Assignment 2: Hello World

Description and purpose: This is a simple assignment to walk through the steps of modifying a simple "Hello World" program and submit it through GitHub. The purpose of this activity is to demonstrate proper use of header files, to use GitHub, to explain proper coding processes, and to demonstrate command line parameters.

Background

You should have all the following set up and working:

- GitHub account.
- git installed and working correctly.
- g++ installed and working correctly.

Note Regarding IDEs: If you install any IDE like CLion (free to students), you will be able to create, compile, and run programs from the IDE itself and not need the command prompt. However, running in an IDE is *not* the same as compiling from the command line. You may be able to get away with this, and you may never have to compile at the command line in this class, and if you're lucky, no one will ever know the difference (but don't bet on it). However, there is a danger in this. Compiling from an IDE can mask errors and can allow you to use non-standard C++ libraries (especially if you use Visual Studio or if you are on a Mac). Then when you turn in your code, it will not compile with g++ under Linux, and you will get a 0. The only true test of your code is compiling and running it at the command line using g++.

Read this entire assignment thoroughly before you begin any coding.

Instructions / Specifications:

- Follow the assignment-specific instructions using this GitHub assignment invite: https://classroom.github.com/a/2C_5__Dp
- Once you have completed the assignment-specific instructions up to the "*Now you are ready to begin working*" step, you will have four files on your computer: **README.md**, **.gitignore**, **main.cpp**, **main.h**
- Note that you did not need to make a **.gitignore** this time; it's made for you, but you should study it so you understand how .gitignore works.
- Edit your local **README.md** so it is a proper README.md (you may want to wait to the end so you know what to describe)
- Fix your comment headers in main.cpp and main.h to be yours (delete the ones that are there).
- Make two blank (empty) files locally in your project folder: **functions.cpp** and **functions.h**.
- In both new files, put your comment header.
- In **functions.h**, put in the #ifndef directive below your comment header like so…

```
1 /**********************
2 * Your comment header here.
3 **********************/
4
5 #ifndef COMMANDLINE_FUNCTIONS_H
6 #define COMMANDLINE_FUNCTIONS_H
7
8
9
10 #endif //COMMANDLINE_FUNCTIONS_H
11
```

Note that **all the code** you will later place in this file will go *inside* the #ifndef directive *after* the first #define, as shown, and **before** the #endif.

- In **main.h**, add **#include "functions.h"** right below **#include <stdio.h>**
- Save all your files and compile them at the command line with the following command: **g++ -I ./ *.cpp**. This will create an executable called **a.out**. If it does not compile, read the error and fix your program until it compiles.
- Run your program at the command line. Type **a.out** and it should run. If it doesn't, try **./a.out**
- If everything worked correctly, your command prompt actions should look similar to this…

```
alex@alex-laptop:~/Desktop/temp$ g++ -I ./ *.cpp
alex@alex-laptop:~/Desktop/temp$ a.out
Hello, World!
alex@alex-laptop:~/Desktop/temp$
```

- If it didn't work, fix it now before moving on.
- Assuming it all worked, it's time to commit and push…
    - **Add**: **git add README.md functions.cpp functions.h main.h**
    - **Commit**: **git commit -m "initial commit"**
    - **Push**: **git push -u origin main**
- Go to your repo on GitHub and see that all your files are there and up to date. If they aren't, fix it.

Now you are ready to code…

- In **main.cpp**
    - Change "*Hello World*" to "*My Command Line Analyzer...*"
    - Add this print line below that print line

      printf("The number of command line parameters: %d\n", argc);
- Compile and run your program three ways, and observe the output is a count of the things on the command line.
    - **a.out**
    - **a.out hello**
    - **a.out hello world**
- Stop again and add and commit with an ***intelligent and descriptive*** message about what you just did. You can also push, but it's not necessary at this point.

