CMPUT275—Assignment 1 (Winter 2024)

X. Li R. Hackman

Due Date: Friday January 26th, 8:00PM

Per course policy you are allowed to engage in *reasonable* collaboration with your classmates. You must include in the comments of your assignment solutions a list of any students you collaborated with on that particular question.

For assignment questions we will give you a sample executable — this is *our* solution to the problem which you will be tested against. You should use this assignment question to check what the expected output is. If you ever have a question "What should our program do when XYZ" the answer is what does the sample executable do? With the notable exception of *invalid input.* Breaking the rules of what the program expects is considered invalid input and wouldn't be a valid test case unless we explicitly told you what your program should do to handle such a problem.

Note: These sample executables are compiled on the student environment — that means they are only guaranteed to work on that environment. They will likely not work on your local machine, and will only work on Linux environments that are similar to the student environment.

Most of the questions of this assignment are steps to building a larger script which will help you throughout this course with testing your own assignment code. We suggest you start early so that getting stuck on an early question does not mean you won't be able to finish later questions.

Introductory Information: Most of this assignment is about writing a script in bash that will help you test your own programs to come. When testing your programs you'll want to have many varied test cases. Running many tests manually can be a pain, so instead you'd like to automate it. So we will use a simple system.

We will have a text file which will contain the names of all our test cases. This test file we'll call our "test set file". The contents of this file is, sort of, the names of each of our test cases separated by whitespace. For example `sample_set.txt` might look like:

```
funTest intTest floatTest
```

Then, within your current working directory you would need the files required for each testcase `test1, test2,` and `test3`. For our purposes each test will be made up of four files, an input file (`.in`), a command line arguments file (`.args`), an expected output file (`.out`), and a description file (`.desc`).

Each of the files of a test case are used to define what that test case is. The test case is meant to test *one* particular run of a program. The `.desc` file contains a description of the test case, and perhaps its purpose. The `.in` file contains everything, that if running that test manually, we would type into standard input when the program asks us for input. The `.args` files contains any command line arguments we'd like to pass to the program. Finally, the `.out` file is the expected output our program should generate when we run this test case — this is the file we'll use to check if our program passes the test case. All 4 of these files are written by you, the programmer who is testing their own program. We'll give you some samples in this assignment however.

We'll call the names of our test cases, which are in our test set files, *file stems*. That is because the names of our tests are everything that is going to come before our file extensions `.in`, `.args`, `.out`, and `.desc`. That means, for example, given the test named `funTest` shown above we would have the files `funTest.in`, `funTest.args`, `funTest.out`, and `funTest.desc` in our *current working directory* when we run our testing script.

However, the stems don't have to be only local paths — they can also be any filepath to a file stem we see fit, this is demonstrated in the first question.

1. **This question is about writing a bash script — you should not write any C code for this question.**

   In this question you are writing a bash script named `testDescribe` which expects one command line argument which is a filepath. The filepath `testDescribe` receives should be to a test set file. Here is an example of the contents of a test set file named `set1.txt`:

   ```
   test1  /home/rob/foo/test2
       ./test3
   test4
   ```

   The strings inside the test set file we'll call file *stems*. These strings are meant to represent every part of a filepath *except* for the file extension. You'll notice that the contents can be absolute or relative paths — this should not affect how you write your script.

   Your script must iterate through the contents of the test set file and for each file stem it should perform the process described below. Note that in each place the process says `stem` it means each of the strings contained within the test set file.

   (a) If no command line argument is given for the test set file print a usage message to `stderr`
   (b) If the file `stem.desc` does not exist print out the message `stem:  No test description`
   (c) If the file `stem.desc` does exist print out the contents of the file `stem.desc`
   (d) Make sure steps (b) and (c) are repeated for every `stem` in the test set file

   For example consider your current working directory contains the file `test1.desc` with the following contents:

   ```
   This test uses negative inputs
   ```

   And your current working directory contains the file `test3.desc` with the following contents:

   ```
   This test using zero as an input
   ```

   And your file system contains the file `/home/rob/foo/test2.desc` with the following contents:

   ```
   This test uses positive inputs
   ```

   Then the output of executing your script as follow `$ ./testDescribe set1.txt` would be

   ```
   This test uses negative inputs
   This test uses positive inputs
   This test using zero as an input
   test4 No test description
   ```

Hint: You may need some conditions we didn't talk about in class for your `if` statements in bash. Here's a useful website with lots of bash tips https://devhints.io/bash.

