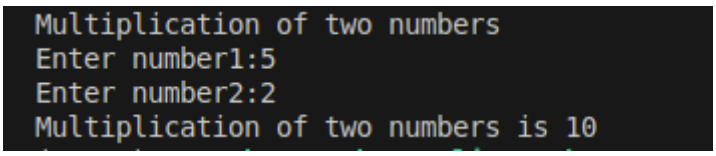
QUESTION # 1

```
#include<iostream>

using namespace std;


int multiply(int a, int b)
{
if(b==1)
{
//this return will used to end the recursion
return a;
}

b-=1;
a+=multiply(a,b);
//this return handle the other return of function
return a;

}


int main(void)
{
cout<<"Multiplication of two numbers "<<endl;
cout<<"Enter number1:";
int num1;
cin>>num1;
int num2;
cout<<"Enter number2:";
cin>>num2;
cout<<"Multiplication of two numbers is
"<<multiply(num1,num2)<<endl;
return 0;
}
```

OUTPUT:



QUESTION # 2

```
#include<iostream>

using namespace std;



//It is code for printing tree to check my other functions are working correctly or not
// struct Trunk
// {
// Trunk *prev;
// string str;

// Trunk(Trunk *prev, string str)
// {
// this->prev = prev;
// this->str = str;
// }
// };

// // Helper function to print branches of the binary tree
// void showTrunks(Trunk *p)
// {
// if (p == NULL)
// return;

// showTrunks(p->prev);

// cout << p->str;
// }
class node
{
public:
node *left;
node *right;
int info;
node(int value)
{
info=value;
left=NULL;
```

```
right=NULL;
}

};


//***********Just to check the tree is working correctly or
not***********
// void printTree(node *&root, Trunk *prev, bool isRight)
// {
// if (root == NULL)
// return;
// string prev_str = " ";
// Trunk *trunk = new Trunk(prev, prev_str);

// printTree(root->right, trunk, true);

// if (!prev)
// trunk->str = "---";
// else if (isRight)
// {
// trunk->str = ".---";
// prev_str = " |";
// }
// else
// {
// trunk->str = "`---";
// prev->str = prev_str;
// }

// showTrunks(trunk);
// cout << root->info << endl;

// if (prev)
// prev->str = prev_str;
// trunk->str = " |";

// printTree(root->left, trunk, false);
// }




class BST
{
```

```
public:
node *root;
BST()
{
root=NULL;
}



//root passed by reference to change the root of the tree
void insertion_in_BST(node *&passed_root,int value)
{
if(root==NULL)
{
passed_root = new node(value);
root = passed_root;
return;
}
if(passed_root->info == value)
{
cout<<"The value is already present in BST"<<endl;
return;
}

if(passed_root->info > value)
{
if(passed_root->left!=NULL)
{
insertion_in_BST(passed_root->left,value);
return;
}
else
{
passed_root->left=new node(value);
return;
}
}

if(passed_root->info<value)
{
if(passed_root->right!=NULL)
{
insertion_in_BST(passed_root->right,value);
return;
```

```
}
else
{
passed_root->right=new node(value);
return;
}
}

return;
}


void In_Order_traversal(node *Passed_root)
{
if(Passed_root==NULL)
return;
In_Order_traversal(Passed_root->left);
cout<<Passed_root->info<<" ";
In_Order_traversal(Passed_root->right);

return;

}


void Post_Order_traversal(node *Passed_root)
{
if(Passed_root == NULL)
return;

Post_Order_traversal(Passed_root->left);
Post_Order_traversal(Passed_root->right);
cout<<Passed_root->info<<" ";

}


void Pre_Order_traversal(node *Passed_root)
{
if(Passed_root==NULL)
{
return;
}
cout<<Passed_root->info<<" ";
Pre_Order_traversal(Passed_root->left);
```

```
Pre_Order_traversal(Passed_root->right);
return;

}

void Smallest_in_BST(node *passed_node)
{
if(root==NULL)
{
cout<<"THe BST is empty"<<endl;
return;
}
//As tree is BST so the smallest value will be in the left most node
if(passed_node->left==NULL)
{
cout<<"The Smallest Element in BST is "<<passed_node-
>info<<endl;
return;
}
Smallest_in_BST(passed_node->left);
return;
}

int Counting_Nodes_in_BST(node *Passed_node)
{
if(root==NULL)
{
cout<<"The BST is Empty"<<endl;
return 0;
}
if(Passed_node->left ==NULL && Passed_node->right==NULL)
{
return 1;
}
int count=0;
if(Passed_node->left!=NULL)
{
count+=Counting_Nodes_in_BST(Passed_node->left);
}
if(Passed_node->right!=NULL)
{
count+=Counting_Nodes_in_BST(Passed_node->right);
}
return count+1;
```

```
}

};


int main(void)
{
BST obj;


while(1)
{
cout << "\n\n ************** Select Option ******************.\n";
cout << "\n Enter any of choices.\n";
cout << "\n 1 : Adding (inserting) node in BST.\n";
cout << "\n 2 : Print in Order Traversal of BST .\n";
cout << "\n 3 : Print Pre Order Traversal of BST .\n";
cout << "\n 4 : Print Post Order Traversal of BST .\n";
cout << "\n 5 : Find Smallest Element in BST .\n";
cout << "\n 6 : Number of Nodes in the BST .\n";
cout << "\n 7 : print tree .\n";
cout << "\n 8 : Quitting the Program.\n";

int choice;

cin>>choice;
switch(choice)
{
case 1:
{
cout<<"Enter the value to insert in BST"<<endl;
int value;
cin>>value;
obj.insertion_in_BST(obj.root,value);
break;
}
case 2:
{
cout<<"In Order Traversal of BST"<<endl;
obj.In_Order_traversal(obj.root);
break;
}
case 3:
{
cout<<"Pre Order Traversal of BST"<<endl;
```

```
obj.Pre_Order_traversal(obj.root);
break;
}
case 4:
{
cout<<"Post Order Traversal of BST"<<endl;
obj.Post_Order_traversal(obj.root);
break;
}
case 5:
{
cout<<"Smallest Element in BST"<<endl;
obj.Smallest_in_BST(obj.root);
break;
}
case 6:
{
cout<<"Number of Nodes in the BST"<<endl;
cout<<obj.Counting_Nodes_in_BST(obj.root)<<endl;
break;
}
case 7:
{
cout<<"If nothing is printed then uncomment the printing code and
functions of tree"<<endl;
// printTree(obj.root,NULL,false);
break;
}
case 8:
{
return 0;
}




}
}


return 0;

}
```

