CS246 lab Notes #3 prototypes, header files, preprocessor directives and printf (maybe)

- cal
    - Unix calendar program
    - cal [[month] year]
    - If one number as argument, it displays that year.
    - If two numbers as argument, the first is the month, the second is the year, displays that month.
    - Example:
        - cal 1 2004
        - cal 1742
- Copy the "example.h" file from the handouts directory into your cs246 directory.
- 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?
        - gcc 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 "printf" 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 <stdio.h>
        - This adds the standard output/input functions to the compiled program
            - printf, scanf, etc.
        - #include <ctype.h>
            - We saw this before, has useful character functions.
        - #include <stdlib.h>
            - Has some important standard library functions
            - exit – forces the program to exit early.
        - #include <math.h>
            - Has a lot of math functions like trigonometry, exponents, etc.

        - Requires to add “-lm” to the compile statement, for example:
            - gcc –g –Wall –lm –o test test.c
        - All the math functions are listed on page 251 in K&R
    - #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 that we haven’t discussed in lecture yet (structs, typedefs, enums… but we’ll get to that later)
    - The reason for a header file is to make multiple source file interdependence much simpler (we will get to this later too!).
    - When dealing with multiple header files, you may get #include loops.
    - To prevent this, use the following convention (illustrated below through example)

  - In file example.h
    - #ifndef EXAMPLE_H
    - #define EXAMPLE_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 files when compiling: gcc –g file1.c file2.c file3.c –o output
  - Try the following exercise:
    - write a file input.c which contains two functions, one that takes an int from the keyboard and returns it, and the other takes a double from the keyboard and returns it.
    - write an appropriate corresponding input.h
    - write a lab03.c which uses the two functions defined in input.c, for example, ask for input of an int and a double from the user and print them out.
    - include all header files properly and compile
- printf
  - You can print anything with printf, integers, floats, strings, hexadecimal numbers, etc.
  - Reference sheet is page 154 in K&R
  - Special conversion chars
    - %d – integer
    - %c – character
    - %s – string
    - %e, %f, %g – floats. e = always scientific notation, f = never scientific notation, g = either based on number size
    - Use leading numbers for display precision
    - Escape characters:
      - To print non-standard characters, we use the special 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

- HOWEVER: exception to this is %% is the escape sequence for %, NOT \%