Assignment 10

Comparing Trees and Hash Tables

Runnable Code #1 Due:	11:00 PM,	Friday, April 25, 2003
`README` Due:	11:00 PM,	Tuesday, April 29, 2003
Runnable Code #2 Due:	11:00 PM,	Wednesday, April 30, 2003
Final Code Due:	11:00 PM,	Thursday, May 1, 2003

1 Scenario

Two popular ways to provide maps are trees and hash tables. In the last assignment, you examined hash tables, but you are left wondering whether binary search trees would be a more elegant solution—after all, you don't have to waste time rehashing with binary search trees. You also wonder whether your implementations of trees and hash tables are as efficient as those provided by the STL (and in the case of hash tables, SGI's STL extensions). You don't know yet, but you are determined to find out...

2 High Level Interface and Behavior

In this assignment, you will use the `myspell` program that you wrote for the last assignment as the basis for developing a benchmark to test tree and hash-table implementations. Your program will accept the same arguments as it did in the previous assignment and behave in almost exactly the same way. In addition to the `-d` option described in the last assignment, the program will also accept the following additional options to control the data type that is used to represent all dictionaries:

`-h` Use *HashSet<string>*

`-t` Use *TreeSet<string>*

`-T` Use *StlSet<string>*

`-H` Use *SgiHashSet<string>*

(See the next section for a discussion of these types.) If none of the above options are supplied, you are free to pick whichever representation you prefer as the default.

The output of the program's `-d` option will vary depending on which of the options above is chosen. If `-d` is specified with either the `-H` or `-T` options, the program should simply print

```
No statistics available.
```

in lieu of statistical output. If `-h` is specified, the statistics should be the same as in the previous assignment. For the `-t` option, you should print a single line containing useful information about the structure of the tree, including its height. It is up to you to decide what additional information to print and how to calculate that information. As usual, you should describe your choices in your `README` file.

If the program is given invalid arguments, it should produce appropriate error messages rather than crash. The exact error message displayed is up to you.

3 Required Classes and Functions

This assignment requires you to provide an abstract base class *AbstractSet*, and four derived classes, *HashSet*, *TreeSet*, *StlSet*, and *SgiHashSet*. The core of your spell checker (i.e., the part that reads the dictionary, checks words, etc.) should only use only the *AbstractSet* interface. In particular, the core spell-checking part of the program may not have different execution paths (via **if** or **switch** statements) for different dictionary representations.

3.1 Required *AbstractSet* Abstract Template Class

Your code must provide and use the abstract template class *AbstractSet*. Note that you cannot create abstract classes, so you can only have pointers or references to, say, an *AbstractSet<string>*.

An instance of *AbstractSet*, on some type *T* (i.e., *AbstractSet<T>*), will specify the following operations:

- A destructor that leaves the object in a sane (empty) state
- A member function *size_t* size() ***const*** that returns the size of the set
- A member function *void* insert(***const*** *T&*), where h.insert(x) adds x to the set, where the function's behavior is undefined if x has already been added to the table
- A member function *bool* exists(***const*** *T&*) ***const***, where h.exists(x) returns true if x is present in the set and false otherwise

(The type *T* is used for illustrative purposes—you may not use such a short name for the template type argument in your template class.)

For some type *T*, an *AbstractSet<T>* is an *abstract class*, which is analogous to an *interface* in Java. With the exception of the destructor, *all* of the above functions must be declared as abstract (and thus virtual) and have no implementation code. The destructor should also be declared virtual, but will need an implementation (that does nothing). As a special concession, you may provide an inside-the-class (Java-style)

implementation of the destructor, but if you do so, you should nevertheless follow good indentation style.[1]

For any class that conforms to the *AbstractSet* interface, the functions size, insert, and exists must execute in $O(\log n)$ expected amortized time, where n is the number of items in the set. In some cases, this bound may be loose.

You may add other (abstract) operations to this class if you wish (e.g., require a printStatistics member function).

You must write your code such that `#include "abstractset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

3.2 Required *HashSet* Template Class

This class will be almost exactly the same as your *HashSet* class from the previous assignment, with one change: the class must now inherit from *AbstractSet* (i.e., for any *T*, *HashSet<T>* will be substituable for *AbstractSet<T>*).

You must write your code such that `#include "hashset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

You should have both *AbstractSet* and *HashSet* written and working by the first runnable code deadline.

3.3 Required *TreeSet* Template Class

This class will represent a set using a binary search tree. Your *TreeSet* template class must inherit from *AbstractSet*, and must satisfy the complexity guarantees given in the specification for *AbstractSet*.

An instance of *TreeSet*, on some type *T* (i.e., *TreeSet <T>*), will specify the following additional operations, beyond the *AbstractSet* interface:

- A member function *ostream&* print(*ostream&*) that recursively prints the tree according to the following algorithm:

```
to print a (sub)tree:
    print "("
    print left subtree
    print ", "
    print value stored at root of this (sub)tree
    print ", "
    print right subtree
    print ")"
```

1. We allow this exemption from the usual rules of CS 70 (which prohibit Java-style implementations of classes) in this case because the class is abstract, there is only one function to implement, it does nothing, and defining it inside the class declaration makes it perform exactly like the compiler's default code (which vanishes when we write a declaration for the destructor, which in turn we had to do to make the destructor virtual—there is no way to say tell the compiler that its default code for the destructor is okay, but that the destructor should be virtual).

