Name: **Muhammad Anas Baig**

Enrollment No.: **01-134152-037**

Section: **BS(CS)-4A**

LAB-JOURNAL-12

Exercise 1:

Complete the given class to implement a binary search tree.

```
class Node
{
public:
      Node *left;
      Node *right;
      int data;
} ;

class bst
{
      Node *root;
      public:
      bst();
      bool isempty();
      void insert(int item);
      bool search(int item);
};
```

Solution:

node.h File:

```
1.  #pragma once
2.  class node
3.  {
4.  public:
5.      node *left;
6.      node *right;
7.      int data;
8.  public:
9.      node(void);
10. };
```

node.cpp File:

```
1.  #include "node.h"
2.
3.  node::node(void)
4.  {
5.  }
```

binarySearchTree.h File:

```
#include "node.h"

#pragma once
class binarySearchTree
{
public:
    node *root;
public:
    binarySearchTree(void);
    bool isEmpty();
    void insert(int item);
    bool search(int item);
};
```

binarySearchTree.cpp File:

```
#include "binarySearchTree.h"
#include "node.h"
#include <iostream>
using namespace std;

binarySearchTree::binarySearchTree(void)
{
    root = NULL;
}

bool binarySearchTree::isEmpty()
{
    return root == NULL;
}

void binarySearchTree::insert(int item)
{
    node * ptr = root;
    node * prev = 0;

    while (ptr != NULL)
    {
        prev = ptr;
        if (item <  ptr->data)
        {
            ptr = ptr->left;
        }
        else if(item > ptr->data)
            {
                ptr = ptr->right;
        }
        else
        {
            cout<<"Value already exist";return ;
        }
    }
    node * temp = new node;
    temp->data = item;
    temp->left = 0;
    temp->right = 0;

    if(prev==0)
    {
        root = temp;
    }
```