Deliverables: For this question include in your final submission zip your bash script file named `testDescribe`

2. **This question is about writing a bash script — you should not write any C code for this question.**

   In this question you will be writing a bash script `runInTests` which expects two command line arguments, the first command line argument is a command to run, and the second command line argument is a test set file (as described in question 1). Below is an example run of your script:

   ```
   ./runInTests wc wc_set.txt
   ```

   Consider the file `wc_set.txt` contains the following:

   ```
   wcTest1
   wcTest2
   ```

   Your script `runInTests` will iterate through the file stems in the test set file and perform the following set of steps for each file stem. Note in all output shown you are not meant to be printing literally `stem`, but rather the current file stem you are considering.

   (a) Run the command given to your script while redirecting input from the file `stem.in`
   (b) Compare the output from that execution of the command to the contents of the file `stem.out`
   (c) If the output does not differ, then output `Test stem passed`
   (d) If the output does differ, then output `Test stem failed`, followed on the next line by `Expected output:`, followed on the next line by the contents of `stem.out`, followed on the next line by `Actual output:`, followed on the next line by the output produced by running the given command with the given input file.

   Try out sample executable with the sample command above with the provided files to see a sample output.

   Note: If your program creates any files they *must* be temporary files. They must also be deleted once you are finished with them.

   Hint 1: Consider using the `diff` command to help you solve this problem. Remember you can read the exit status of the previous command you executed with `$?` — read the `man` pages of the `diff` command to see if the exit status could help you.
   Hint 2: If you want to run a command, but don't want the output produced by that command to print, you can redirect that commands output to `/dev/null`.

   Deliverables For this question include in your final submission zip your bash script file named `runInTests`

3. **This question is about writing a bash script — you should not write any C code for this question.**

   In this question you will be updating your script `runInTests` from question 2, your new updated script should be named `runTests`. The changes you will need to make to your previous question will be quite small if you wrote your solution well. Our sample solution only needed a change to one line.

   Update your previous solution so that when it runs each test case not only does it redirect input from `stem.in` but it also passes command line arguments to the command which are the contents of the file `stem.args`.

   For example, if one of your command was `wc` and one of the stems was `wcTest1` and the contents of the file `wcTest1.args` were as follows:

   ```
   -l -w
   ```

   Then ultimately, for that test case, you'd run the command

   ```
   wc -l -w < wcTest1.in
   ```

   Of course, this needs to be done for each test case. You are not literally writing `-l -w` in the command, as the arguments to pass are the contents of the `.args` file.

   Hint: You'll need to place the contents of each args file directly in a command — what have you seen in class that could help solve this problem?

   Deliverables For this question include in your final submission zip your bash script file named `runTests`

4. **For this question you will be writing a C program.**

   In this question you will be writing a simple C program `divisors.c`

   Your program should:

   - Read one integer from standard input.
   - Print every divisor of the read in integer, from smallest to largest, with a space inbetween every divisor. **You must not print any additional spaces, including before the first divisor or after the last divisor**.
   - Your program should print a newline after the last divisor.

   For example, if your program was executed with the input 256 it would print:

   ```
   1 2 4 8 16 32 64 128 256
   ```

   You should test your program with your `runTests` script from the previous question.

   Deliverables For this question include in your final submission zip your c file named `divisors.c`

5. **Extra exercise:** This question is not for any marks — it is additional steps you can take to make your `runTests` program more helpful and user friendly.

   - Some programs only read input, some only use command line args, some will use both. Our `runTests` looks for an `.in` and `.args` file for every single stem — we may not want that. Update your script so it only provides input or args to the command when the corresponding files exist.
   - We may have forgotten to create a given output file for our test set, right now `runTests` assumes that every `.out` file exists. Update your script so that it prints a meaningful error message when a `.out` doesn't exist.
   - Consider your first program `testDescribe`, those descriptions could be handy to view for each test case. Consider updating your `runTests` script so that when printing out if a test failed or passed it also prints out description of that test.
   - In the real world you'll have to create your expected outputs yourself since you won't have an already compiled executable of the code you're trying to write. In this class you will be given sample executables for the programs you need to write, so take advantage of this! Create a new version of your `runTests` script that takes a third argument, the sample executable, and instead of using `.out` files this version compares the output of the two executables provided for each test case.

How to submit: Create a zip file `a1.zip`, make sure that zip file contains your three bash scripts `testDescribe`, `runInTests`, `runTest`, and your C file `divisors.c`. Assuming all four of these files are in your current working directory you can create your zip file with the command

```
$ zip a1.zip testDescribe runInTests runTests divisors.c
```

Upload your file `a1.zip` to the a1 submission link on eClass.