COMP11 Lab 1: What the Diff?

In this lab you will learn how to write programs that will pass our grading software with flying colors. You'll also be introduced to some new methods of manipulating input and output via the command line. Specifically, the purpose of this lab is to:

- Practice using the `Terminal` to perform input/output redirections
- Learn how to use the `diff` program to mimic our homework grading process
- Practice generating ground truth files to `diff` against
- Learn about decider programs
- Practice using strings, I/O, and conditionals (if and/or if-else statements)
- Become familiar with an important COMP11 mantra: compile-`diff`-submit

As with the first lab, you should navigate into your COMP11 directory and download the starter code for this lab:

- ▶ `cd ~` (takes you to your home directory)
- ▶ `cd comp11` (enter your 'comp11' directory)
- ▶ `pull-code11 lab01` (download the 'lab01' directory here)
- ▶ `cd lab01` (go into the downloaded directory)
- ▶ `ls` (see what files we've provided for you)

You Do Like We Do (15 mins)

Starting with HW1, we will be testing and grading your programs automatically. There is nothing magical about this process. We compile your program, run it on a preselected input, and check to see if the resulting output matches our expectation. This is repeated until we're convinced that your solution is fully functional. In this lab we will walk you through that process so that you can test your programs in the same way.

While you were completing HW0, we were completing it also. You can run our solution, `encrypt_hw0`, to create the 10-shift cipher of the word "`tufts`" with the following command (recall that you must first enter "`use comp11m1`" if you have not added this command to your `.cshrc` profile):

- ▶ `encrypt_hw0`

But what if you wanted to do that again? What if you wanted to do it 100 more times? Having to type "`tufts`" and "`10`" during every single execution would get old fast. It would be great if you could automate the process of entering that input. Well good news: you can. It's called "redirecting."

The basic concept here is simple: instead of manually entering input, you can save that input into a separate file and "trick" your program into reading that file as if the user were entering its contents. The program still thinks it's reading manually entered input, but we have "redirected input" from a file instead. We have created such an input file for you called `encrypt_test.in`. You can view it by entering:

- ▶ `cat encrypt_test.in`

Hopefully nothing shocking to see there - just the word "`tufts`" and the number "`10`", each followed by an `enter` keystroke. Now let's run `encrypt_hw0` again, but instead of typing this input ourselves, we'll tell the program to read from `encrypt_test.in`. The redirection looks like this:

- ▶ `encrypt_hw0 < encrypt_test.in`

The resulting output should look the same as it did before, just without the keystrokes associated with the user typing "`tufts`" and the number "10" and hitting `enter`. This is because that input is coming from `encrypt_test.in`, not the user.

For our next trick, we will do the same thing, but with our program's output. That is, rather than have the program print its output to the `Terminal` console, we will specify a separate file and redirect the program's output into it. Enter the following command:

- ▶ `encrypt_hw0 > encrypt_test.out`

It would appear that nothing has happened, but wait! Type "`tufts`" and hit `enter`, then type "`10`" and hit `enter`. The program terminates, but otherwise nothing of note. But if you enter `ls` you will notice that there is a new file in the directory called `encrypt_test.out`. Take a look with:

- ▶ `cat encrypt_test.out`

The contents of `encrypt_test.out` should look identical to the output you saw in the console the last time you ran the program. That is, all of the program's output, not including the user-entered keystrokes, has been written to `encrypt_test.out`.

So now we've redirected the contents of a file into the program, and we've redirected the program's output into a separate file. Now let's put it all together and do both of these things in a single command. First, delete `encrypt_test.out`. Then, enter the following command:

- ▶ `encrypt_hw0 < encrypt_test.in > encrypt_test.out`

The file `encrypt_test.out` will be recreated, and will contain the output that `encrypt_hw0` produces when it is run with the input contained in `encrypt_test.in`. In general, if you want the input and output of some executable called `program` to be redirected from files `input` and `output`, respectively, the command is `program < input > output`.

Now, the final piece. Suppose that we're not sure if `encrypt_hw0` is producing the correct output, but we have access to a demo program that we *know* is correct. We can use that demo program to create what is known as a "ground truth" output file, and compare it to the output that `encrypt_hw0` produces. We have provided such a demo program called `encrypt_demo`, and Linux provides a program called `diff` that can perform this comparison. This is accomplished with the following commands:

- ▶ `encrypt_demo < encrypt_test.in > encrypt_test.gt` (create a ground truth file from the demo)
- ▶ `diff encrypt_test.gt encrypt_test.out` (compare it to the output of `encrypt_hw0`)

If `diff` prints nothing to the console, these two files are identical. If `diff` produces any output at all, the files are not identical, which means that `encrypt_hw0` is not producing the correct output. To see this, we've provided an alternate version of `encrypt_hw0` called `encrypt_alt`. You can test its output in the same way with the following commands:

- ▶ `encrypt_alt < encrypt_test.in > encrypt_test.alt`

- `diff encrypt_test.gt encrypt_test.alt`

`diff` can spot the difference, can you?

For the bulk of your assignments we will provide you with both a demo program and test input(s). This allows you to test your programs in exactly the same way that they will be tested for grading. You can also create your own input files for testing and use the demo program to generate the ground truths for those inputs as well. Using `diff` to check your programs should become a routine part of your testing process; if you fail to check your program's output with `diff`, it is *highly* likely that you will fail many of our tests and lose significant points for an assignment. To prevent this, you should always ensure that your program (1) compiles and (2) is passing `diff` before you submit it.

The Decider (60 mins)

In computer science, a "decider" is a program that determines whether or not an input string has a certain, pre-determined property. For example, you could build a decider that checks whether an input string begins with a capital letter, or whether it has 8 characters in it, or whether it is equal to the string "`zalzy`".

In this part of the lab, you will build a program that decides whether an input string ends in the suffix "`ing`". If it does, your program will print an enthusiastic confirmation. If it does not, it will sadly and crudely append "`-ing`" to whatever word was submitted. We have provided an example of this program called `ing_demo`, which you can run by simply entering the command:

- `ing_demo`

Your goal is to exactly duplicate the functionality of `ing_demo`. In order to check your work, we have provided two input files, `ing_test1.in` and `ing_test2.in`, which can be used to generate ground truth outputs from the demo. When your program is working, you should be able to run the following commands:

- `g++ -o my_ing_exe -Wall -Wextra my_ing.cpp` (compile your program)
- `./my_ing_exe < ing_test1.in > ing_test1.out` (run your program to get a test output file)
- `ing_demo < ing_test1.in > ing_test1.gt` (run the demo program to get a ground truth file)
- `diff ing_test1.gt ing_test1.out` (compare your output to the ground truth)
- `./my_ing_exe < ing_test2.in > ing_test2.out` (do the same thing with a different input file)
- `ing_demo < ing_test2.in > ing_test2.gt`
- `diff ing_test2.gt ing_test2.out`

If your program is working correctly, both of these `diff` commands should terminate without producing any output. To get you started, we have provided you with boilerplate starter code in `my_ing.cpp`. You may also find it useful to discuss the following questions with your partner:

- What aspects of `ing_demo` and `encrypt_demo` are similar? What aspects are different?
- Given a word, how can your decider program access the characters that you care about?

When you are done (does your code compile and pass `diff`?), submit your work using the following command:

- `submit11-lab01`