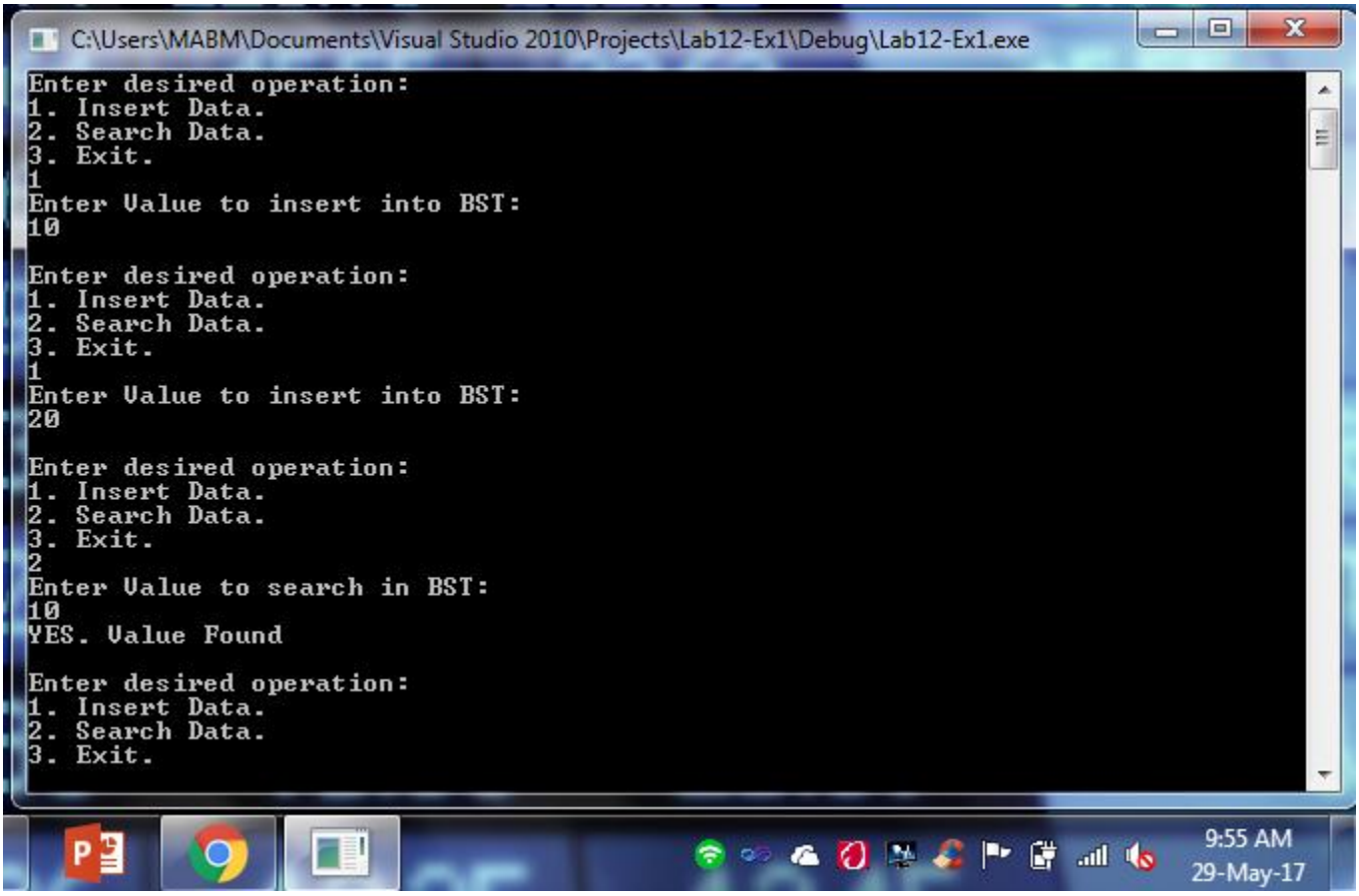
```
46.     else if (item < prev->data)
47.     {
48.         prev->left = temp;
49.     }
50.     else
51.     {
52.         prev->right = temp;
53.     }
54. }
55.
56. bool binarySearchTree::search(int item)
57. {
58.     node * ptr = root;
59.     bool found = false;
60.
61.     for(;;)
62.     {
63.         if(found || ptr == NULL)
64.         {
65.                 break;
66.         }
67.         if(item< ptr->data)
68.         {
69.             ptr = ptr->left;
70.         }
71.         else if (item > ptr->data)
72.         {
73.             ptr = ptr->right;
74.         }
75.         else
76.         {
77.             found = true;
78.         }
79.     }
80.     return found;
81. }
```

main.cpp File:

```
1.  #include "binarySearchTree.h"
2.  #include "node.h"
3.  #include "conio.h"
4.  #include <iostream>
5.  using namespace std;
6.
7.  int main()
8.  {
9.      binarySearchTree b;
10.     int data, choice1, choice2;
11.
12.     cout<<"Enter desired operation:"<<endl;
13.     cout<<"1. Insert Data."<<endl;
14.     cout<<"2. Search Data."<<endl;
15.     cout<<"3. Exit."<<endl;
16.     cin>>choice1;
17.     do
18.     {
19.         if(choice1 ==1)
20.         {
21.             cout<<"Enter Value to insert into BST:"<<endl;
22.             cin>>data;
23.             b.insert(data);
24.         }
25.         else if( choice1 == 2)
```

```
26.         {
27.             cout<<"Enter Value to search in BST:"<<endl;
28.             cin>>data;
29.
30.             if( b.search(data) )
31.             {
32.                 cout<<"YES. Value Found"<<endl;
33.             }
34.             else
35.             {
36.                 cout<<"NO. Value Not Found"<<endl;
37.             }
38.         }
39.         else
40.         {
41.              break;
42.         }
43.         cout<<endl;
44.         cout<<"Enter desired operation:"<<endl;
45.         cout<<"1. Insert Data."<<endl;
46.         cout<<"2. Search Data."<<endl;
47.         cout<<"3. Exit."<<endl;
48.         cin>>choice1;
49.     }
50.     while ( choice1 != 3 );
51.
52.     getch();
53. }
```

Output:



Exercise 2:

Extend the above « BST » class to include functions for pre, post and in-order traversal of the tree. Use recursion to implement the traversal functions.

