HOMEWORK #9:

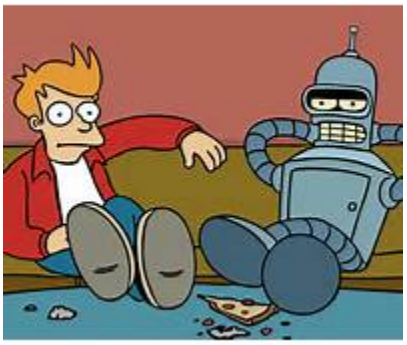
Software Upgrade.

Due Date: Friday, April the 22th, 11:59:59pm

For this assignment, you need to submit a file called '`mybstree.h`' and any other necessary implementation files. Remember to put your **name** and **section** at the top of all your files.

Problem:

Professor Farnsworth has decided to organize his vast collection of trinkets, records, files and doomsday devices that he has gathered from many decades of dedicated research. He wants Bender robotic brain to help him, but for this purpose, he wants Bender to undergo a "software upgrade". Instead of the antiquated vectors and arrays, Prof. Farnsworth wants Bender to use the latest in software technology: Binary Search Trees!.



Not the kind that likes things organized

Your job is to write part of the software upgrade so that Bender knows how to think about collections as Binary Search Trees, so be ready with your recursive tools!

Testing:

Use the provided tester file to check if your implementation is working correctly.

- The program '`treetester.cpp`' uses the '`MyBSTree`' class and the intended output is '`treeoutput.txt`'.

Implementation Requirements:

1. You are expected to derive the '`AbstractBSTree`' class.
2. You are also expected to implement the "**Big-3**" for your '`MyBSTree`' class
3. The functions '`findMin()`' and '`findMax()`' are expected to "`throw`" errors.
4. '`find()`' is **not boolean**, read the description carefully, and see the example.

Useful Hints:

1. Carefully read the comments of each member function.
2. Write down an algorithm for the function before you start coding it.
3. Develop your member functions one at a time, starting from the simplest ones.
   Move to the next function only after the previous one has been tested.
   Trying to code the whole class and then remove the bugs may prove to be too big a task.
4. Use the test functions one at a time.

More Hints:

- You will use **2 classes**, One for the tree nodes, and another as an encapsulation class. That way, you can more easily use recursion and use the `NULL` pointer as your base case.
- In order to use member functions and recursive functions, have the member function call the recursive version, as in the example:

Example:

```
template <typename T>
class TreeNode
{
   T m_data;
   TreeNode* m_right;
   TreeNode* m_left;
};


template <typename T>
class MyBSTree : public AbstractBSTree<T>
{
   TreeNode<T>* m_root;
   int m_size;

   int recursive_foo( TreeNode& t1 )
   {
      … …
      recursive_foo( t2 );
      … …
   }

public:
   int foo()
   {
      recursive_foo( m_root );
   }
}
```

Bonus!:

- The following is the code for “Pretty Printing” a tree:

```
template <typename T>
void prettyPrint (const TreeNode<T>* t, int pad)
{
 string s(pad, ' ');
 if (t == NULL)
     cout << endl;
 else{
   prettyPrint(t->right, pad+4);
   cout << s << t->data << endl;
   prettyPrint(t->left, pad+4);
 }
}
```

- Check this slick recursive function to clone a tree

```
template <typename T>
TreeNode<T>* clone(const TreeNode<T>* t)
{
 if (t == NULL)
     return NULL;
 else{
     return new TreeNode<T>(t->data, clone(t->left),clone(t->right));
 }
}
```

END.