CS4115 Week13 Lab Exercise

Lab Objective: Last week you had the task of writing two small programs that read a stream of `ints` from `stdin` (the keyboard) and inserted them in to a *map* container.

This week you should modify your programs slightly so that you maintain a frequency count of the integers read. You should time how long the operations take in each program. At the end of your program you should write out the frequency table and, finally (and trivially), return an average "access" time over all the inputs you were given.

Here's a summary of the tasks involved

❶ Copy last week's programs `map-ins.cc` and `avl-ins.cc` into a newly created directory for Week13;

❷ Modify the programs so that as each integer is read its frequency count is either set to 1 or incremented and save the programs as `map-ops.cc` and `avl-ops.cc`, respectively;

❸ Test your programs to ensure they give the same output for the same input stream;

❹ `handin` deadline end of next week, Week14

In Detail

❶ Create a new directory for this week's lab, `week13` and copy over the two file you have been working on, renaming them, respectively, to be `map-ops.cc` and `avl-ops.cc`.

❷ You will notice that there is a program called `gen-ints` in the class directory for this week's lab, `~cs4115/labs/week13`. This program, when given a count parameter, will generate that count of random numbers in the range of $[-10,000, 10,000]$. You should feed both of your programs with this output. So, for example, the following should test your AVL map program on 2000000 randomly generated integers in the above range:

```
gen-ints -n 20000000 | map-ops
```

(Make sure you have edited your `~/.bash_profile` so that `.` (the current directory) appears as part of your `PATH` variable. If you have to edit this file today you will need to also make the new settings stick with

```
sh ~/.bash_profile.)
```

For each of the two programs there are two tasks. You should

- maintain a frequency count of the number of occurrences of each integer you encounter and you should time how long it takes to perform the stream of insertions / updates on the map. Code for maintaining a clock in your program is given below.

- to verify the integrity of your table you should write it out to *standard output*, `cout` to C++ programmers. This should be the *last* thing your program and it should not be part of the timing.

The output of your program should conform exactly to the output of the sample program `map-ops`, also in the class directory.

In order to verify the correctness of your program you may wish to generate some random numbers and save them to a file. Then you can test the outputs of your two programs for equality. Here's how you would do that:

```
gen-ints -n 5000 > rand-ints
avl-ops < rand-ints > avl-results
map-ops < rand-ints > map-results
diff avl-results map-results
```

Of course, you may want to compare your output to my output as well to be sure to be sure.

Timing a section of code We have seen how you can use the time utility to time the running of an *entire* program. But what happens when you just want to time part of it, as given here? Here's how this can be achieved:

```
clock_t t1, t2;
t1 = clock();    // read clock
  : // do some work
t2 = clock();    // read clock again
float s = (t2-t1)/float(CLOCKS_PER_SEC);
printf("Total time taken = %.3f(s)\n", s);
```

Note that this is not all the information that should be printed out to the screen. See a sample run of `map-ops` for the definitive output format. The time per op. is given in micro-seconds (abbreviated as "us", seeing as I can't encode μ, the correct Greek letter, in a `printf()` statement :-).)

Make sure that you only time the insertions / updates.

Which program ran fastest for the set of insertions / updates you gave it?

❹ By the end of Week14 I will want you to `handin` your two source files `map-ops.cc` and `avl-ops.cc`. I will compile them and test them on saome randomly generated data. I will also inspect them to make sure that you're not doing anything funny like using an ordinary array (indexed by the inetger you've just read) instead of the associative array.

Also, I am aware that at this time of the semester judgement gets a little frayed, with so many other deadlines and other things to do. Please do not be tempted to take the shortcut of using somebody else's code. This is called cheating and there are very clear penalties in the Student Handbook for this most serious offence. There is a very good plagiarism detector called `moss` in existence and I will be examining your programs for similarities. I am all in favour of you discussing your coding strategy with your friends but when it comes to writing code it must be of your doing.

Good luck.