Solution:

node.h File:

```
#pragma once
class node
{
public:
    node *left;
    node *right;
    int data;
public:
    node(void);
};
```

node.cpp File:

```
#include "node.h"

node::node(void)
{
}
```

binarySearchTree.h File:

```
#include "node.h"

#pragma once
class binarySearchTree
{
public:
    node *root;
public:
    binarySearchTree(void);
    bool isEmpty();
    void insert(int item);
    bool search(int item);
    void preOrder(node *ptr);
    void inOrder(node *ptr);
    void postOrder(node *ptr);
};
```

binarySearchTree.cpp File:

```
#include "binarySearchTree.h"
#include "node.h"
#include <iostream>
using namespace std;

binarySearchTree::binarySearchTree(void)
{
    root = NULL;
}
```

```
10.
11. bool binarySearchTree::isEmpty()
12. {
13.     return root == NULL;
14. }
15.
16. void binarySearchTree::insert(int item)
17. {
18.     node * ptr = root;
19.     node * prev = 0;
20.
21.     while (ptr != NULL)
22.     {
23.         prev = ptr;
24.         if (item <  ptr->data)
25.         {
26.             ptr = ptr->left;
27.         }
28.         else if(item > ptr->data)
29.             {
30.                 ptr = ptr->right;
31.         }
32.         else
33.         {
34.             cout<<"Value already exist";return ;
35.         }
36.     }
37.     node * temp = new node;
38.     temp->data = item;
39.     temp->left = 0;
40.     temp->right = 0;
41.
42.     if(prev==0)
43.     {
44.         root = temp;
45.     }
46.     else if (item < prev->data)
47.     {
48.         prev->left = temp;
49.     }
50.     else
51.     {
52.         prev->right = temp;
53.     }
54. }
55.
56. bool binarySearchTree::search(int item)
57. {
58.     node * ptr = root;
59.     bool found = false;
60.
61.     for(;;)
62.     {
63.         if(found || ptr == NULL)
64.         {
65.                 break;
66.         }
67.         if(item< ptr->data)
68.         {
69.             ptr = ptr->left;
70.         }
71.         else if (item > ptr->data)
72.         {
73.             ptr = ptr->right;
74.         }
```

