<Root->data<<" ";

}

}

Bst\* minValue(Bst\* node)

{

Bst\* current = node;

while (current->left != NULL)

current = current->left;

return current;

}

Bst\* deleteNode(Bst\* Root, int key)

{

if (Root == NULL) {return Root;}

if (key < Root->data)

{Root->left = deleteNode(Root->left, key);}

else if (key > Root->data)

{Root->right = deleteNode(Root->right, key);}

else

{ if (Root->left == NULL)

{

Bst \*temp = Root->right;

free(Root);

return temp;

}

else if (Root->right == NULL)

{

Bst \*temp = Root->left;

free(Root);

return temp;

}

Bst\* temp = minValue(Root->right);

Root->data = temp->data;

Root->right = deleteNode(Root->right, temp->data);

}

return Root;

}

int Height(Bst\* Root)

{

if (Root==NULL)

return 0;

else

{

int lH = Height(Root->left);

int rH = Height(Root->right);

if (lH > rH)

return(lH+1);

else

return(rH+1);

}

}

int main()

{

int data,n,choice, item,S;;

Bst \*Root;

Root=NULL;

while (1)

{

cout<<"\n-------------"<<endl;

cout<<"Operations on Binary Search Tree"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Tree"<<endl;

cout<<"2.Insert Node"<<endl;

cout<<"3.Delete Node"<<endl;

cout<<"4.Inorder Traversal"<<endl;

cout<<"5.Preorder Traversal"<<endl;

cout<<"6.Postorder Traversal"<<endl;

cout<<"7.Height of Tree"<<endl;

cout<<"8.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter number of elements:";

cin>>n;

for(int i=0;i<n;i++)

{ cout<<"Enter Data: ";

cin>>data;

Root=insertnode(Root,data);

}

break;

case 2:

cout<<"Enter Data: ";

cin>>data;

Root=insertnode(Root,data);

break;

case 3:

cout<<"Enter data to delete: ";

cin>>data;

Root=deleteNode(Root,data);

cout<<"Elements in the tree are: ";

inOrder(Root);

cout<<"\n";

break;

case 4:

cout<<"Elements in the tree are: ";

inOrder(Root);

cout<<"\n";

break;

case 5:

cout<<"Elements in the tree are: ";

preOrder(Root);

cout<<"\n";

break;

case 6:

cout<<"Elements in the tree are: ";

postOrder(Root);

cout<<"\n";

break;

case 7:

cout<<"The Height of the tree is: "<<Height(Root);

cout<<"\n";

break;

case 8:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

}

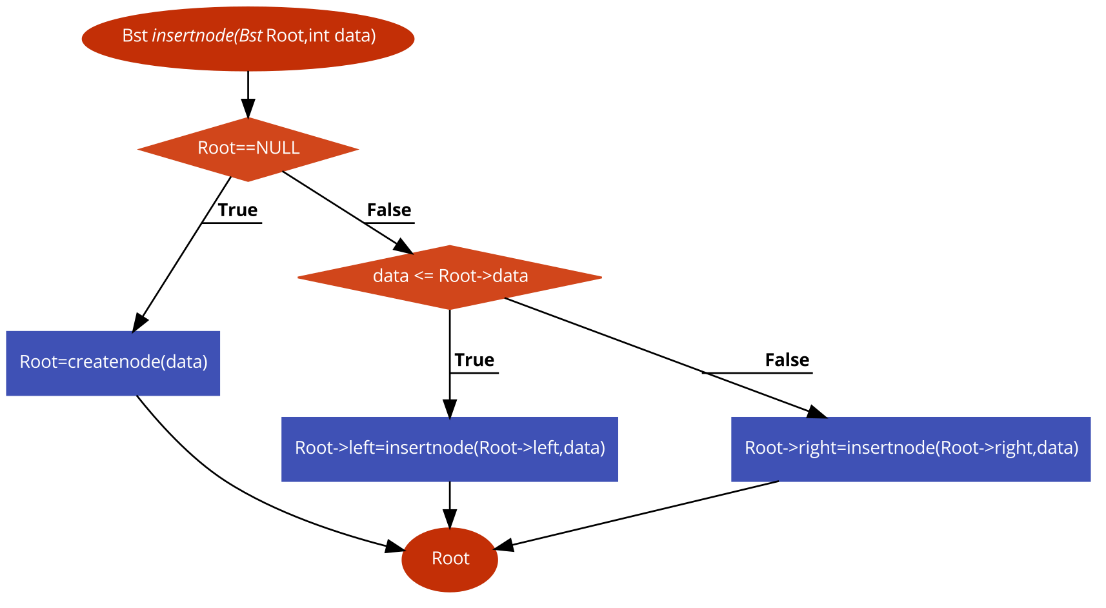
}

return 0;

