COMP11 HW0: Holy Shift

The bulk of this assignment pertains to a variant of the `encrypt.cpp` code that we went over in class, which you will download as part of this assignment. Note that this file (when downloaded) will have WAY more comments in it than we expect from your programs; they're there to help guide you through this assignment's starter code.

Learning Objectives

This homework is designed to give you practice with the following skills:

- How to make copies of files with Linux terminal commands
- How to modify and sync files between the Halligan computers and your personal machine
- How to follow the code-compile-execute cycle of programming for this course
- How to teach yourself about unfamiliar concepts
- How to reason about programming problems

Pre-Section: Survey (5 Points)

Please fill out this short survey to give us a better idea of your computing background. This will help us tailor the course to better suit all of your needs!

Section I: Set-up

The first order of business is to download the files that you'll need for this assignment. Assuming that you are working on your personal computer (and not a Halligan lab computer), use either your `Terminal` program (Mac users) or `puTTY` (PC users) to log into Halligan, just as you did in lab (refer to the lab spec if you cannot remember the steps). Once you are logged in, enter the following commands:

- ► `use comp11` (if you have not yet added this directive to your `.cshrc` profile)
- ► `cd comp11` (this moves you to the `comp11` sub-directory in your home directory)
- ► `pull-code11 hw00` (this downloads the hw00 code into your current directory)
- ► `cd hw00` (this changes to the `hw00` sub-directory you just downloaded)
- ► `ls` (this displays all the files in your `hw00` directory)

The last command above will show that you now have a file called `encrypt.cpp` in your directory. This is a C++ program (since it contains C++ code), but it cannot be run directly on a computer. Instead, you must first *compile* your program into an "executable" file. So we'll start by compiling `encrypt.cpp` and running the resulting executable file, just to make sure that everything is in order. Enter the following commands:

- ► `g++ -o encrypt -Wall -Wextra encrypt.cpp` (compiles your program)
- ► `./encrypt` (runs the resulting executable)

That compile command was a mouthful (don't forget that you can get back to it by hitting the up arrow on your keyboard). Let's break it down piece by piece:

`g++`: This is the name of the program that we're using to compile the program. We're invoking it in the same way that we're invoking the `pull-code11` program (above) and the `submit11` program (below).

`-o encrypt`: The `-o` "flag" indicates that we would like to name our compiled program (i.e., the resulting executable file) something specific, and that specific name is the following word. Here, we are naming our compiled program `encrypt`.

`-Wall -Wextra`: These flags tell the `g++` program that, if our code has an error in it, we want the most detailed error messages available.

`encrypt.cpp`: This is the name of the file that we'd like to compile. If your program consists of multiple files, they must all be listed.

In the next section of this assignment, you will perform three experiments on `encrypt.cpp`. However, to ensure that you don't damage the original file (i.e., make changes that you can't figure out how to do undo), you'll do the experiments on *copies* of `encrypt.cpp`. In general, making a backup copy before you begin a substantial modification is a good practice. Make three copies of `encrypt.cpp` as follows:

```
▶ cp encrypt.cpp my_encrypt.cpp
▶ cp encrypt.cpp broke_encrypt.cpp
▶ cp encrypt.cpp shift_encrypt.cpp
```

The `cp` command is followed by two file names; it makes a *copy* of the first file (if it exists in your current directory) and gives the copy the name of the second file. If you already have a file in your directory that has that second name, this command will overwrite it, so be careful! You can always see what files are in your directory with the `ls` command.

Finally, you will need to sync these files from your Halligan space to your personal computer using `Atom`'s `remote-sync` feature that you set up in lab.

Open `Atom`. If your `comp11` folder doesn't show up in the left sidebar, drag it back into your `Atom` window. Then, just as you did in lab, right click your `comp11` folder and select Remote Sync PRO -> Download Folder. This will copy `hw00` and all the files it contains onto your personal computer, which will allow you to begin editing them.

Section II: Written (10 Points)

Answers to the following questions must be typed, saved as `written.pdf`, and **uploaded to Gradescope**. All of these questions can be answered in, at most, a single sentence. You may use your textbook and the internet at large to find the answers. You are also welcome to modify `encrypt.cpp` in any way that might help you arrive at the answers (but you might want to make a copy first so the line numbers don't get messed up in the original file). You may not discuss these questions with other students.

1. You're all familiar with the standard arithmetic operators such as $+$, $-$, $\times$, and $\div$. What does the % operator do? (aka 20 % 4)
2. Throughout `encrypt.cpp` we use `endl` to begin printing on a new line. What is another way of accomplishing this?
3. On line 51 of `encrypt.cpp` we advance the character that we're working on with:
   `next_letter_pos = next_letter_pos + 1;`
   What is a shorter way to write this instruction?