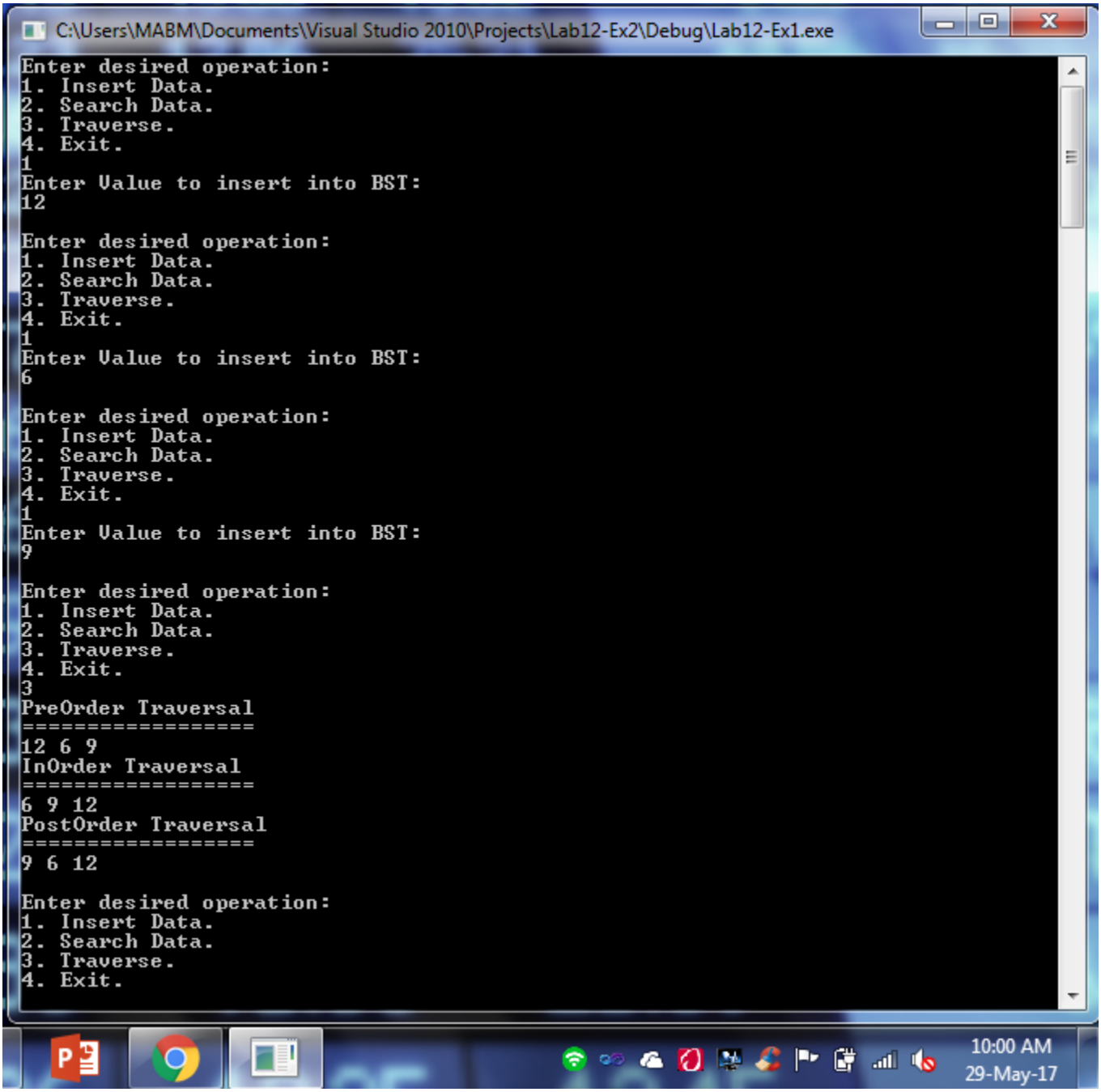
```
        else
        {
            found = true;
        }
    }
    return found;
}

void binarySearchTree::preOrder(node *ptr)
{
    if(ptr != NULL)
    {
        cout << ptr->data << " ";
        preOrder(ptr->left);
        preOrder(ptr->right);
    }

}

void binarySearchTree::inOrder(node *ptr)
{
    if(ptr != NULL)
    {
        inOrder(ptr->left);
        cout << ptr->data << " ";
               inOrder(ptr->right);

           }
       }

       void binarySearchTree::postOrder(node *ptr)
       {
           if(ptr != NULL)
           {
               postOrder(ptr->left);
               postOrder(ptr->right);
               cout << ptr->data << " ";
           }
       }
```

main.cpp File:

```
#include "binarySearchTree.h"
#include "node.h"
#include "conio.h"
#include <iostream>
using namespace std;

int main()
{
    binarySearchTree b;
    int data, choice1, choice2;

    cout<<"Enter desired operation:"<<endl;
    cout<<"1. Insert Data."<<endl;
    cout<<"2. Search Data."<<endl;
    cout<<"3. Traverse."<<endl;
    cout<<"4. Exit."<<endl;
    cin>>choice1;
    do
    {
        if(choice1 == 1)
        {
            cout<<"Enter Value to insert into BST:"<<endl;
```

```
            cin>>data;
            b.insert(data);
        }
        else if( choice1 == 2)
        {
            cout<<"Enter Value to search in BST:"<<endl;
            cin>>data;

            if( b.search(data) )
            {
                cout<<"YES. Value Found"<<endl;
            }
            else
            {
                cout<<"NO. Value Not Found"<<endl;
            }
        }
        else if( choice1 == 3 )
        {
            cout<<"PreOrder Traversal"<<endl;
            cout<<"=================="<<endl;
            b.preOrder(b.root);
            cout<<endl;

            cout<<"InOrder Traversal"<<endl;
            cout<<"=================="<<endl;
            b.inOrder(b.root);
            cout<<endl;

            cout<<"PostOrder Traversal"<<endl;
            cout<<"=================="<<endl;
            b.postOrder(b.root);
            cout<<endl;
        }
        else
        {
             break;
        }
        cout<<endl;
        cout<<"Enter desired operation:"<<endl;
        cout<<"1. Insert Data."<<endl;
        cout<<"2. Search Data."<<endl;
        cout<<"3. Traverse."<<endl;
        cout<<"4. Exit."<<endl;
        cin>>choice1;
    }
    while ( choice1 != 4 );

    getch();
}
```

Output:



Exercise 3:

Write program that creates a binary search tree using the BST class developed in Exercise.

