CISP 430 Data Structures
Project 5: Binary Search Tree

We will implement, as a **C++** class, Abstract Data Type Binary Search Tree. We will store the **Record** class that I have supplied in this tree.

Your tree will be a linked structure and will have one private data member, a pointer to a struct, **node**, which will contain a **Record** and the needed pointers. To use my **Record** class, you will need to add the following to your **STRING** class:

```
//Pre: os must be a valid ostream opened (ios::out | ios::binary)
//Post: *this is put into os
//      using the ostream member function write
  void write( ostream& ) const;

//Pre: is must be a valid istream opened (ios::in | ios::binary)
//     the STRING to be read must have been
//     written in the format used by STRING::write
//Post: a STRING is read from is
//      using the istream member function read
//      and assigned to *this
  void read( istream& );
```

These will write a **STRING** to a binary **ostream** and read from a binary **istream**

To make all of this work, you will need to correct **ANY AND ALL** errors that are in your **STRING** class. In addition, I will be supplying the header file and the object code for class **Record**. You must name your **STRING** files **MYSTRING.h** and **MYSTRING.cpp** so that they will work with my **Record** class header file.

BSTree type with the following functionality
PRE: None
POST: An empty **BSTree** is created

```
BSTree ()
```

PRE: None
POST: The **BSTree** is destroyed and all allocated memory is released

```
~BSTree ()
```

PRE: None
POST: Function Return Value == (tree is empty)

```
bool IsEmpty() const
```

PRE: NOT IsEmpty()
POST: Function Return Value == **Record** item with this key. If this key is not in tree, it returns an "empty" **Record**

```
Record Find(const KeyType& key) const
```

PRE: None
POST: The **Record** inserted into BSTree. If the BST already contains this key, it does nothing
left_child < parent ≤ right_child

```
void Insert( const Record& )
```

PRE: NOT IsEmpty()
POST: BSTree is traversed in-order and the function is applied to each node.

```
void InOrderTraversal ( void (*)(const Record&));
```

PRE: NOT IsEmpty()
POST: BSTree is traversed pre-order and the function is applied to each node.

```
void PreOrderTraversal ( void (*)(const Record&));
```

PRE: NOT IsEmpty()
POST: BSTree is traversed post-order and the function is applied to each node.

```
void PostOrderTraversal ( void (*)(const Record&));
```

PRE: NOT IsEmpty()
POST: Function Return Value == depth of the tree

```
unsigned Depth (  ) const;
```

**********Extra Credit**********

PRE: NOT IsEmpty()
POST: **Record** with this key is deleted; the tree is still a BST. If item with key is not in tree, it does nothing

```
void Delete(const KeyType& key);
```

****

You must carefully manage memory for this assignment. You must not use unallocated memory or leave memory allocated when you are done with it.

You are not being asked to provide a test driver, however you must carefully test your classes to ensure that they work correctly. When working on a large project, it is important that naming conventions are understood and followed by the groups responsible for the separate units. For this assignment I will be providing a test driver. It is important that you name your files, class and functions **EXACTLY** the same as those in these specifications so that my test driver can use your code. You **WILL BE** marked-down if you vary from the specifications. (Remember: **C++** is case sensitive!!!!!)

Submit your three files, **BSTree.h**, **BSTree.cpp** to the D2L drop box before the closing date for project #5

Do not change the names of these files. Be sure to save your originals.

DO NOT ZIP THE FILES