# CS 112 - Week 4 Lab Exercise - 2022-09-16

## Deadline

Due by the end of lab on 2022-09-16.

## How to submit

Submit your `.cpp`, `.h`, and `.txt` files for the problems below on https://canvas.humboldt.edu.

IF you prefer, you may instead compress your `.cpp`, `.h`, and `.txt` files to be submitted into a single `.zip` file and submit that `.zip` file to Canvas.

(I'll also accept the `.zip` file created when one downloads a folder from the CS50 IDE, as long as it includes all of your lab's `.cpp`,`.h`, and `.txt` files -- I suspect it will also contain your resulting executables, but that's OK.)

## Purpose

To practice reading everything from a file, and to practice writing and using functions that expect array parameters.

## Important notes

- Be sure to put BOTH of your names and today's date in each of the files for the new functions in this lab exercise.

## Problem 1 - function count_file_words

The purpose of this function is to give you an excuse to practice writing a loop to read everything in a file. (This function could also be a useful helper-function for a later lab or homework.)

Use the design recipe to design a function `count_file_words` that expects the name of a file, has the side-effects of attempting to open it and read each "word"/white-space-separated "chunk" from that file, counting how many there are, and returns the number of "words" read. (Remember to close the input file stream involved!)

In this case, if there is any trouble opening and attaching an input file stream to the provided file, `count_words` can simply return 0 (it could not read any words from that file, after all).

Hand-create files named `lab4-1.txt` and `lab4-2.txt` for two of your test calls, each containing a different number of "words".

Submit your files `count_file_words.cpp`, `count_file_words.h`, `count_file_words_test.cpp`, `lab4-1.txt`, and `lab4-2.txt`.

**OPTIONAL VARIATION:**

Use Week 2 Lab Exercise's function `five_letter_word` and only count the number of 5-letter words that are in the provided file. (That is, your optional-variation version would not count any "chunks" it reads that are of other lengths or that are not made up of five letters.) And it would return the number of 5-letter words it read.)

## Problem 2 - function show_words

Use the design recipe to design a function `show_words` that expects an array of words and its size, has the side-effect of printing those words to the screen, one per line, and returns the number of words written.

This function should not change its parameter array, so include keyword `const` in `show_words`' function header to enforce this. (As is done in `print_nums`' function header.)

Look at the posted Week 4 Lecture 1 file `print_nums.cpp` to see how `show_words`' tests should be written in `show_words`'s opening comment in `show_words.cpp`.

Look at the posted Week 4 Lecture 1 file `print_nums_test.cpp` to see how `show_words`' tests should be written in `show_words_test.cpp`.

Submit your files `show_words.cpp`, `show_words.h`, `show_words_test.cpp`

**OPTIONAL VARIATIONS:**

- If you would like to "number" each word in your version's printed results in some fashion -- for example,

```
1 eagle

2 to

3 finagle
```

  ...you may do so. You may add other "prettier" output features along with the words if you wish, also.

- If you would prefer for your version to return nothing -- to be a `void` function -- you may implement it that way.

## Problem 3 - function get_words

Use the design recipe to design a function `get_words` that expects a file name supposedly containing words, one per line, an array able to hold words, and that array's size, has the side-effects of:

- attempting to set up reading from a file with that name
- attempting to read enough words from that file to fill the provided array
- closing the input file stream involved

...and returns the number of words it thinks it read from that file into the array. (The provided file can contain words of any length, and they will be read into the array provided, unless you choose to do the optional variation below.).

You should **not** use keyword `const` in declaring this parameter array in `get_words`' function header -- it DOES change the parameter array (and also the corresponding argument)!

THIS FUNCTION CAN ASSUME THAT THE USER CALLS THIS PROPERLY -- THAT THE PROVIDED FILE CONTAINS AT LEAST AS MANY WORDS AS THE PROVIDED ARRAY's SIZE. (And it is fine if your function crashes/etc. if the user does not.)

Amongst your tests for this in `get_words_test.cpp`:

- You should (hand-)create files named `lab4-3.txt` and `lab4-4.txt` for two of your test calls, each containing a different number of words
- use arrays of different sizes for two of your test calls

- include a `cout` for each test, after the call of `get_words`, listing the words that now should be in the argument array, and then call Problem 2's `show_words` to print the contents of the array that should have just been filled.
- For example, assuming that `lab4-3.txt` happened to contain `alpha`, `beta`, `gamma`, `nu`:

```
string test1_words[4];

cout << (get_words("lab4-3.txt", test1_words, 4) == 4) << endl;

cout << "...and you should now see alpha, beta, gamma, nu,\n"

     << "   each on their own line:" << endl;

 show_words(test1_words, 4);
```

Submit the files `get_words.cpp`, `get_words.h`, `get_words_test.cpp`, `lab4-3.txt`, and `lab4-4.txt`.

**OPTIONAL VARIATION:**

Use Week 2 Lab Exercise's function `five_letter_word` and only read 5-letter words from the provided file into the provided array. (That is, your optional-variation version would skip words of other lengths, and only write 5-letter words into the provided array -- and it would return the number of 5-letter words actually written into the provided array.)