CS1XC3 - Computer Science Practice and Experience: Development Basics

Assignment #8 - Makefiles

Weight

7% of total course grade

Due date

Sunday March 28th at 11:59pm

Primary Learning Objectives

- Create a makefile to compile code and conduct automated testing.
- Write C programs that utilize multiple files and command-line arguments.

Requirements

You are given a starter.zip file that includes the following:

- A Makefile very similar to that covered in week 10 lab.
- A src folder containing a library.h, library.c, and files max.c, mulitply.c and square.c
    - The library.h and library.c files are a common library of functions that are to be used in the implementation of max.c, multiply.c and square.c. We'll call this "the library" in this document.
    - multiply.c has already been implemented for you.
- A test_data folder that contains input and expected results txt files to be used in the creation of automated tests for max, multiply and square.

You should unzip the folder and copy the contents to the pascal server with an SFTP client (see the videos posted for accessing the pascal server in order to use an SFTP client).

In this assignment you will do the following:

- Implement max.c and square.c to find a max number, calculate square areas using the library functions and command-line arguments.
- Modify the makefile so that it is able to build max.c, square.c and multiply.c using the library.

- Modify the makefile so that it conducts automated tests of max.c, square.c and multiply.c using diff and the given input and expected results txt files.

It is **highly recommended** that you complete the assignment in the order it is described below, and that you get one section working before proceeding with the next. The assignment description is long because how to complete each step is described very explicitly with examples provided, but the total lines of code that either need to be created or modified is only 34 to complete the assignment.

Implementing Programs: multiply, square and max

The multiply program is implemented in multiply.c. It requires one int command-line argument that it uses to multiply numbers provided via standard input until EOF is reached. This program has been implemented for you already. Assuming that you have unzipped the starter code on the pascal server, if you navigate to the src folder you should be able to compile and run the program like this:

```
[brownek@pascal src]$ gcc –o multiply multiply.c library.c
[brownek@pascal src]$ ./multiply 2 < ../test_data/multiply_i
nput.txt
20
40
80
160
[brownek@pascal src]$ cat ../test_data/multiply_expected.txt

20
40
80
160
```

Notice how we compile the program using ***both*** multiply.c and library.c, similar to the example done in-lecture during week 9. We run the program using the multiply_input.txt file for standard input, and we see that the content matches the multiply _expected.txt file.

Implement square.c such that it reads ints from standard input that represent the side of a square until EOF is reached, and for each side outputs the area of a square with that side. The square program should utilize the relevant library method to help implement this functionality. The square program should accept a command-line argument that is the units (e.g. inches, cm, feet, etc.). This unit should be included with the output of each square area. The finished square program should work like this if used with the square_input.txt file as standard input (and we see that the output matches the square_expected.txt file):

```
[brownek@pascal src]$ gcc -o square square.c library.c
[brownek@pascal src]$ ./square inches < ../test_data/square_
input.txt
25 inches
100 inches
225 inches
625 inches
[brownek@pascal src]$ cat ../test_data/square_expected.txt
25 inches
100 inches
225 inches
625 inches
```

Implement max.c such that it accepts "any" number of ints as command-line arguments. It should use the relevant library method to find the maximum of these ints, and it should output this maximum int. Note that because command-line arguments are a string (char array) in C, they will need to be converted to ints using atio(), as done in the provided multiply program (see: https://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm). Dynamically allocate space for an array of ints that is large enough to hold the number of ints required (argc can tell you how much space is required). The finished max program should work like this if used with the following command-line arguments:

```
[brownek@pascal src]$ gcc -o max max.c library.c
[brownek@pascal src]$ ./max 4 3 2 1 5 7 8 10 6
10
[brownek@pascal src]$ cat ../test_data/max_expected.txt
10
```

Modify the makefile to build multiply, square, max

Next modify the provided makefile so that if the command **make all** is run from the command-line, it will compile the multiply, square and max programs, with the resulting executables placed in a directory called **build** (that as with test_data and src, is a subdirectory of the directory containing the makefile).

Your makefile must compile the library.o object file ***before*** compiling multiply, square and max, and your makefile must use it as a prerequisite in the compilation of multiply, square and max, rather than compiling multiply, square and max with library.c each time as in the above examples. This is so that the same library is not compiled repeatedly and unnecessarily. After running **make all**, in addition to the executables for multiply, square and max, the build directory should also contain library.o.

In order to implement this functionality, modify the existing makefile rule for $(all_targets) to compile the targets using library.o, and add an additional rule to compile the target library.o object file. These are the only changes required to implement this functionality and in total requires writing or modifying 4-5 lines of code Whenever possible while writing this code, use the variables for BUILD_DIR, SRC_DIR, as well as the special variables for the compiler and compiler flags. An implementation that "hardcodes" all of this functionality will not receive full marks.

Note that we can compile library.c separately to library.o, and then use it in the compilation of multiply, square and max. Here is an example of compiling library.o and then using it to compile the max, multiply and square executables, done manually from the src folder. Your makefile code is essentially doing this, but library.o and the executables should be put into the build folder.

```
[brownek@pascal src]$ ls
library.c  library.h  max.c  multiply.c  square.c
[brownek@pascal src]$ gcc -c -o library.o library.c
[brownek@pascal src]$ ls
library.c  library.o  multiply.c
library.h  max.c      square.c
[brownek@pascal src]$ gcc -o max max.c library.o
[brownek@pascal src]$ gcc -o multiply multiply.c library.o
[brownek@pascal src]$ gcc -o square square.c library.o
[brownek@pascal src]$ ls
library.c  library.o  max.c     multiply.c  square.c
library.h  max        multiply  square
```

Modify the makefile to automatically test multiply, square and max

Create a new target in the makefile called **test** that will run automated tests of each executable using diff (as discussed in the week 10 lab) when we run **make test**. diff will be used to compare the actual program output to the expected program output, knowing that if diff finds differences, then make will cease execution (again as discussed in the week 10 labs). You can assume that **make test** is being run after **make all** has already been run.

