Wesleyan University, Fall 2022, COMP 211

Lab 4: Header Files and Search

---

1. Overview

In today's lab we'll look at multi-file C programs and correct (and incorrect) implementations of binary search. Rather than any specific programming task in lab today, you are free to get started on the homework and I and the CAs will be around to answer any questions.

For this lab, you should create a new directory lab4 in your labs directory to hold your code.

2. Compiling multi-file C programs

Creating and using header files (`.h` files) and multiple `.c` files will give us better ways to organize our code, and to re-use code we have already created. Before continuing, read through the `compiling.pdf` handout associated with the lab.

Now, download the code files associated with today's multi-file linear search program from the class webpage. In this code you'll see that the linear search function is in one `.c` file, yet can be called from one or more other `.c` files. Use the `Makefile` to compile a `tests` program and a `driver` program by typing the following commands:

```
make tests
make driver
```

If you only change a `.h` file but not a `.c` file, the compiler will not think it needs to re-build the executable, so you will need to type `make clean` to remove the executable before then building `tests` or `driver`.

Run the `tests` program and look at the associated code. Make sure you understand what is happening. Similarly run the `driver` program and look at the associated code. Make sure you understand what is happening.

3. Variations on binary search: `binsearch_examples.c`

We will talk about binary search in detail in class tomorrow, as well as this code. In lab today, your goal is just to get a little bit of familiarity with binary search and its variations, and work with assertions, if you have already finished the other parts of lab.

Binary search is notorious for being easy to incorrectly implement. The file `binsearch_examples.c` gives 4 attempts at binary search, two of which are correct and two of which are incorrect. Playing around with these examples, do you understand why the incorrect implementations are indeed incorrect?

Challenge: Assertions can be used to assert that something should be true about your code at any given point in the program. For example you might wish to use assertions to check that the value of a variable is as you think it should be. To use assertions, include the header file `assert.h` in `binsearch_examples.c`. The function call `assert(e)` takes a boolean expression `e` and evaluates it. This expression could be another function call. If `e` evaluates to true, nothing happens and

control passes to next instruction in your code. If `e` evaluates to false, then an error message is printed to the terminal and execution terminates without executing any more lines of code. Now, go back and look at the the binary search functions: is there a way you could use assertions to check whether your implementation is correct? What if you were to use linear search to do the checking? Recall our discussion in class on loop invariants and what must be true each time through the loop in binary search. Hint: you'll want to add assertions at the top of each iteration through the loop and after the loop finishes (or conversely before you enter the loop and at the bottom of each iteration through the loop). Note: in practice, since checking assertions is costly, typically after one is sure the code is operating correctly, assertions are set to not be evaluated (are turned off).