Assignment Homework 1 (warmup)

Due Date Mon 4/2/2017, **before class**.

Homework 1

The goal of this homework is to get basic familiarity with the environment which we will be using during the rest of the course: connecting to `linux.cs.uchicago.edu` using SSH, preprocessing, linking and compiling C files and git repository access.

Tasks

1. Connect to `linux.cs.uchicago.edu` using SSH.
2. Generate a public/private key pair. Add the public key to your gitlab profile via `https://mit.cs.uchicago.edu/profile/keys` to enable password-less access to gitlab.)
3. Clone the class repository (the repository URL can be found at `https://mit.cs.uchicago.edu/groups/mpcs51040-spr-17/mpcs51040-spr-17/`). Make sure to use the 'SSH' url when cloning.
4. Clone your personal repository. You can find the URL for your personal repository when you login to gitlab. It should look like `https://mit.cs.uchicago.edu/groups/mpcs51040-spr-17/yourcnetid`.
5. Write your own `helloworld.c`. Make sure it compiles *without warnings* when compiled using `gcc -Wall -Werror -pedantic -std=c11`. **Make sure your program contains at least two functions, i.e. don't put all your code in `main`.**
6. Add your `helloworld.c` **in directory `homework/hw1` of your personal git repository**. and commit. Do not forget to push to the server! (You can check using the gitlab website if the file was properly pushed.) *Only add the `.c` file, not the compiled program.*
7. Create a file called `preprocessor.txt` and `preprocessor2.txt` (and add it under `homework/hw1`) which demonstrates the following principles:
   - Token substitution (i.e. replacing text)
   - Conditional output (`#ifdef` and friends...)
   - Including another file (called `preprocessor2.txt`)

   Remember, you can invoke the preprocessor using the `cpp` command.
8. Is it possible to use the preprocessor to change "Hello World" into "Hello Class" in your example C program? (You can find an explanation of why (or why not) here: `https://gcc.gnu.org/onlinedocs/gcc-4.9.2/cpp/`)
   Update your `preprocessor.txt` file to show that replacement of text in a string literal (i.e. text between `"`) is or is not possible.
   (Don't forget to commit your updated file)

Brief Recap of the Preprocessor

- The preprocessor is a program which reads input, performs a set of changes to the input and outputs the changed text.
- On `linux.cs.uchicago.edu`, you can invoke the preprocessor using `cpp` or `gcc -E`.
- The preprocessor processes text (i.e. human readable, as opposed to binary) files.
- The input text is interspersed with *preprocessor directives*, which are commands to the preprocessor.
- The default behaviour of the preprocessor is to pass the text unmodified from input to output.
- Text changes made by the preprocessor broadly fall in two categories:
  - *Token* substitution
  - Directives to tell the preprocessor to stop outputting text and to start outputting text again.
  - Directives to tell the preprocessor to suspend reading the current file and to continue processing another file instead (`#include` directive).
- The preprocessor reads the input, and splits it into *tokens*. In the simplest case, a token corresponds roughly to a word. (See `https://gcc.gnu.org/onlinedocs/gcc-4.9.2/cpp/Tokenization.html#Tokenization` for a more formal description of the tokenization process).
- The `#define token1 token2` command tells the preprocessor that, going forward, whenever the *token1* is seen in the input, it should be replaced by *token2* in the output. Note that *token2* can be empty, in which case the directive takes the form `#define token1`.
- The `#ifdef token1` ... `#else` ... `#endif` directives instruct the preprocessor to either process the input between `#ifdef` and `#else` (if *token1* is `#define`d) or between `#else` and `#endif` (if *token1* has no special meaning to the preprocessor).