Create a menu and perform the following operations on user inputs. (Make a separate function for each).

a. Insert a node in the binary tree.

b. Count all leaf nodes of the tree*.

c. Count all non-leaf nodes of the tree.

d. Determines the size of a binary tree by counting the number of nodes in the tre

e.

*Hints:
Use the following recursive definition: `getLeafCount(Node *)`

1. If node is NULL then return 0.
2. Else If left and right child nodes are NULL return 1.
3. Else recursively calculate leaf count of the tree using below formula.

Leaf count of a tree = Leaf count of left subtree + Leaf count of right subtree

Solution:

node.h File:

```
#pragma once
class node
{
public:
    node *left;
    node *right;
    int data;
public:
    node(void);
};
```

node.cpp File:

```
#include "node.h"

node::node(void)
{
}
```

binarySearchTree.h File:

```
#include "node.h"

#pragma once
class binarySearchTree
{
public:
    node *root;
public:
    binarySearchTree(void);
    bool isEmpty();
```

```
    void insert(int item);
    bool search(int item);
    void preOrder(node *ptr);
    void inOrder(node *ptr);
    void postOrder(node *ptr);
    int getLeafCount(node *ptr);
    int countAll(node *ptr);
    int countNonLeafNode(node *ptr);
};
```

binarySearchTree.cpp File:

```
#include "binarySearchTree.h"
#include "node.h"
#include <iostream>
using namespace std;

binarySearchTree::binarySearchTree(void)
{
    root = NULL;
}

bool binarySearchTree::isEmpty()
{
    return root == NULL;
}

void binarySearchTree::insert(int item)
{
    node * ptr = root;
    node * prev = 0;

    while (ptr != NULL)
    {
        prev = ptr;
        if (item <  ptr->data)
        {
            ptr = ptr->left;
        }
        else if(item > ptr->data)
            {
                ptr = ptr->right;
        }
        else
        {
            cout<<"Value already exist";return ;
        }
    }
    node * temp = new node;
    temp->data = item;
    temp->left = 0;
    temp->right = 0;

    if(prev==0)
    {
        root = temp;
    }
    else if (item < prev->data)
    {
        prev->left = temp;
    }
    else
    {
        prev->right = temp;
    }
```

```
}

bool binarySearchTree::search(int item)
{
    node * ptr = root;
    bool found = false;

    for(;;)
    {
        if(found || ptr == NULL)
        {
                break;
        }
        if(item< ptr->data)
        {
            ptr = ptr->left;
        }
        else if (item > ptr->data)
        {
            ptr = ptr->right;
        }
        else
        {
            found = true;
        }
    }
    return found;
}

void binarySearchTree::preOrder(node *ptr)
{
    if(ptr != NULL)
    {
        cout << ptr->data << " ";
        preOrder(ptr->left);
        preOrder(ptr->right);
    }

}

void binarySearchTree::inOrder(node *ptr)
{
    if(ptr != NULL)
    {
        inOrder(ptr->left);
        cout << ptr->data << " ";
               inOrder(ptr->right);

           }
       }

       void binarySearchTree::postOrder(node *ptr)
       {
           if(ptr != NULL)
           {
               postOrder(ptr->left);
               postOrder(ptr->right);
               cout << ptr->data << " ";
           }
       }

       int binarySearchTree::getLeafCount(node *ptr)
       {
           if( ptr == NULL )
           {
```

