COSC1076 | Advanced Programming Techniques

Tutorial/Lab | Week 06

Overview

The Week 06 tutorial/lab is to revise content from the last few weeks.

- C-Style strings
- `std::string` Class
- Program I/O

Tutorial Questions

Consider the following questions:

1. Compare and contrast a `std::string` and a C-style string?
   (a) What are the advantages of `std::string` over using a C-style string?
   (b) Under what circumstances would you use a C-style string?
2. What is:
   (a) An `std::istream`?
   (b) An `std::ostream`?
   (c) A `std::istringstream`?
   (d) A `std::ofstream`?
3. How do you open a file for reading?
4. How do you open a file for writing?
5. To open a file for reading or writing, the name of the file must be provided. What does this name actually mean about the location of the file on the computer's harddrive?
6. How do command line arguments get supplied to a C++ program?
7. How are command line argument separated?

Group Registration for Assignment 2

Assignment 2 will be conducted in groups of 4. These groups ***must*** be formed only within your lab. This is because progress updates for assignment 2 will be presented to your tutor during lab times.

Groups for Assignment 2 must be ***confirmed*** with your tutor by the ***your week 7 lab***.

If you have formed your group, inform your tutor.

Planning for Assignment 2 Progress Update 1

The first progress update for each group will be held during ***Week 7 labs***. (If you are already pre-pared you could do this update in Week 6 instead).

In this update you will need to have a brief discussion with your tutor about:

- How your group will manage their codebase. That is, will you have a git repository?
- How will your group co-ordinate/discuss issues?
- Does your group have a planning schedule of when to get work done?
- How will your group split up tasks?

Lab Questions

It is a good idea to attempt the lab questions before coming to class. The lab might also be longer than you can complete in 2 hours. It is a good to finish the lab at home.

You should demonstrate your work to your tutor.

Exercises

The exercise for this lab extend upon each other to build up a single complex program. You will need to work on them one at a time.

In this lab you are going to write your own mini-version of a single Unix utility program called `wc`. The `wc` program is available with most *Nix systems (including Linux, Ubuntu and MacOS) that can count the number of characters, words or lines in a file. You can find out more about this command by look up its "manual" (man-page), using the command:

```
man wc
```

We are interested in three of the modes of the `wc` utility:

- Line count
- Character count
- Word count

The lab question come with two sample text files of famous Australian poems. We can get the line count for one of these files by the command:

```
wc -l snowyRiver.txt
```

the character count by:

```
wc -c snowyRiver.txt
```

and and word count by:

```
wc -w snowyRiver.txt
```

These command will be useful to test your program that you will develop through this lab exercise.

1. Start by writing a very simple C++ program `myWC` that:
   (a) Opens a file for reading
   (b) Reads in the file line-by-line
   (c) Prints out the line
   (d) Closes the file
   To help you do this, you may hard-code the filename.

2. Now modify your `myWC` program to count the number of lines in the files and print that out instead. If you have done this correctly, the number your `myWC` prints should be the same as if you run:

   ```
   wc -l snowyRiver.txt
   ```

3. Modify you program to do the line counting in a separate function:

   ```
   int countLines(std::string filename);
   ```

   This function is give the name of the file to open, and returns the number of lines that have been counted

4. Add a function to your `myWC` program:

```
int countCharacters(std::string filename);
```

This function counts the number of character in the given file. Modify your program so that the number of lines *and* the number of characters is printed out. The number of characters should be the same as given by the `wc` utility:

```
wc -c snowyRiver.txt
```

5. Now modify your program to accept 2 command-line arguments. The first argument is the *mode* of the program, and the second is the *filename*. Like the `wc` utility, the mode is either:
   - `-l` to count the number of lines
   - `-c` to count the number of characters

   The filename is the name of the file for which to do the correct count.
   Once you are done, your program should print *either* the number of lines *or* characters of the given file, just like `wc`. That is you can execute:

```
./myWC -l snowyRiver.txt
```

   and

```
./myWC -c snowyRiver.txt
```

6. Finally extend your program to count the number of words in a file. Your program should:
   (a) Use the `-w` argument to specify that words should be used
   (b) Have a separate function for counting words
   (c) Produce the same output as:

```
wc -w snowyRiver.txt
```

7. *(Extension)* You may notice that in your program you are duplicating the opening and closing of file in each function. This can be avoided by using the file input-streams are parameters to the functions instead of the filename.
   Modify the `countLines` function, to use the following prototype:

```
int countLines(std::ifstream& input);
```

   Instead of being given the filename, it is given a *reference* to a file input-stream. The function may assume that the `std::ifstream` has already been opened for reading.
   After this:
   (a) Modify the main function to open/close the file.
   (b) Modify the functions for counting characters and counting words to also take the `ifstream` reference.