This is a proper development process! Make a small, self-contained, and logical change; stop and test it; and if it works, commit with an intelligent commit message. If you go much further than this without testing and committing, you're doing it wrong! This is the process that will be expected for all assignments in the course. If you fail to develop using this process, you will fail the course.

- Now stub a function called **strlength** in your functions module (a 'stub' is a prototype plus an empty function definition).
    - In **functions.h** add a prototype inside the **#ifndef** directive: **int strlength(const char*);**
    - In **functions.cpp,** add **#include "functions.h"** and the following code:

```
int strlength(const char *str){

    return 0;
}
```

- In **main.cpp** add the following below the last **printf** you added:

```
if(argc > 1){
    printf("The length of parameter 1 is : %d\n", strlength(argv[1]));
}
```

- Compile and run with:
  - **a.out**
  - **a.out hello**
- You should see output like this:

```
alex@Alex-Katana:~/Documents/Code/scratch$ a.out
The number of command line parameters: 1
alex@Alex-Katana:~/Documents/Code/scratch$ a.out hello
The number of command line parameters: 2
The length of parameter 1 is : 0
```

- The length is wrong because the **strlength()** function is just stubbed, but **stop here** to test and commit. **This is a proper programming process.** You now know you have a working architecture, so it's time to add and commit again. It's also a good time to push (not necessary, but good practice).
- Now, working in **main.cpp** and **functions.cpp** and **functions.h**, modify your program so it works like the example below.
- You must use a loop of some sort in **strlength( )** and may not use any library function to tell you the string length. In other words, make a **while loop** to count the characters in the array and return that count. Do not use a for loop for this function; see best practices for why.
- Other than having to use a while loop in **strlength( )** and maintaining the basic file and function structure already given (do not add more functions), now change/add whatever you think you need to code to make your program work **EXACTLY** like the examples below.
- **Your program has to work for any number of parameters from 0 to n, where n is any number.**

```
alex@Alex-Katana:~/Documents/Code/scratch$ a.out
The program name is 'a.out'
The length of the program name is 5
The number of command line parameters: 0
alex@Alex-Katana:~/Documents/Code/scratch$ a.out hello
The program name is 'a.out'
The length of the program name is 5
The number of command line parameters: 1
    The length of parameter 1 is : 5
alex@Alex-Katana:~/Documents/Code/scratch$ a.out hello world
The program name is 'a.out'
The length of the program name is 5
The number of command line parameters: 2
    The length of parameter 1 is : 5
    The length of parameter 2 is : 5
alex@Alex-Katana:~/Documents/Code/scratch$ a.out this is my first program in data structures
The program name is 'a.out'
The length of the program name is 5
The number of command line parameters: 8
    The length of parameter 1 is : 4
    The length of parameter 2 is : 2
    The length of parameter 3 is : 2
    The length of parameter 4 is : 5
    The length of parameter 5 is : 7
    The length of parameter 6 is : 2
    The length of parameter 7 is : 4
    The length of parameter 8 is : 10
```

Notes:

- Your output must be **identical** with the same input. Even one character, one space, or one line off, and you will lose points. Attention to detail and specifications is one of the most important qualities of a programmer.
- You will be graded on efficiency. Needless variables, superfluous if statements, overly complex code, etc. will all count against you. Remember, coding is as much about *removing* unnecessary things as you add code as it is about actually adding code.

Grading: The grading process is detailed on the course website. Read that page carefully! **Pay particular attention to the "automatic zero" section.** Do not lose points or get a zero for failure to follow specifications or best practices. Use that section like a checklist, and go over it **carefully**.

Always use this checklist to double-check your code. 👈 IMPORTANT!

Submission: When you are ready for grading, use the write submission feature in Blackboard and submit your repo's link. Unless you have an important note to include about your submission, **just submit the link—no additional information**. If you do have important information to convey, that's fine, but otherwise—just the link. If you need to fix/change something **after** you submit but **before** it's graded, just fix/change it and push again. **Do not resubmit the assignment before getting a grade**. Only re-submit after you get a grade and want a re-grading.