4. If `encrypt.cpp` is given the word `zap` to encrypt, how many times will line 41 get executed?
5. If `encrypt.cpp` is given the word `zap` to encrypt, how many times will line 44 get executed?
6. If `encrypt.cpp` is given the word `zap` to encrypt, how many times will line 35 get executed? Read the overview section of the wikipedia page on "while loops" if you'd like a hint.
7. What happens if you change the `<` on line 35 to `<=`? Why? Hint: try adding a print statement on line 36, then compile and run the program before and after making the change on line 35.
8. What is the Linux command that you would use to compile the file `hw0.cpp` into an executable called `exe_ready`?
9. What will Linux name the executable if you compile a program without specifying an executable name?
10. A good program will behave consistently and predictably across every possible input that the user might supply. `encrypt.cpp` is not a good program. What input would make `encrypt.cpp` behave badly and why? Hint: try running the program on different wacky inputs and see what happens!

Section III: Experimentation

Next, you are going to perform three different experiments on the copies of `encrypt.cpp` that will hopefully help you understand and debug future programs.

Put it in your own words...

Open `my_encrypt.cpp`. At multiple points, the program uses `cout` to print text to the screen in order to prompt the user for information. But that text is in my words, not yours. Change these text prompts to sound more like you.

Restrict your program to just these prompts by commenting out the loop that spans lines 35 through 52. All text that appears after '//' on a line of code is a *comment*, which is ignored by the computer and thus has no effect when the program is run. Commenting out blocks of code is a great way to track down a bug by reducing the scope of your program. See if you can figure out the shortcut in `Atom` that allows you to comment out a selected block of code.

Each time that you make a change to `my_encrypt.cpp`, save it, compile it, and run it to ensure that your changes accomplish the intended task via:

```
▶ g++ -o encrypt -Wall -Wextra my_encrypt.cpp
▶ ./encrypt
```

Don't move onto the next task below until the program successfully prints out your custom prompt!

Break it...

Open `broke_encrypt.cpp`. You're going to break the program in three different ways. For each of these, think through the effect it might have before you make the change. Then change the code in the specified way, save it, compile it, and run it (if it compiles) via:

- ▶ `g++ -o encrypt -Wall -Wextra broke_encrypt.cpp`
- ▶ `./encrypt`

Observe what happens and whether or not it was what you expected. If the compiler gives you an error message, read it carefully. On future assignments, you will rely on these error messages to find unexpected bugs. Once you feel like you understand the error that you introduced (feel free to google the error!), correct it and introduce the next error.

1.) On line 28, delete the semicolon at the end of the line.
2.) Delete the left bracket on line 45.
3.) Comment out the entirety of line 51.

In case you happen to be wondering, you can force quit a program in `Terminal/puTTY` with `ctrl-c`.

A different shift...

Open `shift_encrypt.cpp`. The current shift value is 5. Change the shift value to something new (in order for this change to fully work, you'll need to make one other change to the program - what is it? Hint: how does the program check if a shifted letter might require wrapping around to the front of the alphabet?). Before you test out your new program, pick 3 words and compute their encrypted counterparts on paper. Your words should try to capture different "areas" of the alphabet (i.e., letters from the start, end, and middle). Then run your new program on each of those words to confirm that it produces the expected output.

- ▶ `g++ -o encrypt -Wall -Wextra shift_encrypt.cpp`
- ▶ `./encrypt`

Programming (85 Points: 10 for style, 35 for compiling, 40 for correctness)

****Please review our course policy on acceptable forms of assistance and collaboration on programming assignments before beginning this section.****

Now that you're warmed up, you're ready to alter the original `encrypt.cpp`. Currently, `encrypt.cpp` has a hard-coded shift amount. Once it is compiled into an executable, users are only able to encrypt words according to that single shift. Lame! Alter `encrypt.cpp` to allow the user to specify a shift amount of their choosing. That is, after the program asks the user to specify a word to encrypt, it should then ask them to specify a shift amount and use that amount to perform the encryption.

You may assume that the user will enter a value between 0 and 25. However, for extra credit, you may submit a program that can handle any integer as a shift value.

Make sure that you test your solution with many different words and shift amounts (pick a word and a shift amount, calculate the encrypted word on paper, then see if your program does the same thing). Your code should also adhere to the same style that is demonstrated in the starter code (indentation, good variable names, etc.). When you arrive at a working and readable solution, submit `encrypt.cpp` using the following command:

- ▶ `submit11-hw00`