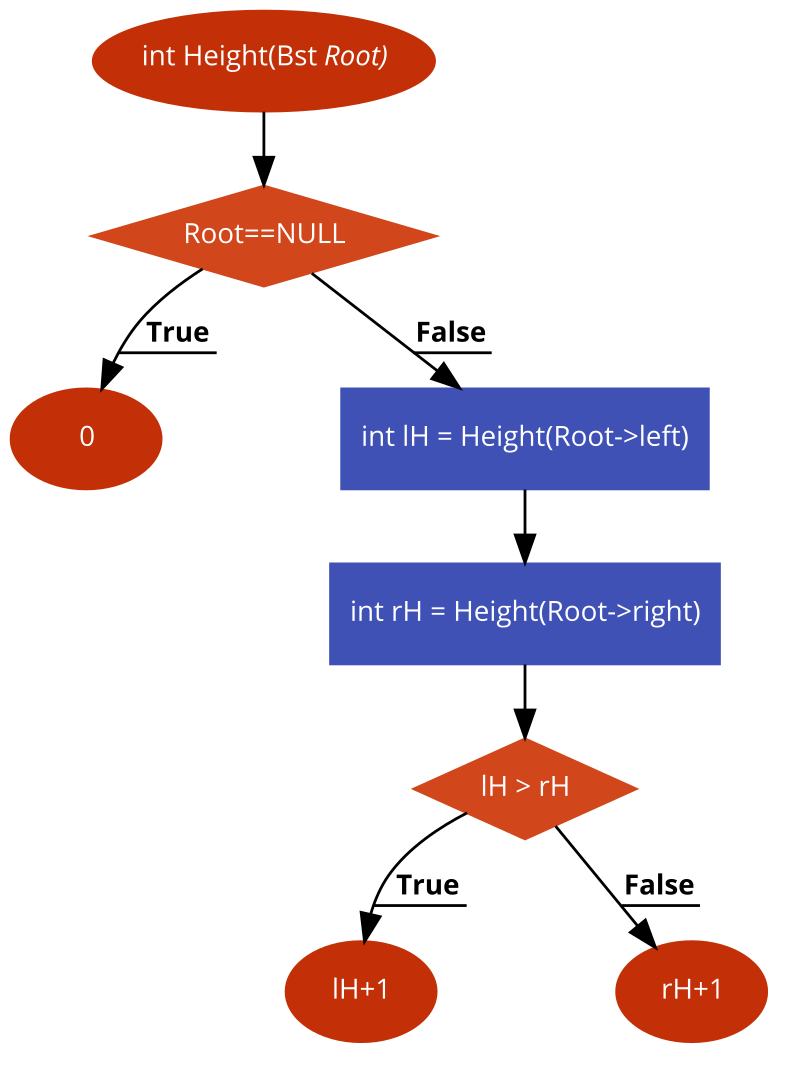
}

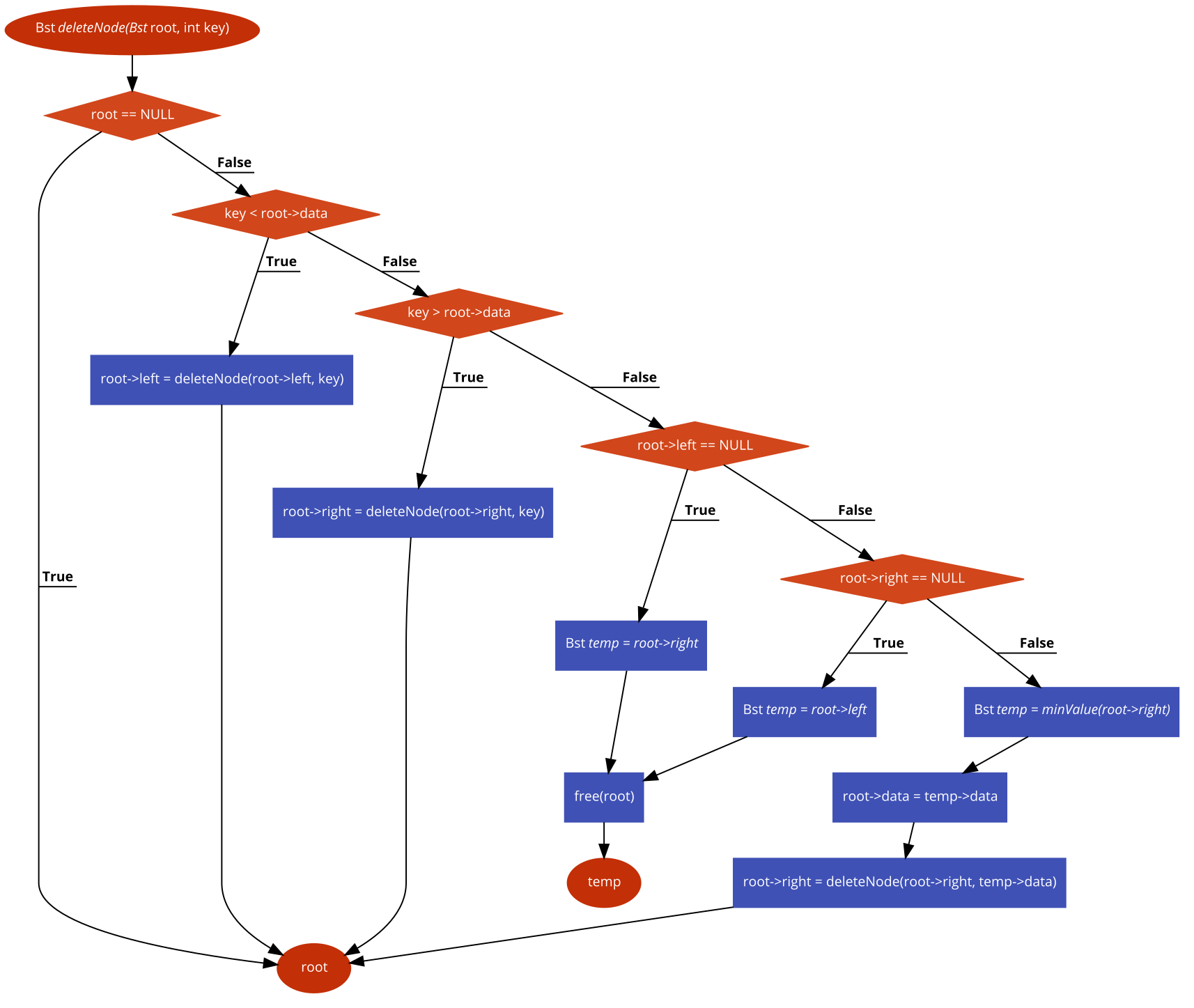
**Flow Chart:**

* **Insert Node function:**

****

* **Height function:**



* **Delete Node function:**

**Pseudo-Code:**

* InsertNode function:
  + if Root is equal to NULL
    - Then call create node function, Root=createnode(data)
  + else if data less than or equal to Rootdata
    - Then recursively call insernode function and pass the left tree,

Root->left=insertnode(Root->left,data)

* + else
    - Recursively call insernode function and pass the right tree,

Root->right=insertnode(Root->right,data)

* + return Root
* DeleteNode function:
  + if Root is equal to NULL
    - Then return Root
  + if key is less than Rootdata
    - Then recursively call deleteNode function and pass it the left tree,

Root->left = deleteNode(Root->left, key)

* + else if key is greater than Rootdata
    - Then recursively call deleteNode function and pass it the right tree,

