Implement the bag template class from Section 10.5, using a binary search tree to store the items.

[HW10.docxPreview the document](https://wvm.instructure.com/courses/37898/files/3805069/download?wrap=1)

You'll need the following files: [bintree.h](https://wvm.instructure.com/courses/37898/files/3805070/download?wrap=1" \o "bintree.h)[Preview the document](https://wvm.instructure.com/courses/37898/files/3805070/download?wrap=1) [bintree.template](https://wvm.instructure.com/courses/37898/files/3805068/download?wrap=1" \o "bintree.template)[Preview the document](https://wvm.instructure.com/courses/37898/files/3805068/download?wrap=1) [bag6.hPreview the document](https://wvm.instructure.com/courses/37898/files/3805072/download?wrap=1) [bag6.templatePreview the document](https://wvm.instructure.com/courses/37898/files/3805073/download?wrap=1) [bagtest.cxxPreview the document](https://wvm.instructure.com/courses/37898/files/3805074/download?wrap=1) [bagexam.cxxPreview the document](https://wvm.instructure.com/courses/37898/files/3805075/download?wrap=1)

You'll implement 4 functions:

1. template <class Item>  
void bst\_remove\_max(binary\_tree\_node<Item>\*& root\_ptr, Item& removed)

2. template <class Item>  
typename bag<Item>::size\_type bst\_remove\_all  
(binary\_tree\_node<Item>\*& root\_ptr, const Item& target)

3. template <class Item>  
void bag<Item>::insert(const Item& entry)

4. template <class Item>  
typename bag<Item>::size\_type bag<Item>::count(const Item& target) const

update:

Some students have discovered that the bag6.h is missing its definition for its no-argument constructor, so you'll see a linker error. Please provide a body to the Bag's no arg constructor like this bag6.h:

bag( ) { root\_ptr = NULL; }

**The Assignment:**

Implement the bag template class from Section 10.5, using a binary search tree to store the items.

**Purposes:**

Ensure that you understand and can use binary search tree.

**Before Starting:**

Read all of Chapter 10, especially Sections 10.3 and 10.5.

**Files that you must write:**

1. bag6.h: Header file for this version of the bag class. You don't have to write much of this file. Just copy our version from [www.cs.colorado.edu/~main/chapter10/bag6.h](http://www.cs.colorado.edu/~main/chapter10/bag6.h) and add your name and other information at the top.
2. bag6.template: The implementation file for the new bag class. I have written much of this to get you started. You can download my starting file from[www.cs.colorado.edu/~main/chapter10/bag6.template](http://www.cs.colorado.edu/~main/chapter10/bag6.template)  
   There are four functions in this implementation file that you must implement. These files are marked with the words STUDENT WORK.

**Other files that you may find helpful:**

1. bintree.h: and bintree.template This is the binary tree node template class from Section 10.3. You can download them from [www.cs.colorado.edu/~main/chapter10/bintree.h](http://www.cs.colorado.edu/~main/chapter10/bintree.h) and[www.cs.colorado.edu/~main/chapter10/bintree.template](http://www.cs.colorado.edu/~main/chapter10/bintree.template)

**NOTE:** This version of the binary tree node has a small change from the original version that appears in the first printing of the second edition of the textbook. In particular, I have changed the return values from the non-const versions of the left() and right() functions so that they return a reference to the pointer in the node. This is indicated by the & symbol here:

binary\_tree\_node\*& left( )

The use of a "reference" (indicated by the ampersand) in the return value has two advantages that simplify the material of Chapter 10:

* + It now allows a direct assignment such as: p->left() = NULL. This is not a huge advantage since the same thing can be accomplished by using the set\_left function.
  + The expression p->left() can be passed as the argument to a function such as: tree\_clear(p->left()); The parameter of tree\_clear is a reference parameter, so that any changes that tree\_clear makes to p->left() will now affect the actual left pointer in the node \*p. In this example, the tree\_clear function does set its parameter to NULL, so that the total effect of tree\_clear(p->left()) is to clear the left subtree of p and to set p's left pointer to NULL.

In the case of tree\_clear, this is not a huge advantage because we could have just set p's left pointer to NULL ourselves. But, in this assignment, there are two functions, bst\_remove and bst\_remove\_max, which are easier to write if we can use p->left() and p->right() as the parameters of recursive calls. See my implementations in bag6.template for details.

1. [bagtest.cxx:](http://www.cs.colorado.edu/~main/chapter10/bagtest.cxx) A simple interactive test program.
2. [bagexam.cxx:](http://www.cs.colorado.edu/~main/chapter10/bagexam.cxx) A non-interactive test program that will be used to grade the correctness of your bag class.

**The Bag Class Using a Binary Search Tree   
Discussion of the Assignment**

Start by understanding the entire pseudocode for the binary search tree operations (from Section 10.5). Then read through the portions that I have already implemented for you. Implement the rest of your work in two parts: (1) The insert and count functions, and (2) The bst\_remove\_all and bst\_remove\_max functions. Don't move to step 2 until you have completely finished and tested step 1.

Since this is a template class, debugging can be more difficult (some debuggers don't permit breakpoints in a template function. To help in debugging, you can call b.debug() in a program to print the binary search tree for the bag b (using the format shown on page 484).

The bag and bintree templates are never compiled on their own, but in order to create bagexam.o, all the template files must be present in the current directory.