CSE 333 25au Exercise 5

out: Friday, October 3, 2025
due: Monday, October 6, 2025 by **10:00 am**, **No late exercises accepted**.

NOTE: we're skipping exercise 4 this quarter, which is just a simplified version of exercise 5. Y'all are just too good!

Goals: Write a multi-file C program that properly uses header, implementation, and client files to partition a program. Learn how to compile programs with Make.

Description: Your job is to write a multi-file C program. You should write the following three files:

- `NthPrime.h` : a header file, containing a single function prototype declaration for a function called `NthPrime()` , as well as comments above the prototype documenting how to use the function. The function should accept a single `int16_t` parameter, and it should return an `int64_t` . The function should return the nth prime number, where `n` is the function's parameter. Note that `NthPrime(1)` should return 2, `NthPrime(2)` should return 3, `NthPrime(3)` should return 5, and so on. The result of `NthPrime(n)` is not defined if `n` <= 0, and implementations do not need to check for this possibility. The header file should include proper header guards.
- `NthPrime.c` : a file containing the implementation of `NthPrime` . Feel free to use the simplest possible primality testing algorithm (hint: "x is not prime if it is divisible by ..."). You may also want to define some helper functions here as well. Use `static` to ensure that such helper functions, if any, have internal, not external linkage and thus are not visible to other files.
- `ex5.c` : a file containing a `main()` function that tests `NthPrime` by printing the input and output output of `NthPrime` for at least two different, non-trivial arguments (but you do not need to print an enormous number of test cases - keep it reasonable).
- `Makefile` : as explained below.

You should use (and include in your final submission) this unmodified Makefile: link (ex05_files/Makefile). Place the `Makefile` in your exercise directory and then run the terminal command `make` to compile your solution binary. Conversely, run `make clean` to delete any files generated by the compiler (including the final executable). The latter is useful to make sure you're not accidentally adding any temporary files to a git commit!

Your code must:

- compile without errors or warnings on CSE Linux machines (lab workstations, attu, or CSE home VM) by running `make` in the submission directory
- have no crashes, memory leaks, or memory errors on CSE linux machines
- be pretty: the formatting, modularization, variable and function names, and so on must make us smile rather than cry. (Suggestion: see if `cpplint --clint` reports any problems. However, you

may ignore warnings from `cpplint --clint` about the header guard `#define` name not including a full file path as long the identifier has an otherwise appropriate name.)
- be robust: you should think about handling bogus input from the user, and you should handle hard-to-handle cases (if there are any) gracefully.
- have a comment at the top of each of your files with your name, student number, and CSE or UW email address.

You should submit your exercise using the Gradescope dropbox linked on the course resources web page.