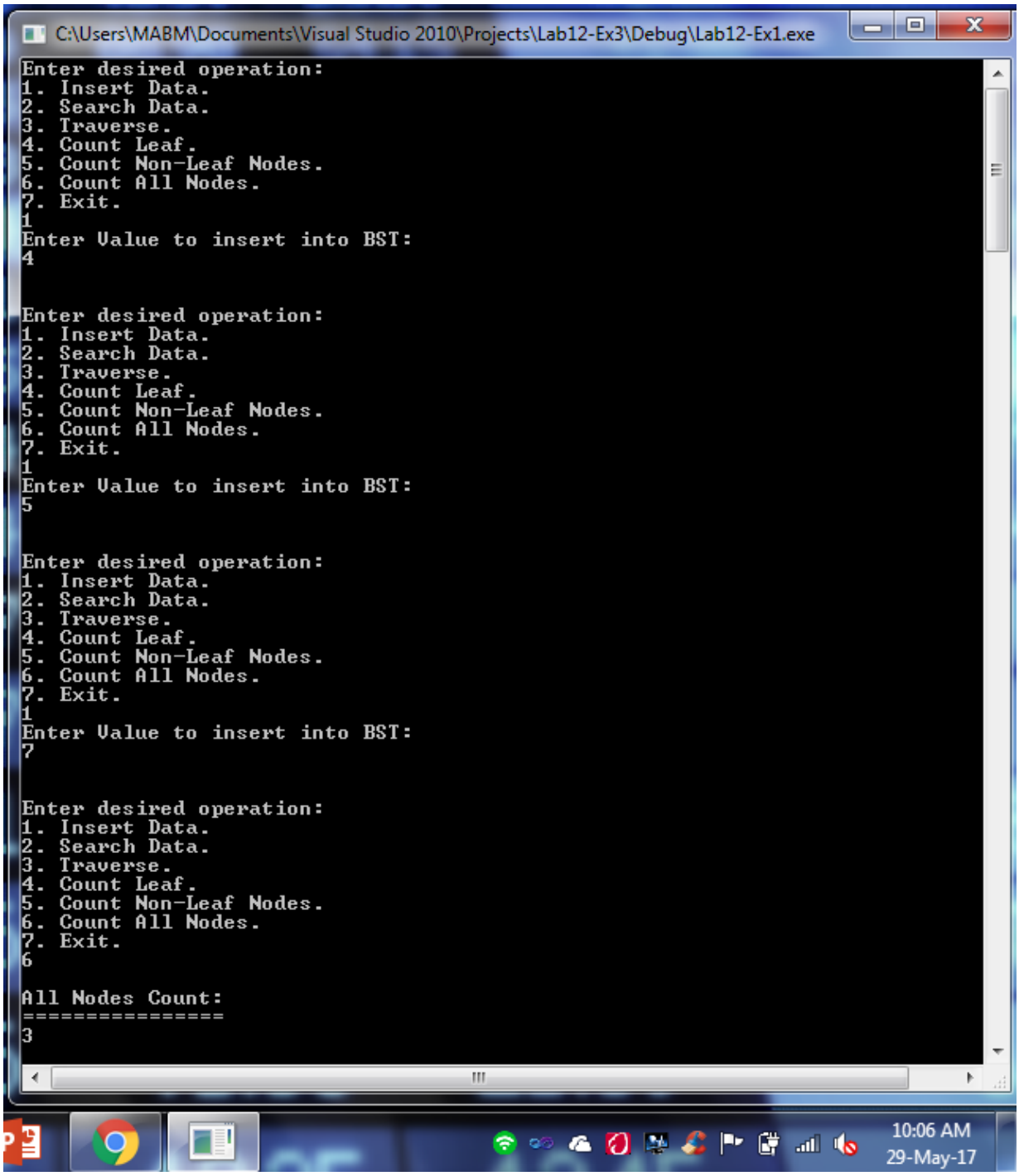
```
119.                return 0;
120.            }
121.            else if( ptr->left == NULL && ptr->right == NULL )
122.            {
123.                return 1;
124.            }
125.            else
126.            {
127.                return ( getLeafCount( ptr->left ) + getLeafCount( ptr->right ) );
128.            }
129.        }
130.
131.        int binarySearchTree::countAll(node *ptr)
132.        {
133.            if(ptr == NULL)
134.            {
135.                return 0;
136.            }
137.            else
138.            {
139.                return ( 1 + countAll( ptr->left ) + countAll( ptr->right )  );
140.            }
141.        }
142.
143.        int binarySearchTree::countNonLeafNode(node *ptr)
144.        {
145.            return( countAll(root) - ( 1 + getLeafCount(root->left) + getLeafCount(root->right) ) );
146.        }
```