OUTPUT:

1)

```
********** Select Option *****************.

Enter any of choices.

1 : Adding (inserting) node in BST.

2 : Print in Order Traversal of BST .

3 : Print Pre Order Traversal of BST .

4 : Print Post Order Traversal of BST .

5 : Find Smallest Element in BST .

6 : Number of Nodes in the BST .

7 : print tree .

8 : Quitting the Program.
1
Enter the value to insert in BST
15
```

2)

```
********** Select Option *****************.

Enter any of choices.

1 : Adding (inserting) node in BST.

2 : Print in Order Traversal of BST .

3 : Print Pre Order Traversal of BST .

4 : Print Post Order Traversal of BST .

5 : Find Smallest Element in BST .

6 : Number of Nodes in the BST .

7 : print tree .

8 : Quitting the Program.
7
        .---25
    .---20
   |    `---19
---15
   |    .---12
    `---10
        `---5
            `---1
```

3)

```
********** Select Option **********.

Enter any of choices.

1 : Adding (inserting) node in BST.

2 : Print in Order Traversal of BST .

3 : Print Pre Order Traversal of BST .

4 : Print Post Order Traversal of BST .

5 : Find Smallest Element in BST .

6 : Number of Nodes in the BST .

7 : print tree .

8 : Quitting the Program.
5
Smallest Element in BST
The Smallest Element in BST is 1
```

4)

```
********** Select Option **********.

Enter any of choices.

1 : Adding (inserting) node in BST.

2 : Print in Order Traversal of BST .

3 : Print Pre Order Traversal of BST .

4 : Print Post Order Traversal of BST .

5 : Find Smallest Element in BST .

6 : Number of Nodes in the BST .

7 : print tree .

8 : Quitting the Program.
6
Number of Nodes in the BST
8
```

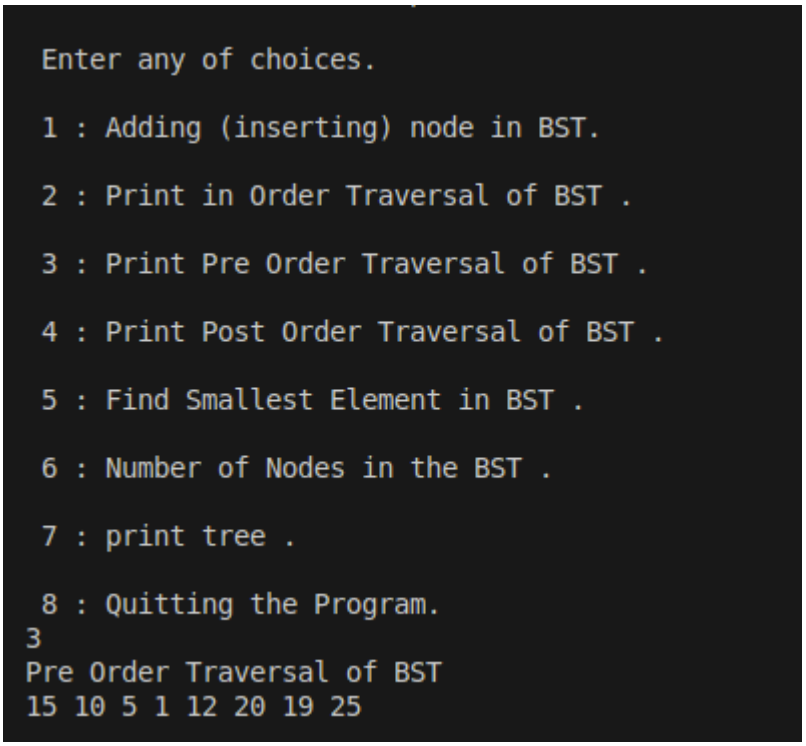
5)

```
Enter any of choices.

1 : Adding (inserting) node in BST.

2 : Print in Order Traversal of BST .

3 : Print Pre Order Traversal of BST .

4 : Print Post Order Traversal of BST .

5 : Find Smallest Element in BST .

6 : Number of Nodes in the BST .

7 : print tree .

8 : Quitting the Program.
2
In Order Traversal of BST
1 5 10 12 15 19 20 25
```