Your test target recipe (i.e. list of commands) should begin by creating a new subdirectory called **test** similar to how the **build** folder is created when we run **make all**. This **test** folder should temporarily contain text files of the actual output from programs run during automated testing. The folder should be removed as the last command of the recipe after all testing has completed (very similar to how we remove the build folder with the clean target). The exact test directory name to use should be configurable via a variable, in the same way the build and src directory names are configurable with a variable.

After the test subdirectory is created, each executable should be tested:

- The square executable should be run with the command-line argument "inches" using the square_input.txt file contained in the test_data subdirectory as standard input, and the output should be redirected to a file called square_output.txt in the test subdirectory. A diff should be done to ensure that square_output.txt in the test subdirectory is equivalent to square_expected.txt file in the test_data subdirectory.
- The multiply executable should be run with the command-line argument "2" using the multiply_input.txt file contained in the test_data subdirectory as standard input, and the output should be redirected to a file called multiply_output.txt in the test subdirectory. A diff should be done to ensure that the multiply_output.txt file in the test subdirectory is equivalent to the multiply_expected.txt file in the test_data subdirectory.
- The max executable should be run with "4 3 2 1 5 7 8 10 6" as command-line arguments and the output should be redirected to a file called max_output.txt in the test subdirectory. A diff should be done to ensure that the max_output.txt file in the test subdirectory is equivalent to the max_expected.txt file in the test_data subdirectory.

After each executable is tested as above, the test folder and all contents should be deleted. All of this should be done with the relevant variables for directories rather than hardcoding directory names.

Add a command to the clean target recipe that will remove the test subdirectory as well... if a test fails due to a diff finding differences, we may want to examine the results in the test subdirectory, but after doing so, we will want to remove the subdirectory.

After completing the above it should be possible to build and test the programs by running **make all** followed by **make test** followed by **make clean** or by running **make all test clean**. Example output for doing so is provided after the marking scheme using the instructor's solution. **This example output can be used to help create your makefile code, *as it effectively tells you what each command should look like for each of the above requirements*.**

Submission

Save your solution in a file named **a8.zip** and submit it to the Assignment #8 dropbox on Avenue to Learn. Your solution should include the exact same files and folders as the starter code. Your solution should include a modified max.c, modified square.c, and modifed Makefile, but all other files should remain unedited from the starter code.

Marking scheme

Component	Description	Marks
max, square	Do the functions produce the correct values? Do they use command-line arguments as expected? Does max use a dynamically allocated array? Do they use the library methods?	40
make all	Are the max, square and multiply executables compiled using library.o, and are all of these files in the build folder?	30
make test	Is automated testing carried out using diff, the test_data files, and the test subdirectory? Is the test subdirectory configurable? Is the test subdirectory temporary, i.e. created for test output files and then removed after testing is complete?	30
Total:		100

Example output:

*Running **make all,** followed by **make test**, followed by **make clean**:*

```
[brownek@pascal ~]$ make all
mkdir -p build
gcc -Wall -c -o ./build/library.o src/library.c
mkdir -p build
gcc -Wall -o ./build/multiply src/multiply.c ./build/library.o
mkdir -p build
gcc -Wall -o ./build/max src/max.c ./build/library.o
mkdir -p build
gcc -Wall -o ./build/square src/square.c ./build/library.o
[brownek@pascal ~]$ make test
mkdir -p test
```

```
./build/square inches < ./test_data/square_input.txt > ./test/square_output.txt
diff ./test_data/square_expected.txt ./test/square_output.txt
./build/multiply 2 < ./test_data/multiply_input.txt > ./test/multiply_output.txt
diff ./test_data/multiply_expected.txt ./test/multiply_output.txt
./build/max 4 3 2 1 5 7 8 10 6 > ./test/max_output.txt
diff ./test_data/max_expected.txt ./test/max_output.txt
rm -f -r test
[brownek@pascal ~]$ make clean
rm -f -r build
rm -f -r test
```

*Running **make all test clean***

```
[brownek@pascal ~]$ make all test clean
mkdir -p build
gcc -Wall -c -o ./build/library.o src/library.c
mkdir -p build
gcc -Wall -o ./build/multiply src/multiply.c ./build/library.o
mkdir -p build
gcc -Wall -o ./build/max src/max.c ./build/library.o
mkdir -p build
gcc -Wall -o ./build/square src/square.c ./build/library.o
mkdir -p test
./build/square inches < ./test_data/square_input.txt > ./test/square_output.txt
diff ./test_data/square_expected.txt ./test/square_output.txt
```

```
./build/multiply 2 < ./test_data/multiply_input.txt > ./test/multiply_output.txt

diff ./test_data/multiply_expected.txt ./test/multiply_output.txt

./build/max 4 3 2 1 5 7 8 10 6 > ./test/max_output.txt

diff ./test_data/max_expected.txt ./test/max_output.txt

rm -f -r test

rm -f -r build

rm -f -r test
```