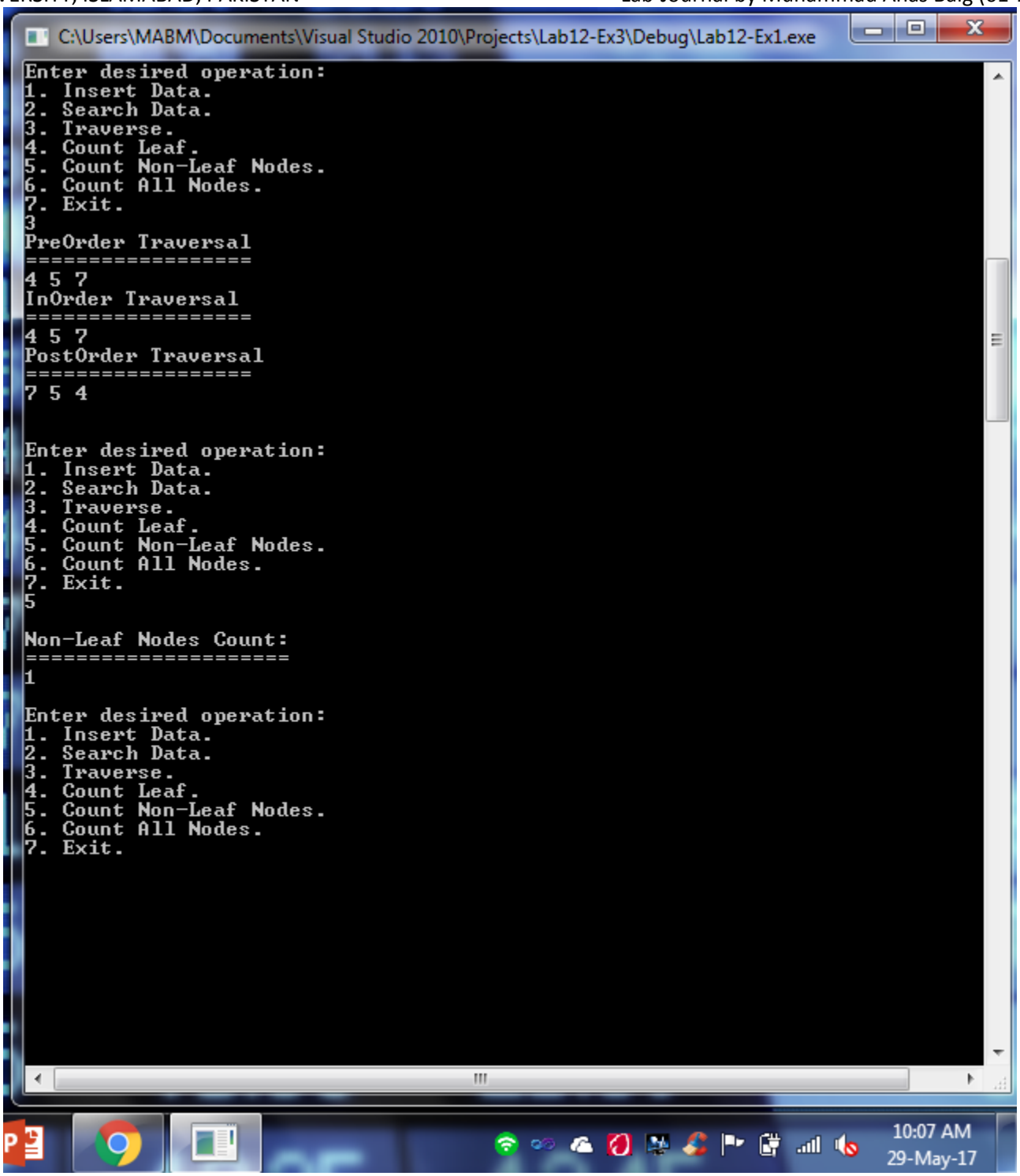
main.cpp File:

```
1.  #include "binarySearchTree.h"
2.  #include "node.h"
3.  #include "conio.h"
4.  #include <iostream>
5.  using namespace std;
6.
7.  int main()
8.  {
9.      binarySearchTree b;
10.     int data, choice1, choice2;
11.
12.     cout<<"Enter desired operation:"<<endl;
13.     cout<<"1. Insert Data."<<endl;
14.     cout<<"2. Search Data."<<endl;
15.     cout<<"3. Traverse."<<endl;
16.     cout<<"4. Count Leaf."<<endl;
17.     cout<<"5. Count Non-Leaf Nodes."<<endl;
18.     cout<<"6. Count All Nodes."<<endl;
19.     cout<<"7. Exit."<<endl;
20.     cin>>choice1;
21.     do
22.     {
23.         if(choice1 == 1)
24.         {
25.             cout<<"Enter Value to insert into BST:"<<endl;
26.             cin>>data;
27.             b.insert(data);
28.         }
29.         else if( choice1 == 2)
30.         {
31.             cout<<"Enter Value to search in BST:"<<endl;
32.             cin>>data;
33.
```

```
            if( b.search(data) )
            {
                cout<<"YES. Value Found"<<endl;
            }
            else
            {
                cout<<"NO. Value Not Found"<<endl;
            }
        }
        else if( choice1 == 3 )
        {
            cout<<"PreOrder Traversal"<<endl;
            cout<<"==================="<<endl;
            b.preOrder(b.root);
            cout<<endl;

            cout<<"InOrder Traversal"<<endl;
            cout<<"==================="<<endl;
            b.inOrder(b.root);
            cout<<endl;

            cout<<"PostOrder Traversal"<<endl;
            cout<<"==================="<<endl;
            b.postOrder(b.root);
            cout<<endl;
        }
        else if( choice1  == 4 )
        {
            cout<<"Leaf Nodes Count:"<<endl;
            cout<<"================="<<endl;
            cout<<b.getLeafCount(b.root);