6)

```
Enter any of choices.

1 : Adding (inserting) node in BST.

2 : Print in Order Traversal of BST .

3 : Print Pre Order Traversal of BST .

4 : Print Post Order Traversal of BST .

5 : Find Smallest Element in BST .

6 : Number of Nodes in the BST .

7 : print tree .

8 : Quitting the Program.
4
Post Order Traversal of BST
1 5 12 10 19 25 20 15
```

7)



QUESTION # 3:

```
#include<iostream>

using namespace std;

//It is code for printing tree to check my other functions are working correctly or not
struct Trunk
{
Trunk *prev;
string str;

Trunk(Trunk *prev, string str)
{
this->prev = prev;
this->str = str;
}
};
```

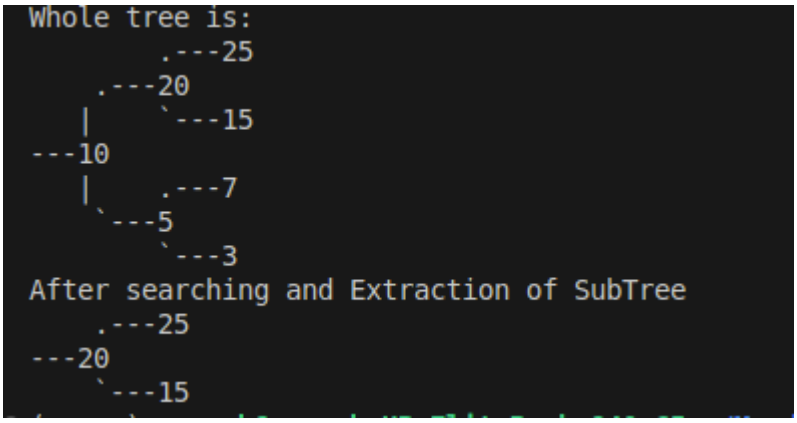
```
// Helper function to print branches of the binary tree
void showTrunks(Trunk *p)
{
if (p == NULL)
return;

showTrunks(p->prev);
std::cout<< p->str;
}




class Node
{
public:
int info;
Node *left;
Node *right;
Node(int value)
{
info=value;
left=NULL;
right=NULL;
}
};


class BST
{
public:
Node *root;
BST()
{
root=NULL;
}

Node* Value_and_Sub_tree(Node *n, int value)
{
if(n==NULL)
{
return NULL;
}
if(n->info==value)
```

```
{
return n;
}
Node *curr;
curr=Value_and_Sub_tree(n->left,value);
if(curr!=NULL)
return curr;
curr=Value_and_Sub_tree(n->right,value);
if(curr!=NULL)
return curr;
return NULL;
}
};

void printTree(Node *&root, Trunk *prev, bool isRight)
{
if (root == NULL)
return;
string prev_str = " ";
Trunk *trunk = new Trunk(prev, prev_str);

printTree(root->right, trunk, true);

if (!prev)
trunk->str = "---";
else if (isRight)
{
trunk->str = ".---";
prev_str = " |";
}
else
{
trunk->str = "`---";
prev->str = prev_str;
}

showTrunks(trunk);
cout<< root->info << endl;

if (prev)
prev->str = prev_str;
trunk->str = " |";

printTree(root->left, trunk, false);
```

```
}

int main(void)
{
BST obj;
obj.root=new Node(10);
obj.root->left=new Node(5);
obj.root->right=new Node(20);
obj.root->left->left=new Node(3);
obj.root->left->right=new Node(7);
obj.root->right->left=new Node(15);
obj.root->right->right=new Node(25);

cout<<"Whole tree is: \n";
printTree(obj.root,NULL,false);
obj.root=obj.Value_and_Sub_tree(obj.root,20);
//obj.root=obj.Value_and_Sub_tree(obj.root,1);

cout<<"After searching and Extraction of SubTree"<<endl;
if(obj.root!=NULL)
printTree(obj.root,NULL,false);
else
cout<<"Value not found in tree"<<endl;
return 0;
}
```

OUPUT:

1)



2)