The motivation and objectives of new builder:

The motivation of the new builder is to add a new feature to the InitClassFiles builder. I created a script named CreateMakefile.sh. What my script does is create a Makefile for the all classes created with the InitClassFiles script. One of the great features of C++ is separate compilation. Instead of writing all the code in one file, and compiling that one file, you can write multiple ones and compile them each as well. There will most likely be many files created so you want to use a fast way to (re)compile everything. When working on homework 0 and homework 1, I came up with the idea to create a Makefile script so that I wouldn't have to keep typing 4+ files to compile. Creating a Makefile script would also save time from having to go into the Makefile and editing it whenever it needed changes, and possibly missing something. That would also take time to debug if there were a lot of files in the Makefile.

Current state of builder:

Currently, CreateMakefils.sh takes in at least 1 parameter, and an optional 2nd parameter. The first parameter is required because it will be the name of the executable created by the Makefile. To run the executable, the user must "make all", and then "./<executable name>". The second parameter is a file that is passed in which contains flags that the user wants to use when compiling and creating object files. If the user does not pass in any parameters, the Makefile will not be created, and will output the usage.

How to improve/change it:

A way to improve the builder is not have the user pass in a file with the desired cflags. Instead, the user can directly pass it in after the first parameter. (ie: ./CreatMakefile.sh hello –W –Wall –Werror) The user can also pass in nothing, and have a default executable name instead. An interesting utility that can be added is chmod for a Makefile on scripts. LFLAGS, linking flags, could be used as well. A DEBUG flag would also be useful so that it will include debugging information into the executable. There could also be a way to change the compiler and compiler options in the Makefile if the user wants, since currently g++ is the default compiler in CreateMakefile.sh.

Modifications/additions that must be done:

An important modification is updating classList.txt when a class created by InitClassFiles.sh is removed. If a file is removed, classList will still have the name of the object. Therefore, CreateMakefile will still include the removed file inside the Makefile. Storing all the classes created was to help CreateMakefiles find the correct dependencies. A way of fixing this is adding a section to CreateMakefile that checks if classList matches with the current classes in the directory, and if it isn't, find the name of the class that was removed and delete it from the list.

Another modification to my script could be the way that it checks whether there is an unrelated file in the directory. Currently, it goes through the directory and stores the names of the .cc and .hh files, then compares it to classList (a file that keeps track of all the classes created with InitClassFiles). A possibly easier way could be just counting the number of relevant objects in the directory (.cc, .hh), and then comparing it to the number of classes in classList. Or instead of creating a new file to store the files in the directory, it can directly compare the files in the directory to each item in classList.

Realistically, the programmer should not even have unrelated files in the same directory as a certain project. A way to change my script in this sense could be that it doesn't create a Makefile at all until the unrelated files are moved or deleted, or if those files are related, he updates classList accordingly.

Design and implementation strategies:

The user calls the script, and passes in at least one argument. If not enough arguments are passed in, it will echo the usage, and exit out of the program. No Makefile is created. The first time CreateMakefile.sh is called, it will create a file called Makefile. The header is added in to the top of the Makefile first.

After that, the script checks if there is already a main.cc. If there isn't, a skeleton main function is created.

Next, are the macros for the Makefile. There is a macro for execution name (PGRM), one for objects (OBJS), one for the name of the compiler (CC), and another for the compiler flags (CFLAGS). The first parameter passed into the script is stored in PGRM. OBJS is created by appending ".o" to each item in classList. The default compiler is g++. CFLAGS is created by reading in cflags.txt (if it is passed in) and writing it to the Makefile. If the program is passed in an invalid file, or cflags does not exist, then it will output an error message. The Makefile will be created without cflags.

"all:" is the first target, whose dependency is the executable name PGRM. target PGRM is written. For the separate object files, the script reads in line by line from classList and uses the current class to write each.

For make clean and make tar, I check if the current directory has any unrelated files. If it does, then it will not use the wildcard symbol for clean and tar. Instead, it will output each class with the correct extension respectively. For example, instead of "*.o", "classA.o classB.o" etc will be written to the Makefile.

At the end, cflags.txt will be removed if it exists. This is because the user might not pass in cflags the next time he uses the script, but the script will include the old file even if the user does not want to use cflags.

Figure: http://puu.sh/jV2Cq/09634d3e99.png

Difficulties encountered and actions taken:

The most difficult problem that I dealt with was with checking to see if the current directory has unrelated files. The problem with unrelated files was that the wildcard character that I used in the Makefile will take the unrelated files as well if they are .cc .hh or .o. I listed all the files with a certain extension, trimmed off the extension, stored each into a file, and compared the newly created file to the classList. If a match was not found, then it would return 1. If it was found, then it would return 0. (Something to add to modifications is to switch those values around since they're not very intuitive.)

If there are no unrelated files, make clean and make tar use the wildcard symbol to get the corresponding files needed. (This is the default.)

Figuring out what to do/output was also a challenge. At first I was going to just not create the Makefile at all, but I ended up outputting a message about unwanted files and creating the Makefile with the correct files. The reason why is because it isn't necessarily wrong to output everyfile that the user wants for a certain command.

Probably the part that took me the longest, and the most frustrating was figuring out how to create a correct Makefile. I hadn't used on in 2 years, and when I used it 2 years ago, I didn't completely understand it. So creating each target was difficult because I wasn't sure what to write to the file or what was needed.

Results/conclusions:

I was able to produce exactly what I wanted. Creating this script was relatively easier than before because I've become more accustomed to bash scripting.

Bibiography/references:

Lin, Charles. "Makefiles." *Makefile Tutorial*. University of Maryland, n.d. Web.
Mrbook. "Makefiles." *Mrbooks Stuff*. Mrbook.org, 29 Nov. 2008. Web. 31 Aug. 2015.
Tutorialspoint. "Makefile Macros." *Makefile Macros*. Tutorialspoint, n.d. Web. 31 Aug. 2015.