Hints

Note that questions 3-5 all follow the same error-handling in terms of validating the command-line arguments for each of `produceOutputs` and `runSuite`. **Use the provided sample executables to determine how your scripts should handle files that do not exist, or cannot be opened for reading, or cannot be opened for writing.** The error messages and exit values do not have to be exactly the same, you just need to provide reasonable messages and zero or non-zero exit values, as appropriate.

3. **5 marks.** Write a `bash` script called `produceOutputs` that is invoked as follows:

```
./produceOutputs suite-file program
```

The argument `suite-file` is the name of a file containing a list of filename stems (more details below), and the argument `program` is the name of a program to be run.

The `produceOutputs` script runs `program` on each test in the test suite and, for each test, creates a file that contains the output produced for that test.

The file `suite-file` contains a list of stems, from which we construct the names of files containing the command-line arguments used by each test. Stems will not contain spaces. For example, suppose our suite file is called `suite.txt` and contains the following entries:

```
test1 test2
reallyBigTest
```

Then our test suite consists of three tests. The first one (`test1`) will use the file `test1.args`. The second one (`test2`) will use the file `test2.args`. The last one (`reallyBigTest`) will use the file `reallyBigTest.args`.

A sample run of `produceOutputs` would be as follows:

```
./produceOutputs suite.txt ./myprogram
```

The script will then run `./myprogram` three times, once for each test specified in `suite.txt`:

- The first time, it will run `./myprogram` with command-line arguments provided to the program from `test1.args`. The results, captured from standard output, will be stored in `test1.out`.
- The second time, it will run `./myprogram` with command-line arguments provided to the program from `test2.args`. The results, captured from standard output, will be stored in `test2.out`.
- The third time, it will run `./myprogram` with command-line arguments provided to the program from `reallyBigTest.args`. The results, captured from standard output, will be stored in `reallyBigTest.out`.

Note that if the test suite contains a stem but the corresponding `.args` file is not present, the program is run without providing any command-line arguments.

Your script must also check for incorrect number of command-line arguments to `produceOutputs` and invalid command-line arguments as described in the preceding "Hints" section. If such an error condition arises, print an informative error message to standard error (the exact message is up to you) and abort the script with a **non-zero** exit status (exact value is up to you). **Use the provided executable** `q3produceOutputs` **to see how your program should behave for valid test cases.**

Note on purpose of this script: This script will be useful in situations where we provide you with a binary version of a program (but not its source code) that you must implement. By creating your own test cases (`.args` files) and then using this script to produce the intended output you will have something to compare with when you implement your own solution (see the next question for how to automate the comparisons). Only the program being run will be able to tell if the provided command-line arguments are correct or not. It is not something that `produceOutputs` can determine.

4. **10 marks.** Create a `bash` script called `runSuite` that is invoked as follows:

   ```
   ./runSuite suite-file program
   ```

   The argument `suite-file` is the name of a file containing a list of filename stems (more details below), and the argument `program` is the name of the program to be run.

   In summary, the `runSuite` script runs `program` on each test in the test suite (as specified by `suite-file`) and reports on any tests whose output does not match the expected output (created either manually or through use of `produceOutputs`). Note that if all of the tests are passed, no output will be produced by the script.

   The file `suite-file` contains a list of stems, from which we construct the names of files containing the command-line arguments and expected output of each test. Stems will not contain spaces. For example, suppose our suite file is called `suite.txt` and contains the following entries:

   ```
   test1 test2
   reallyBigTest
   ```

   Then our test suite consists of three tests. The first one (`test1`) will use the file `test1.args` to hold its command-line arguments, and `test1.out` to hold its expected output. The second one (`test2`) will use the file `test2.args` to hold its command-line arguments, and `test2.out` to hold its expected output. The last one (`reallyBigTest`) will use the file `reallyBigTest.args` to hold its command-line arguments, and `reallyBigTest.out` to hold its expected output.

   A sample run of `runSuite` would be as follows:

   ```
   ./runSuite suite.txt ./myprogram
   ```

   The script will then run `./myprogram` three times, once for each test specified in `suite.txt`:

   - The first time, it will run `./myprogram` with command-line arguments provided to the program from `test1.args`. The results, captured from standard output, will be compared with the contents of `test1.out`.
   - The second time, it will run `./myprogram` with command-line arguments provided to the program from `test2.args`. The results, captured from standard output, will be compared with the contents of `test2.out`.
   - The third time, it will run `./myprogram` with command-line arguments provided to the program from `reallyBigTest.args`. The results, captured from standard output, will be compared with the contents of `reallyBigTest.out`.

