CS4115 Week12 Lab Exercise

Lab Objective: For my money one of the most useful data structures when programming is the *map*. Also called an *associative array*, most major programming languages support this in some form, or other. It is part of the STL in C++ and it exists in Perl, too. Java, in fact, has two classes that implement the `map` interface, `TreeMap` and `HashMap`. It is maps implemented as trees that we are going to consider now, but not in Java.

The idea of a map is that it allows us to save some chunk of information by associating a *key* with it. How it is implemented is something we will get to later but for now it allows us to set up an array that is indexable by something other than an integer index. In effect, we can write a line of C++ such as:

```
telNum["Frank"] = "+353 61 212 469";
```

The array `telNum` *associates* "+353 61 212 469" with "Frank".

The goal of this lab is to compare two implementations of a map by measuring the time it takes to complete a sequence of *insert* and *find* operations. The lab will be spread over two weeks. For this week you should set the goal of writing two programs that each insert a random stream of `int`s into a map.

Here's a summary of the tasks involved

❶ Read the background of *map*s and their near neighbour the *set*;

❷ Write a program `map-ins` that reads a stream of integers from standard input, `cin`, and inserts them into a map data structure;

❸ Write the corresponding program `avl-ins` that uses an AVL tree instead of an STL map.

In Detail

❶ Wikipedia to the rescue again. To swot up on the general area please take a few minutes to consult the Wikipedia page on associative arrays here and, in particular on the STL implementation, the map. There is very useful code provided on the latter page.

Although it is not mentioned on that page (it just refers to the implementation as a "self-balancing binary search tree") it appears that the STL implementation is *not* an AVL tree, but rather a *red-black* tree (Wikipedia), another self-balancing BST.

❷ Based on the code you have seen on the previous pages you should be able to write a program based on the STL that reads from standard input until exhaustion a stream of integers (possibly negative) and inserts them into a map, while maintaining a count of the number of times each is seen. What you will need here is to make an association between

an integer (as read from stdin) and another integer that is the count of the occurrences of the integer read in.

Call this program `map-ins`.

I hope you use a debugger when you code. While it may seem like a poor use of time the opposite, in fact, is the case. It can save you loads of time in fixing a program. The debugger I use is `gdb`. One of the things I disliked about the STL was that it was impossible to inspect the internals of an STL "object" within the debugger. Recently I discovered a set of macros that help in this regard. Please consider using them, or some other method of accessing the internals of STL from a debugger.

❸ You should now write a corresponding program, `avl-ins`, that performs identically but this time using an AVL tree as the underlying data structure. But fear not, I am not asking you to implement an AVL class in C++ . A particularly nice one exists here, coming as it does, with sample usage. This zip file is actually available locally at `~cs4115/labs/week12`.