CS246 lab Notes #3 prototypes, header files, preprocessor directives, cout, and thg
Tue, Feb. 9
Instructor: David Cooper

- Make a lab03 directory in your course directory
- Initialize a repository and clone it to your home cs246 directory
- Copy the "hello.h" file from the /rd/cs246s2016/shared/lab03/helloLanguages directory into your home cs246/lab03 directory.
- Add and commit hello.h (in cs246/lab03)
- Prototypes reviewed
    - Recall a function prototype is the function declaration statement without the actual code for the function.
    - Prototypes tell the compiler that a function exists but will be defined later.
    - Why do we have prototypes?
        - g++ is a "one-pass compiler"
        - It doesn't go back and figure out what to do with a function they didn't know would exist later
        - Thus functions would have to be written in a dependency based order
        - Prototypes are used to tell the compiler that a function WILL exist, but it hasn't been defined yet.
- Preprocessor
    - What is the preprocessor?
        - There are advantages to certain things being done before compiling begins. For example, giving the compiler prototypes for functions like "hello" so that assembly can be written properly.
        - The preprocessor does many convenience jobs, rewrites the source file without doing any actual compiling.
    - #include
        - Adds library functions to do specific tasks
        - #include <iostream>
        - This adds the standard output/input streams to the compiled program
            - cout, cerr, cin, etc.
        - #include <string>
            - for std::string.
        - #include <cmath>
            - Has a lot of math functions like trigonometry, exponents, etc.
            - All the math functions are listed at http://www.cplusplus.com/reference/cmath/

- #define
  - Syntax: #define <replace-ee> <replace-or>
  - Example:
    - #define BUFFERSIZE 100
    - Then you might have code like:
    - int buffer[BUFFERSIZE]
    - for(i = 0; i < BUFFERSIZE; i++)
    - This makes code more readable, and easier to modify because you change the BUFFERSIZE in one place.
  - #define vs. const
    - #define is a preprocessor operation, changes text, but is not part of the actual compiler
    - Affects all subsequent code, regardless of scope
    - #define can change more than just variables, can represent functions, etc. but this is not recommended.
- #if, #ifdef, #ifndef, #else, #elif, #endif
  - The C preprocessor contains a simplified conditional system.
  - Works very similarly to "if-then" statements, but more efficient in the compiled code because the code removed never even gets compiled.
  - Usage
    - #if X == 2 (if X is #defined to be 2)
    - #ifdef X (if X is #defined at all)
    - #ifndef X (if X is not #defined)
    - #else – as expected
    - #elif X == 2– "else if X is #defined to be 2"
    - #endif – ends the if block
  - Often used for cross-platform code
    - #ifdef WINDOWS, #ifdef UNIX
    - #ifdef LITTLE_ENDIAN
  - Dependency checking
    - #ifdef is very useful in header files…

- Header files
  - We've seen header files be used, but not actually written one.
  - Header files contain function prototypes, #includes, #defines, global variables, and other things such as structs, enums, and typedefs.
  - The reason for a header file is to make multiple source file interdependence much simpler.
  - When dealing with multiple header files, you may get #include loops.
  - To prevent this, use the following convention (illustrated below through example)
  - Add this to the file hello.h
    - #ifndef HELLO_H
    - #define HELLO_H
    - // header file stuff
    - #endif

    - Thus if the file is included more than once you cannot have a loop (all the code in the header will be erased by the preprocessor)
- Multiple source files
    - If you have split your source code into more than one file, you must list all relevant files when compiling, for instance:
      `g++ –g file1.o file2.cpp –o output`
    - Try the following exercise:
        - Declare a function `mystery`, that takes a double from the console input (keyboard) and does a mystery to it, then returns the result of the mystery as a double.
            - put the function in a `mystery.h` header file with preprocessor code so that it is only included once.
        - write a `lab03.cpp` which uses the two functions declared in `hello.h` and `mystery.h`, for example, say hello to the user, and ask for input of a double from the user and print the result of the mystery function.
        - compile `lab03.cpp`, but do not link it (since the functions are not yet defined; they are only declared)
        - You should now have a `lab03.o` file
        - add and commit your files to your repository.
        - write a file `<username>_twoFunctions.cpp` which includes "`hello.h`" and "`mystery.h`" and contains two functions, one that defines/implements the hello function from `hello.h`, and the other, called mystery from `mystery.h` takes a double from the console input (keyboard) and does a mystery to it, then returns the result of the mystery as a double.
        - Have your partner write a second file `<partnerusername>_twoFunctions.cpp` which defines hello and mystery function each in a different way than the original functions.
        - include all header files properly and compile 2 different programs:
            - `<username>_lab03`, which uses `lab03.o` and `<username>_twoFunctions.cpp`
            - `<partnerusername>_lab03`, which uses `lab03.o` and `<partnerusername>_twoFunctions.cpp`
        - test each program and see how they differ.
        - Add, commit and push your code back to your course directory repository.
- cout and <iomanip>
    - You can print anything with cout
    - Using setbase, you can change to convert decimal values to output decimal, hexadecimal and octal.
    - Reference sheet is page 351 in Gregoire, Professional C++, 3rd Ed.

    - Special manipulators

        - boolalpha (noboolalpha) - whether or not booleans should be true/false vs. 1/0
        - hex, oct, dec -base manipulators
        - setprecision(x) - x is the number of decimal places displayed
        - setw(x) - x is the minimum field width for numeric data
        - setfill(x) - x is the fill character to pad a number
        - Escape characters:
            - To print non-standard characters, we use the special backslash character '\' to represent when a character is non standard.
            - \n – newline
            - \r – carriage return. Equivalent to \n, however there is a difference on Unix Systems.
            - \a – bell
            - \b – backspace
            - \t – horizontal tab
            - \v – vertical tab
            - \\ - backslash
            - \? – question mark
            - \' – single quote
            - \" – double quote
            - \000 – octal number
            - \xhh – hexadecimal number
            - Note: using streams, the % symbol does not need to be escaped, unlike printf.
- thg - TortoiseHg
    - this is a GUI to interact with mercurial repositories.
    - run thg using the command:
        - thg
    - a gui will show up
    - File... Open Repository (or Ctl-O)
    - Select your local repository for lab 03
    - do you see different revisions
    - select each one and see what changes.
    - explore the gui and try to learn what the buttons do.
    - http://tortoisehg.readthedocs.org/en/latest/quick.html