   Note that if the test suite contains a stem but a corresponding `.args` file is not present, the program is run without providing any command-line arguments.

   If the output of a given test case differs from the expected output, print the following to standard output (assuming test `test2` failed):

   ```
   Test failed: test2
   Args:
   (contents of test2.args, if it exists)
   Expected:
   (contents of test2.out)
   Actual:
   (contents of the actual program output)
   ```

   with the *(contents ...)* lines replaced with actual file contents, *without any changes*, as described. The literal output `Args:` must appear, even if the corresponding file does not exist. Note that there is no whitespace after the colon for each of `Args:`, `Expected:`, and `Actual:` except for the newline character.

   Use the provided executable `q4runSuite` to see how your program should behave for valid test cases.

   Follow these output specifications *very carefully*. You will lose a lot of marks if your output does not match them. If you need to create temporary files, create them in `/tmp`, and use the `mktemp` command to prevent name duplications. **Also be sure to delete any temporary files you create in `/tmp`.**

Note: Do **NOT** attempt to compare outputs by storing them in shell variables, and then comparing the shell variables. This is a very bad idea, and it does not scale well to programs that produce large outputs. We reserve the right to deduct marks (on this and all assignments) for bad solutions like this would be.

You can get most of the marks for this question by fulfilling the above requirements. For full marks, your script must also check for the following error conditions:

- incorrect number of command-line arguments to `runSuite` as described in the preceding "Hints" section
- incorrect permissions on the command-line arguments to `runSuite` as described in the preceding "Hints" section
- if a `.args` file exists but isn't readable,
  (a) produce an error message,
  (b) do not run the test,
  (c) do not compare the outputs i.e. just move on to the next test, and
  (d) ensure that the script will eventually terminate with a non-zero exit status
- missing or unreadable `.out` file (for example, the suite file contains an entry `xxx`, but `xxx.out` doesn't exist or is unreadable), in which case print an informative error message to standard error and terminate the script with a **non-zero** exit status.

5. **15 marks.** In this question, you will generalize the `produceOutputs` and `runSuite` scripts that you created in questions 3 and 4. As they are currently written, these scripts cannot be used with programs that take input from standard input. For this problem, you will enhance `produceOutputs` and `runSuite` so that, in addition to (optionally) passing command-line arguments to the program being executed, the program can also be (optionally) provided input from standard input. The interface to the scripts remains the same:

```
./produceOutputs suite.txt ./myprogram
./runSuite suite.txt ./myprogram
```

The format of the suite file remains the same. But now, for each `testname` in the suite file, there might be an optional `testname.in` in addition to, or instead of `testname.args`. If the file `testname.in` is present, then the script (`produceOutputs` or `runSuite`) will run `myprogram` with the contents of `testname.args` passed on the command-line as before and the contents of `testname.in` used for input on `stdin`. If `testname.in` is not present, then the behaviour is almost identical to question 3/4 (see below for a difference in the output).

The output of `runSuite` is changed to now also show the input provided to a test if the test failed. Assuming test `test2` from Q4 failed, the output generated by the updated `runSuite` is as follows:

```
Test failed: test2
Args:
(contents of test2.args, if it exists)
Input:
(contents of test2.in, if it exists)
Expected:
(contents of test2.out)
Actual:
(contents of the actual program output)
```

with the *(contents ...)* lines replaced with actual file contents, as described. The literal output `Args:` and `Input:` must appear, even if the corresponding files do not exist. Note that there is no whitespace after the colon for each of `Args:`, `Input:`, `Expected:`, and `Actual:` except for the newline character. **Use the provided executables `q5produceOutputs` and `q5runSuite` to see how your programs should behave for valid test cases.**

All error-checking that was required in questions 3 and 4 is required here as well.

(a) Modify `produceOutputs` to handle input from standard input.

(b) Modify `runSuite` to handle input from standard input.

Note: To get this working should require only very small changes to your solution to questions 3 and 4.