Lab 2 Directions

CS 3481 - Computer Systems I

Makefiles

make is a build automation tool that reads commands in a file (typically named makefile or Makefile) that specify how to compile code and create the executable target. Code that is compiled from the source for a Linux machine will almost always come with a makefile to do that work. Thus it is incumbent upon us as Linux developers to know how to create and modify makefiles.

For this lab assignment, you will modify one makefile, create two from scratch, and answer some questions in a quiz. The makefiles are worth 52 points. The quiz is worth 48 points.

First, take time to carefully read the "Introduction to Makefiles" document below this document's link.

Next, use a browser to log into git hub and then visit the following URL:

https://classroom.github.com/a/GUONQ29s

This will display a page where you will accept the assignment and then create a repository at GitHub that contains the starting code for this lab assignment. If you then click on the link that takes you to the repository, you can grab the URL to be used in the clone command on our Linux machine.

Now, log into our Linux machine.

Get into your CS 3481 directory and clone the repository that was created by typing this (but substitute githubusername with your GitHub username):

git clone https://github.com/ASU-Hamza-Classrooms/cs3481-lab2-githubusername.git

You'll need to enter your GitHub username and **Personal Access Token** (you can either use the one you created for lab 1, if you saved it somewhere, or create a new one) to perform the clone.

This will create a new directory containing a copy of the repository that is at GitHub. The links contain three directories. You'll be working on makefiles in each of those directories.

1. Modify the makefile in the Number directory

The makefile that is provided is not very sophisticated and unnecessarily complex. If you read the intro document carefully then you'll know that some dependencies and commands are not necessary. You will need to fix that makefile.

You can test your makefile by executing touch commands and then typing make. The touch command can be used to change the date on a file to the current date/time. Make decides which commands to execute by comparing the dates on files. Consider:

touch Number.C
make

The touch will cause Number.C to become newer then Number.o. The make utility will compare the dates, see that Number.C is newer then Number.o then execute the command to create a new Number.o. After that, make will see that Number.o has a newer date then the executable and will execute the command to create a new executable.

The commands below should cause every file that includes Number.h to be recompiled and the .o files to be relinked to create a new executable.

touch Number.h
make

The commands below should simply cause a new executable to be created (by relinking):

touch Number.o
make

After you fix the makefile, answer the questions in the first page of the quiz. Also, push your makefile to the remote repository.

git add makefile
git commit -m "Fixed Numbers makefile"
git push origin main

2. Create a makefile for the Animals code

The makefile in the Animals directory only contains comments that explain what needs to be added to the makefile. Follow the directions in that makefile. After you get that working, answer the questions on the second page of the quiz. Also, push your makefile to the remote repository.

git add makefile
git commit -m "Fixed Animals makefile"
git push origin main

3. Create a makefile for the Sorts code

Often developers of large projects will divide the project up into multiple directories. As you can see in the lab3/Sorts directory, there is a source directory to hold the .C files, and an include directory to hold the developer's header files. Take a look to see what is in each of these directories and then get into the source directory.

g++ provides the -I (dash-capital i) option to indicate the path to the developer's header files. For example, the command

g++ -c -I ../include BubbleSort.C

tells g++ to look in the directory ../include for the header files included by BubbleSort.C. Thus, the BubbleSort.C file can contain the include:

#include "BubbleSort.h"

and the compiler will look for that header file in the ../include directory. Note that the path given here is a relative path. The .. indicates one directory above the current directory. We could have instead given a full path like in this command.

g++ -c -I /u/css/studentid/3481/githubrepository/include BubbleSort.C

Note that the -I option does not apply to the standard header files within angled brackets. The compiler always looks in the standard header directory for those.

Your next task is to write a makefile for the Sorts code. That makefile will be in the source directory. You should see that you have a file called makefile in the source directory with more directions about what needs to be added. Before continuing, create a bin directory that is at the same level as the source and include directories.

Because the header files are in a separate directory from the source, you'll need to provide the path to each header file in the makefile. The -I option explained above works for gcc, but not for make. For make, you'll need to provide the path to the header file. For example:

sortProg.o: ../include/BubbleSort.h <rest of the dependencies are missing>

Better yet, you could make the makefile more portable by defining a macro and then using the macro like this:

sortProg.o: $(INC)/BubbleSort.h <rest of the dependencies are missing>

You'll want your makefile to be as simple as possible and still provide the desired functionality. To that end, you'll need to define the CC macro to specify the compiler to use and the CFLAGS macro to specify the compiler options. You also want another macro that specifies the path to the header files. You'll use that macro in the definition of the CFLAGS macro (you have to tell gcc where to find the header files) and when you

specify the location of the header files that a target is dependent upon. For example, the line below is using a macro named INC which was defined at the top of the file to be equal to ../include

sortProg.o: $(INC)/BubbleSort.h <rest of the dependencies are missing>

After you get this working, complete the third part of the quiz. Also, push your makefile to the remote repository.

git add makefile
git commit -m "Fixed Sorts makefile"
git push origin main

Last modified: Tuesday, January 28, 2025, 2:03 PM