        }
        else if( choice1 == 5 )
        {
            cout<<endl;
            cout<<"Non-Leaf Nodes Count:"<<endl;
            cout<<"====================="<<endl;
            cout<<b.countNonLeafNode(b.root);
        }
        else if( choice1 == 6 )
        {
            cout<<endl;
            cout<<"All Nodes Count:"<<endl;
            cout<<"================"<<endl;
            cout<<b.countAll(b.root);
        }
        else
        {
             break;
        }
        cout<<endl<<endl;
        cout<<"Enter desired operation:"<<endl;
        cout<<"1. Insert Data."<<endl;
        cout<<"2. Search Data."<<endl;
        cout<<"3. Traverse."<<endl;
        cout<<"4. Count Leaf."<<endl;
        cout<<"5. Count Non-Leaf Nodes."<<endl;
        cout<<"6. Count All Nodes."<<endl;
        cout<<"7. Exit."<<endl;
        cin>>choice1;
    }
    while ( choice1 != 7 );

    getch();
}
```

Output:


C:\Users\MABM\Documents\Visual Studio 2010\Projects\Lab12-Ex3\Debug\Lab12-Ex1.exe
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
1
Enter Value to insert into BST:
4
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
1
Enter Value to insert into BST:
5
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
1
Enter Value to insert into BST:
7
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
6
All Nodes Count:
==========
3
10:06 AM
29-May-17


C:\Users\MABM\Documents\Visual Studio 2010\Projects\Lab12-Ex3\Debug\Lab12-Ex1.exe
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
3
PreOrder Traversal
==========
4 5 7
InOrder Traversal
==========
4 5 7
PostOrder Traversal
==========
7 5 4
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
5
Non-Leaf Nodes Count:
==========
1
Enter desired operation:
1. Insert Data.
2. Search Data.
3. Traverse.
4. Count Leaf.
5. Count Non-Leaf Nodes.
6. Count All Nodes.
7. Exit.
10:07 AM
29-May-17