ME 459 Assignment 5

Jackson Fox

Due 10/21/2021

Problem 1

a)

The purpose of the include guard lines are to prevent redeclaration of variables, structs, or other items when a .h file is utilized in multiple places. If include guards are not utilized, and you utilize a .h file starter.h in Function1.h, then attempt to *#include* both starter.h and Function1.h in your main file, the the compiler will try to define anything contained in starter.h twice, which results in a compilation error in c. The include guards allow header files to be called in multiple locations in the same compilation without the preprocessing and subsequent allocating/defining being repeated.

b)

The first line *#ifndef EXAMPLE.h* checks if EXAMPLE.h has already been defined. If it returns false, the .h file has not been defined as part of the compilation so far, and moves to the next line *#define EXAMPLE.h*, and continues through the file. If the check *#ifndef EXAMPLE.h* returns true, meaning that EXAMPLE.h has already been defined by the preprocessor during the compilation, then the preprocessor jumps to the *#endif* at the end of the declaration file, skipping any declarations included in the .h and avoiding the problem of redeclaration discussed above.

Problem 4

b)

When you git add a file, the file at that point in time is staged to be committed to a repository, but is not yet committed. If changes are made to the file after the file is staged, git add will need to be ran again to stage the file with the new changes.

When you git commit a file, the version (snapshot) of the file that was staged through git add is committed to the repository DB. That means that the file is added and this version can be accessed from the git repo going forward. Only files that are git added (staged) and then git committed to the repository can be accessed from the repo in the future, regardless of if other files are locally edited.

The git push is used when working with a team on a project. The push sends a specified branch from the local GIT repository DB to the master repository which can be accessed by all other team members on the project. This branch includes any and all commits that you have created since the last fetch of the remote repo to your local repo, and is the way to contribute your changes to a project for access to the group.

c)

Executables should not be tracked in git because they can be setup/machine specific. When working on a group project, each member may utilize a different approach for executing their code or be on different machines and require different specifics for execution of the code, therefore minimizing the utility of including the executable in the repo. Additionally, every time that you commit a new version of a file to the repo, you would need to commit a new version of the executable. Finally, one of the key powers of git is the ability to go back in time to compare changes to old versions of files or to create branches to deal with specific bugs or behaviors. Executables are not human readable, and cannot be modified to correct for problems or bugs, you would need to find the related files that generated the executable. Therefore, this key value of git is lost when dealing with executable files.