Root->right = deleteNode(Root->right, key)

* + else
    - if Root->left is equal to NULL
      * Then create a temporary node, Bst \*temp = Root->right
      * Free Root
      * return temp
* else if Root->right is equal to NULL
  + - * Then create a temporary node, Bst \*temp = Root->left
      * Free Root
      * return temp
* Create a temporary node and call minimum value function,

Bst\* temp = minValue(Root->right)

* Rootdata = temp->data
* Root->right = deleteNode(Root->right, temp->data);
  + return Root
* Height function:
  + if Root is equal to NULL
    - Then return 0
  + else
    - Recursively call Height function and pass it the left tree,

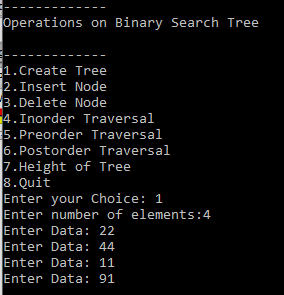
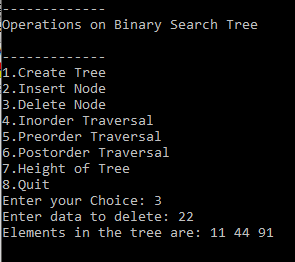
leftH = Height(Root->left)

* + - Recursively call Height function and pass it the right tree,

rightH = Height(Root->right);

* + if leftH is greater than rightH
    - Then return leftH+1
  + else
    - return rightH+1

**Output:**

** **