COP 4530: Data Structures
Project 5

You are not allowed to use the Internet. You may only consult approved references[*].
This is an individual project.
This policy is strictly enforced.

You must submit a **hard copy** of all of the items requested below. You must also submit your *code*[†] to Canvas.

For full credit, the code that is submitted must:

- Use the specified signature, if applicable.
- Be implemented in a file using the specified file name, if applicable.
- Be correct (i.e., it must always return the correct result).
- Be efficient (i.e., it must use the minimum amount of time and the minimum amount of space necessary to be a correct implementation).
- Be readable and easy to understand. You should include comments to explain when needed, but you should not include excessive comments that makes the code difficult to read.
  - Every class definition should have an accompanying comment that describes what is for and how it should be used.
  - Every function should have declarative comments which describe the purpose, preconditions, and postconditions for the function.
  - In your implementation, you should have comments in tricky, non-obvious, interesting, or important parts of your code.
  - Pay attention to punctuation, spelling, and grammar.
- Follows ALL coding guidelines from section 1.3 of the textbook. Additional coding guidelines:
  - No magic numbers. Use constants in place of hard-coded numbers. Names of constants must be descriptive.
  - No line of the text of your source code file may have more than 80 characters (including whitespace).
  - All header files should have #define guards to prevent multiple file inclusion. The form of the symbol name should be `<FILENAME>_H_`
  - Do not copy and paste code. If you need to reuse a section of code, then write a function that performs that code.
  - Define functions inline only when they are simple and small, say, 5 lines or less
  - Function names, variable names, and filenames must be descriptive. Avoid abbreviation.
  - Use only spaces (no tabs), and indent 3 spaces at a time.
- Compile and run on the C4 Linux Lab machines (g++ compiler, version 4.8.2). *The shell script and makefile that I will use to compile and run your code will be posted on Canvas. Please note that I may use my own `main.cpp` file to test the code you submit.*
- Have no memory leaks.

[*]The list of approved references is posted on Canvas. You must cite all references used.
[†]Your code must compile and run on the C4 Linux Lab machines

Project Tasks

Implement functions to complete the Binary Search Tree Class Definition in `bst.h`:

```
template <class T>
class BinarySearchTree {
   private:
      class Node {
         public:
            T data;
            Node * left;
            Node * right;
            Node * parent;

            Node():left(NULL),right(NULL),parent(NULL) {};
            Node(const T& item) {
               data=item;
               left=NULL;
               right=NULL;
               parent=NULL;
            };
      };
   public:
      BinarySearchTree();
      BinarySearchTree(BinarySearchTree&);

      ~BinarySearchTree();

      bool isEmpty() const;
      bool search(const T&) const;

      T getSuccessor(const T&) const;
      T getPredecessor(const T&) const;
      T getMinimum() const;
      T getMaximum() const;
      int getHeight() const;
      int getSize() const;

      void inorder(std::ostream&) const;
      void postorder(std::ostream&) const;
      void preorder(std::ostream&) const;

      bool insert(const T&);
      bool remove(const T&);

      void displayGraphic(std::ostream&) const;

      BinarySearchTree& operator=(const BinarySearchTree& rhs);
   private:
      Node * _root;

      // Additional private functions implemented as needed
};
```

To earn full credit you are not permitted to use a size member variable.

1. Implement the following functions (and their associated helper functions, if applicable) in `bst.h`. *Purpose/Preconditions/Postcondition for these functions are provided in* `bst.h`
   (a) [10 points] `T getMinimum() const` using a helper function: `void getMinimumHelper(Node *, Node * &)`
   Note that the helper function is also used by other provided functions in the class.
   (b) [10 points] `T getMaximum() const` using a helper function: `void getMaximumHelper(Node *, Node * &)`
   Note that the helper function is also used by other provided functions in the class.
   (c) [5 points] `void inorder(std::ostream&) const` using a helper function:
   `void inorderHelper(std::ostream&, Node *) const`
   (d) [5 points] `void preorder(std::ostream&) const` using a helper function:
   `void preorderHelper(std::ostream&, Node *) const`
   (e) [5 points] `void postorder(std::ostream&) const` using a helper function:
   `void postorderHelper(std::ostream&, Node *) const`
   (f) [5 points] `bool search(const T&) const;`. Note that you may NOT use any of the provided helper functions to implement your search function. The helper functions are only provided for the insert/delete functions.
   (g) [10 points] `int getHeight() const;`
   (h) [10 points] `int getSize() const;`
   (i) [15 points] `void deleteBinarySearchTree(Node * &subtreeRoot);` which is used by the destructor function.
   To delete a binary search tree rooted `subTreeRoot` *you must first delete its left and right subtree and delete the root itself.*
   (j) [15 points] `void copyBinarySearchTree(Node * original, Node * & copy);` which is used by the copy constructor and assignment operator.
   To copy a binary a binary search tree you must first create a root node, copying the data from the original tree. Then recursively copy the left and right subtrees. Don't forget to update the parent pointers for each node.

2. [10 points] Write a main function to *completely and thoroughly* test all of the functions included in the Binary Search Tree class (including all of the provided functions). Make sure you test for binary search trees that hold different types of objects.
   Note that I will be creating my own `main.cpp` *file to test your code from Task 1 above.*