# CS 112 - Week 3 Lab Exercise - 2022-09-08

## Deadline

Due by the end of lab on 2022-09-08.

## 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 with C++ file input/output and arrays.

## 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 words_to_file

Use the design recipe to design a function `words_to_file` that expects a file name and a number of words desired, has the side-effects of:

- attempting to set up a file with that name for writing
- asking the user to enter that many words, writing each to this file on its own line
- closing the output file stream involved

...and returns the number of words it thinks it wrote to that file. (The user can enter words of any length, unless you choose to do the optional variation below.)

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

- Use `words_to_file` to create at least the files `lab3-1.txt` and `lab3-2.txt`, each with a **different** number of words.
- Include a `cout` to begin **each** test, before its call of `words_to_file`, letting the user know how many words they should be asked to enter for that test
- Include a `cout` to end **each** test, after its call of `words_to_file`, saying the user should now find a file with that name, containing how many words, which should be those they just entered.
- For example,

```
cout << "You should be asked to enter 4 words: " << endl;
cout << (words_to_file("lab3-1.txt", 4) == 4) << endl;
```

```
cout << "...and you should now have a file lab3-1.txt with the 4 words\n"
     << "   you just entered" << endl;
```

Submit the files `words_to_file.cpp`, `words_to_file.h`, `words_to_file_test.cpp`, `lab3-1.txt`, and `lab3-2.txt`.

**OPTIONAL VARIATION:**

Use one or both of Week 2 Lab Exercise's functions `five_letter_word` and `ask_for_word` to ensure that *only* 5-letter words are written to the file.

## Problem 2 - function read_first

Use the design recipe to design a function `read_first` that expects a file name, has the side-effects of:

- attempting to set up a file with that name for reading
- attempting to read a single word from that file
- closing the input file stream involved

...and returns the word it read.

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

- Use files `lab3-1.txt` and `lab3-2.txt` for two of your test calls.

Submit the files `read_first.cpp`, `read_first.h`, and `read_first_test.cpp`,
(and you are already submitting `lab3-1.txt` and `lab3-2.txt` as part of Problem 1.)

## Problem 3 - some starting array practice

Write a `main` function in a file `array_play.cpp` that does at least the following:

- declares and fills three different arrays, each of a different size, and each containing elements of a different data type
- loops through each and prints to the screen the contents of each array is some pleasing, readable way

Submit the file `array_play.cpp`.

**OPTIONAL VARIATIONS:**

- You can interactively request input from the user to fill one or more of your arrays if you would like (instead of hard-coding their contents).
- You can do something else to or with each element in one or more of your arrays, in addition to printing their contents to the screen, if you would like.