# CMPUT275—Assignment 1 (Winter 2026)

R. Hackman

Due Date: Friday January 30th, 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 likely will not work on your local machine, you should use them on the student environment if this is the case, which is where you should be testing your own code anyway.
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.

Finally, in all questions in which C programming is performed the student must compile with the flags `-Wall` `-Wvla` and `-Werror`, which turn on some important warnings as well as treat warnings as errors, which means code with warnings will not compile. This is how your code will be compiled when we are testing it, and if your code does not compile then your solution cannot be executed which results in a zero. As such, make sure you are *always* compiling with these flags. In order to help you do so, you may want to add the following line to the bottom of your `.bashrc` file in your home directory on the student server:

```
alias gcc="gcc -Wall -Wvla -Werror"
```

1. **This question is about connecting to the student server and running programs on it. This is essential to this course as you must *always* test your assignment solutions on the student server before submitting to make sure they work correctly!**

   First, you must connect to the student server. As discussed in class you can connect to the student server using the `ssh` command.

   Once you have connected to the student server you should clone the course repository on the student server if you have not done so already, so that you have access to it there as well as your local machine.

   Inside the assignment 1 folder there is a program named `generateCodeword`. You must run this program on the student server, and if executed correctly it will print out a single word. Make a plaintext file named `a1q1.txt` that contains the output of the `generateCodeword` program and include that text file in your submission. This code word is unique for each student, so you cannot have a friend run the program for you. The purpose of this question is to make sure *you* are able to connect to the student server so that you are set up to properly test assignment solutions!

**Deliverables:** For this question include in your final submission zip your bash script file named `a1q1.txt`

2. **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 text file whose contents are a series of strings separated by whitespace. We'll call this file 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 command line argument is given, then it should be assumed that it is a test set file. For each of the strings in the test set files (which represent file stems) you should perform the following process:
      i. Print out a message of the form '`Description for test case <stem>:`' followed by a newline.
      ii. If the file `stem.desc` does not exist print out the message '`<stem>: No test description`'
      iii. If the file `stem.desc` does exist print out the contents of the file `stem.desc`
      iv. In the above steps when a string literal contains '`<stem>`' note that it means to replace that with the actual stem currently being processed.

   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

```
Description for test case test1:
This test uses negative inputs
Description for test case /home/rob/cmput275/a1/testDesc/test2:
This test uses positive inputs
Description for test case test3:
This test using zero as an input
Description for test case test4:
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`

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 writing a bash script `runInTests` which expects two command line arguments, the first command line argument is a command to run (which may be a path to a program), 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 each step you should consider `stem` is a variable that represents any given file stem:

   (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`

4. **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`

5. **Credit Card Verification**

   If you've ever attempted to purchase an item online with a credit card, you may have accidentally entered your credit card information incorrectly. Upon entering your credit card information incorrectly you may have immediately been informed that the provided credit card was not valid. Have you ever wondered how, out of all the possible credit card numbers, the online store immediately knew the number you entered was not a valid credit card? This is because credit card numbers are verified with an algorithm known as *Luhn's Algorithm.* This is a very simple *checksum* algorithm that once could choose to use to specify what set of values are valid versus those that are not. Note, Luhn's algorithm does not provide any security it is simply a way to quickly validate if a number is possibly valid, and the major credit card companies all use it to do so (so all valid credit card numbers from these companies meet the checksum determined by Luhn's algorithm).

   Given an "account number" Luhn's algorithm verifies the number is potentially valid with a simple checksum. given an account number of $n$ digits of the form $d_1d_2d_3d_4...d_n$ then the following procedure is followed.

   (a) The nth digit is the check digit, and is only used at the end to verify validity.

(b) Moving left from the nth digit each digit has its value added to a sum. However, we follow a different procedure for every second digit.

(c) The first digit to the left of our check digit has its value doubled, if that product is larger than or equal to 10 then 9 is subtracted from the product to give us the value to add to our sum.

(d) The next digit to the left just has its value added to our sum.

(e) Repeat that process moving left until there are no more digits: doubling one subtracting 9 if necessary, then not doubling the next.

(f) Once the sum is calculated it should be multiplied by 9, the result of this product should have a remainder when divided by 10 that is equal to the check digit.

For example, if we want to check if an account number `34682` is valid, we would do the following procedure:

- Our last digit is our check digit, it is 2.
- The next digit to the left is 8, so since this is the first digit to the left we multiply it by 2 which leaves us with 16. Since 16 is larger than or equal to 10 we subtract 9 from it leaving us with 7 to add to our sum.
- The next digit to the left is 6, since it follows a digit we just doubled we do not double it and simply add 6 to our sum.
- The next digit to the left is 4, since we didn't double the last digit we now double this one leaving us with 8 to add to our sum.
- The next digit to the left is 3, since it follows a digit we just doubled we do not double it and simply add it to our sum.
- So our sum is now 7+6+8+3=24. 24*9=216, the remainder when dividing 216 by 10 is 6, so since our check digit is not 6 this account number is *invalid.*

For this question you must complete the program `luhns.c` which reads digits from the standard input stream until the first non-digit character is received (indicating the end of the account number). Your program should print `Valid` if the account number read in passes Luhn's algorithm and should print `Invalid` if the account number does not pass Luhn's algorithm.

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

**Constraints:** For this question you are not allowed to use arrays (on the heap or stack), nor are you allowed to use recursion. Use of arrays or recursion of any kind will result in 0 on this question. You do not need an array to solve this question, and part of the challenge of this question is thinking of how to tackle the task without using an array.

**Deliverables** For this question include in your final submission zip your c source code file named `luhns.c`

6. **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 codeword file `a1q1.txt`, three bash scripts `testDescribe`, `runInTests`, `runTest`, and your C file `luhns.c`. Assuming all five of these files are in your current working directory you can create your zip file with the command

```
$ zip a1.zip a1q1.txt testDescribe runInTests runTests luhns.c
```

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