- A member function *int* `height()` that returns the height of the tree

(Each of these member functions may take $O(n)$ time.)

The choice of binary-search-tree technique is up to you, provided you meet the complexity guarantees for *AbstractSet*—we strongly recommend you choose a technique you understand and can code elegantly.[2] In particular, you cannot make any assumptions about the word order for words in the dictionary; you can only assume that each dictionary word occurs only once.

You may assume that all types used in instances of this class will support the `<` operation, much as your hash-table code assumed the existence of a `myhash` function.

You must write your code such that `#include "treeset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

Finally, remember that you *may not* copy anyone else's code for your *TreeSet* class. In this case, you may not even take code from your textbook. You are allowed to read Weiss for ideas and use his descriptions to guide your implementation but you may not directly transcribe any code from the textbook.

You should also have at least some of the code for `TreeSet` written by the first runnable code deadline.

3.4 Required *StlSet* Template Class

This class will represent a set using the STL's *set* template class. In essence, it will be a fairly trivial wrapper around a *set* object, adapting the interface of *set* to the interface required by *AbstractSet*. As with the other classes, it must inherit from *AbstractSet*.

You must write your code such that `#include "stlset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

3.5 Required *SgiHashSet* Template Class

This class will represent a set using SGI's *hash_set* extension to the STL (provided in the `__gnu_cxx` namespace and available using `#include <ext/hash_set>`). In essence, it will be a fairly trivial wrapper around a *hash_set* object, adapting the interface of *hash_set* to the interface required by *AbstractSet*. As with the other classes, it must inherit from *AbstractSet*.

Annoyingly, SGI *hash_set* class does not automatically provide hashing for C++ *string*s, nor does it automatically use a function called `myhash` like the one you have written for your own hash table implementation. Section 5 has code to make the SGI *hash_set* class use *myhash*. You should use this code, or an equivalent technique.

2. We will consider nonrecursive code inelegant. Similarly, we consider trees with backwards links from children to parents inelegant.

You must write your code such that `#include "sgihashset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

4 Describing Performance

Your `README` file should contain some description of the performance of the different techniques, in addition to the usual material for a CS 70 `README`. Once you have all your code working, you may wish to compile your code with optimization to give a more accurate comparison. You should also compare performance with your code from Assignment 9 to see whether using inheritance/subtyping had a noticeable performance cost. Exactly how you compare the techniques is up to you.

5 Tricky Stuff

As usual, there are parts of this assignment that are tricky. Here are a few tips:

- This assignment will probably be the first time you have written any code that uses inheritance, pure virtual functions, and related concepts. Be sure allow yourself some time to get familiar with coding in this way.
- You cannot create an object belonging (only) to an abstract class. If you write

  ```
  AbstactSet<string> myDictionary;
  ```

  your code will not compile because you cannot make such an object. You can, however, make objects of type *HashSet<string>*, *TreeSet<string>*, *StlSet<string>*, and *SgiHashSet<string>*, each of which can also be seen as an *AbstactSet<string>*. You can have a variable of type *AbstactSet<string>** or *AbstactSet<string>&*. Thus, you could, for example, create a *HashSet<Foo>* object and pass it to a function that expects a reference to an *AbstractSet<Foo>*.
- You *must* use subtyping (i.e., *AbstractSet*) to provide the switchable functionality, not genericity (templates). In other words, if you created a *SpellChecker* class for the last assignment, you should *not* make that class into a class template.
- Remember that *AbstractSet* is a class template, not a class. When you use it, you need to provide a type argument, such as *AbstractSet<int>*. This requirement applies even when you are specifying an inheritance relationship for a derived class (or a template for a class whose instances will be derived classes).
- You are allowed to have more than one abstract class in your program. For example, some students may wish to have a class *AbstractSetFactory<T>* that has a method `newAbstractSet()` that returns a pointer (or perhaps an *auto_pointer*) pointing to an *AbstractSet<T>* (the object itself will come from a class derived from *AbstractSet<T>*).

- SGI's *hash_set* class is described at http://www.sgi.com/tech/stl/hash_set.html.
- To make an instance of the SGI *hash_set* class that uses the myhash hashing function, you must provide a hashing functor. The easiest way to do so is to put the following private nested class (commented appropriately) inside your *SgiHashSet* class (assuming that your template type parameter is called *Value*),

```
class MyHash {
public:
    size_t operator()(const Value& value) const
    {
        return myhash(value);
    }
};
```

  and then declare your hash-table data member as a *hash_set*<*Value,MyHash*>, instead of declaring it as a *hash_set*<*Value*>. (Normally, we don't declare and define a class at the same time, but we'll allow it in this case—both because it is private, and because it is a trivial functor.)