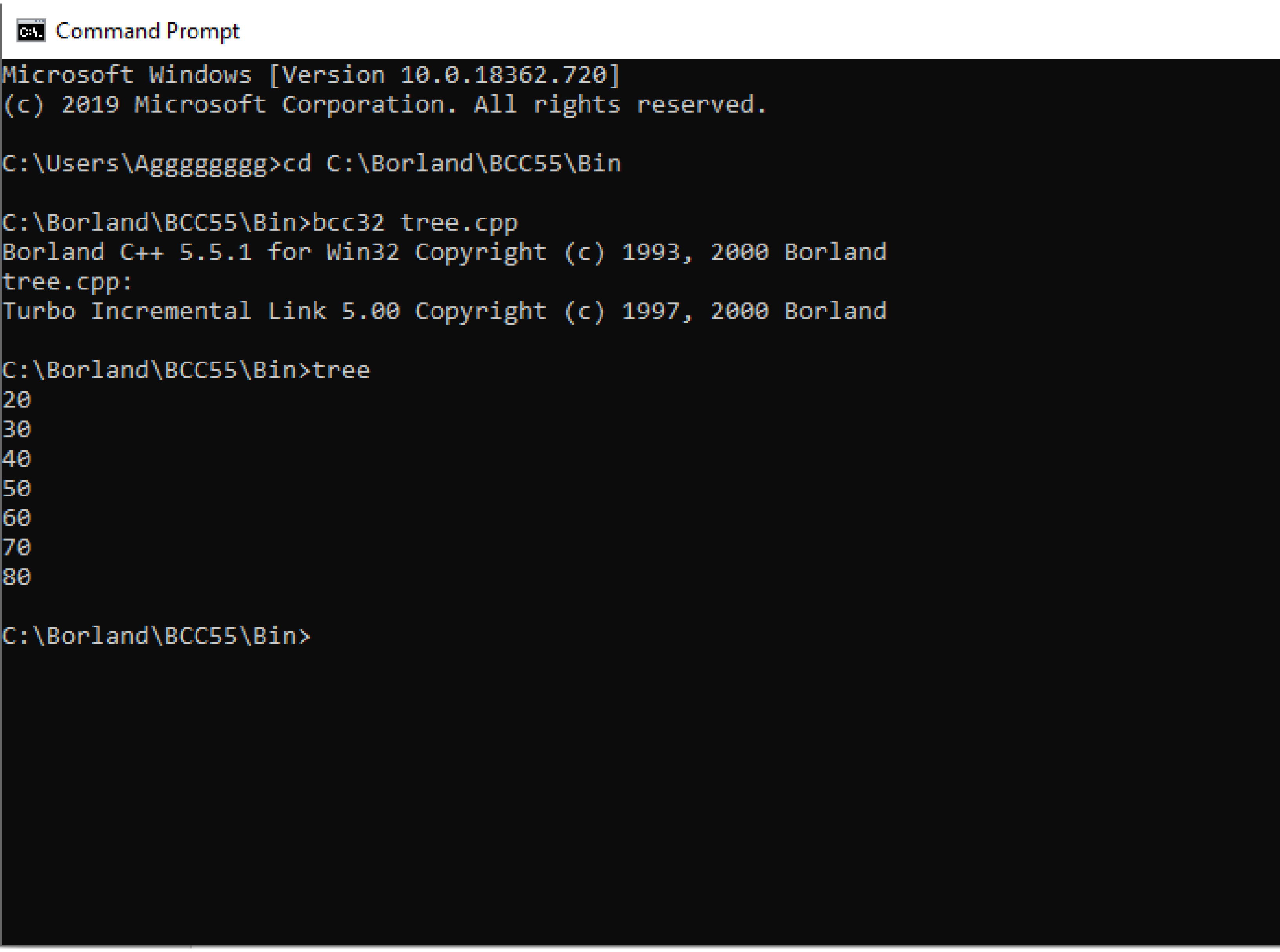


Command Prompt

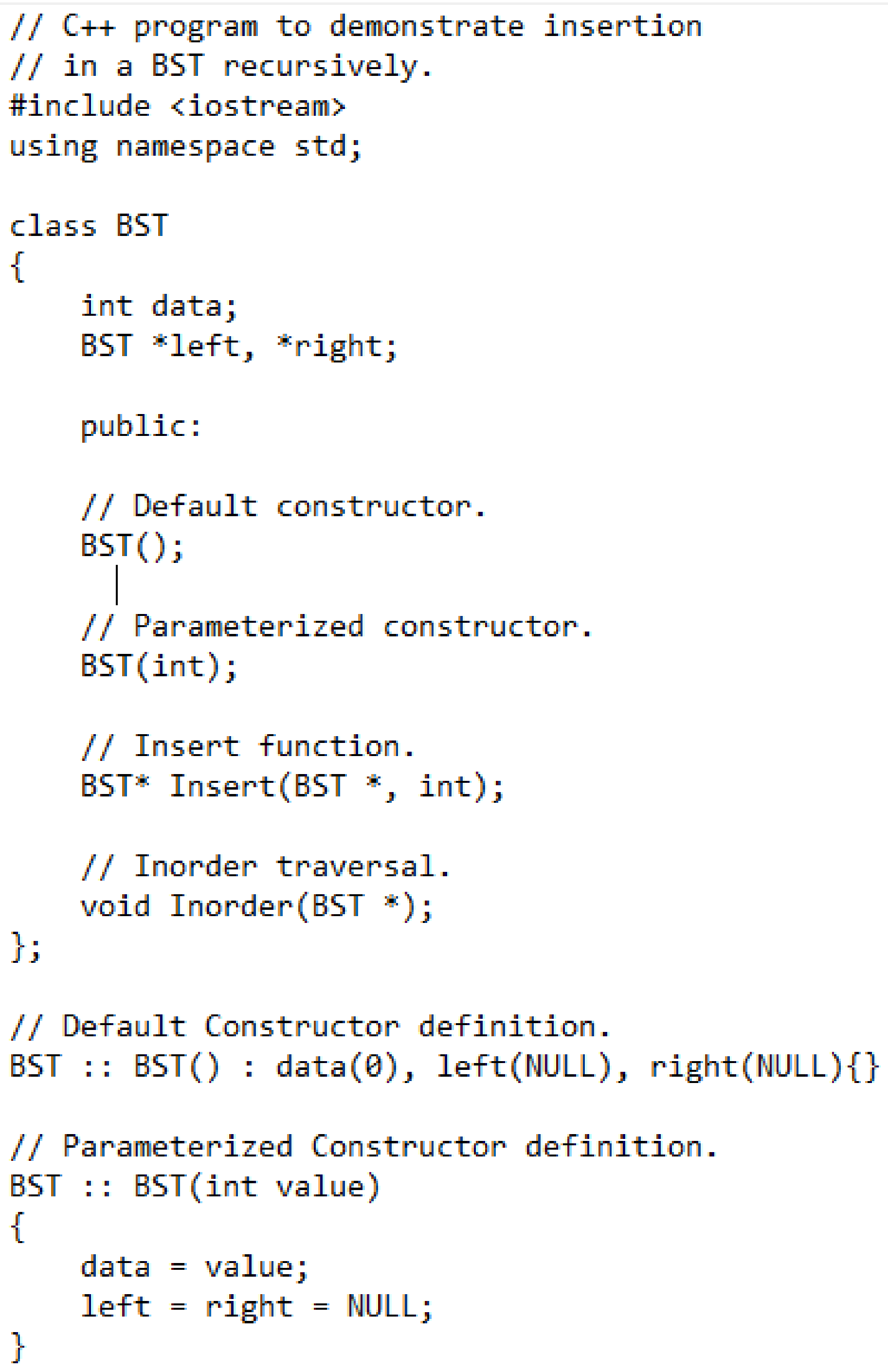
```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Aggggggggg>cd C:\Borland\BCC55\Bin

C:\Borland\BCC55\Bin>bcc32 tree.cpp
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
tree.cpp:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:\Borland\BCC55\Bin>tree
20
30
40
50
60
70
80

C:\Borland\BCC55\Bin>
```



```
// C++ program to demonstrate insertion
// in a BST recursively.
#include <iostream>
using namespace std;

class BST
{
    int data;
    BST *left, *right;

    public:

    // Default constructor.
    BST();

    // Parameterized constructor.
    BST(int);

    // Insert function.
    BST* Insert(BST *, int);

    // Inorder traversal.
    void Inorder(BST *);
};

// Default Constructor definition.
BST :: BST() : data(0), left(NULL), right(NULL){}

// Parameterized Constructor definition.
BST :: BST(int value)
{
    data = value;
    left = right = NULL;
}
```

```
// Insert function definition.
BST* BST :: Insert(BST *root, int value)
{
    if(!root)
    {
        // Insert the first node, if root is NULL.
        return new BST(value);
    }

    // Insert data.
    if(value > root->data)
    {
        // Insert right node data, if the 'value'
        // to be inserted is greater than 'root' node data.

        // Process right nodes.
        root->right = Insert(root->right, value);
    }
    else
    {
        // Insert left node data, if the 'value'
        // to be inserted is greater than 'root' node data.

        // Process left nodes.
        root->left = Insert(root->left, value);
    }

    // Return 'root' node, after insertion.
    return root;
}
```

```cpp
// Inorder traversal function.
// This gives data in sorted order.
void BST :: Inorder(BST *root)
{
    if(!root)
    {
        return;
    }
    Inorder(root->left);
    cout << root->data << endl;
    Inorder(root->right);
}

// Driver code
int main()
{
    BST b, *root = NULL;
    root = b.Insert(root, 50);
    b.Insert(root, 30);
    b.Insert(root, 20);
    b.Insert(root, 40);
    b.Insert(root, 70);
    b.Insert(root, 60);
    b.Insert(root, 80);

    b.Inorder(root);
    return 0;
}
```