Exercise 4 – BST Sort

Demo due by the end of your Week-6 laboratory class. (2 marks)

This exercise is to be completed during your week 5 laboratory class. When you complete the exercise show your work to your lab tutor to have your work marked. The marking is based mainly on correct implementation and code readability. You should implement your code in one file (e.g. ex4.cpp, ex4.c, ex4.java). Make sure your program has a header comment block containing the name of the exercise, your name and your student login (e.g. jfk01). You may implement your solution in C, C++, java or Python.

For this exercise, you are to implement BST sort and test it for correctness. Your program should prompt for the name of an input file and then read and process the data contained in this file.

The file contains a sequence of integer values. Read them and construct a binary search tree from the values in the order they are read. Thus, the first number read will be the root of the tree.
For this exercise, you may use dynamic data, as shown with the pseudo code on page 2.

You do not need to balance the tree as you construct it.
When you have read the last value into the BST, conduct an in-order traversal to output the values in ascending order.

Print the values 10 to a line in a 5-character wide field.
Do not use STL or other libraries to implement the BST.

When you are finished, test your program using the provided text file named "ex4.txt" and show your code and the output to your lab tutor to receive your mark. Also, submit your file via unix (banshee) using the submit command below.

`$ submit –u login –c CSCI203 –a ex4 filename`

where '***login***' is your UNIX login ID and '***filename***' is the name of your file.

If you are unable to attend your lab class and demonstrate your work on time due to circumstances beyond your control (e.g. sickness), contact your lecturer to request an extension.

BST Pseudo Code (from the week 4 lecture notes)

```
type tree_node = record
        contents: stuff
        left: ^tree_node
        right: ^tree_node

root: tree_node


procedure find(value: stuff, node: ^tree_node): ^tree_node
        if value == nil then
                return not found
        fi
        if value == node.contents then
                return node
        else if value < node.contents then
                find(value, node.left)
        else
                find(value, node.right)
        fi
end


procedure insert(value: stuff, node: ^tree_node)
        next: ^tree_node, left: boolean
        if value == node.contents then
                return                                  // already in the tree
        else if value < node.contents then
                next = node.left; left= true            // we need to go left
        else
                next = node.right; left = false                 // we need to go right
        fi
        if next != nil then
                insert (value, next)                    // keep trying
        else
                next = new_tree_node                    // make a new node
                next.contents = value                   // store the value
                if left then                            // update the parent
                        node.left = next
                else
                        node.right = next
                fi
        fi
end


procedure insert_first(value): ^tree_node
        node: ^tree_node
        start = new_tree_node
        start.contents = value
        return start
end


procedure visit(node: ^tree_node)
        if node.left != nil then
                visit(node.left)
        fi
        print(node.contents)
        if node.right != nil then
                visit(node.right)
        fi
        return
end


procedure main()
        root: ^tree_node
        open file & print error if file not found
        read item
        root = insert_first(item)
        for each item in file
                insert(item, root)
        visit(root)
end
```