Assignment 4

Adding Redirection to your Shell

Author: Brian Crites, Mike Izbicki

This project must be done in a group of two

Coding Requirements

Extend your `rshell` program so that it properly handles input redirection `<`, output redirection `>` and `>>`, and piping `|`. This will require using the Unix functions `dup` and `pipe`. You can find help on how to use these functions in the man pages.

As an example, after this assignment, your program should be able to successfully handle the following command:

```
$ cat < existingInputFile | tr A-Z a-z | tee newOutputFile1 | tr a-z A-Z > newOutputFile2
```

> **Note:** This is a necessary but not sufficient test case. You must come up with others on your own.

Bash has an extensive syntax for redirection, and you are not required to implement all of it but only `<`, `>`, `>>`, and `|` as listed above. But if you're curious, see the linux documentation project's bash io-redirection tutorial for details. Below are some additional materials that previous students have found useful:

- Video: fd, dup, dup2 system calls
- Video: pipe tutorial for linux
- Video: syscalls for dummies: dup

Project Structure

You must have a directory called `src/` which contains all the source (.cc/.cpp) files for the project. For header files you may either have a folder `header/` which contains your header files or you may keep them in the `src/` directory. You must have a `unit_tests/` directory which should contain all the unit tests you've written using the Google Unit Test framework (the main for the unit tests can be located either in `unit_tests/` or `src/`). You must also have an `integration_tests/` directory which should contain all the integration tests you've written using bash. We recommend using IO redirection for your bash integration tests, and automated validation is not necessary.

Your root directory must have a `CMakeLists.txt` with two targets. The first target is named `rshell` and should build the main executable and the second should be `test` which runs the unit tests that you have created using the Google Unit Test framework.

> **Note:** The file/directory names above are a standard convention. You must use the exact same names in your project, including capitalization. We utilize these names when performing steps of our automated grading process, so any deviation may result in missing points.

The google test code for your unit tests can have any name as long as the files reside in the `unit_tests/` directory and an executable is generated named test in the root directory. The integration test shell scripts that you develop and place into the `integration_tests/` directory must have the following names:

`input_redirection_tests.sh`	tests primarily for the test commands execution
`output_redirection_tests.sh`	tests primarily for the symbolic tests commands execution
`pipe_operator_tests.sh`	tests primarily for the precedence being respected

> **Note**: the above tests should not only validate that the newly created functionality works correctly but also that it works with the features that were developed during Assignments 2 and 3.

Your project should not contain any files not necessary for the project. This includes things like CMake temporary build files used for building the project on your machine such as `CMakeCache.txt` and the `CMakeFiles/` directory as well as executables. We have provided a `.gitignore` in the template repository which will stop may of these files from showing as untracked when you run git status, and you should extend this file with additional temporary and machine specific files.

Submission Instructions

You will also need to add an **annotated** `hw4` tag to the commit that you want to be graded. Annotated tags are described in lab 2 and in the git documentation. `git push` will not automatically push tags to your repository. Use `git push origin hw4` to update your repository to include the `h4` tag. If you need to update your tag, you can remove the old tag using `git push --delete origin hw4` and push an updated one to your repo.

Note: We will test your project on the hammer server, so while it is not required that you develop on hammer you should verify that your project builds and functions properly on hammer.cs.ucr.edu.

Your project must also contain a names.txt file in the root directory which contains the name, SID, and email of each of the partners in your group. It should have the following format.

```
Brian Crites, 860XXXXXX, bcrit001@ucr.edu
Andrew Lvovsky, 860XXXXXX, alvov001@ucr.edu
```

Do not wait to push your assignment to Github until the project due date. You should be committing and uploading your assignment continuously. If you wait until the last day and can't figure out how to use git properly, then you will get a zero on the assignment or be forced to use the course late policy. No exceptions.

For late assignment submissions, a penalty of 10% will be deducted for every day the assignment is late up to a maximum of 3 days, with exceptions only for documented emergencies. For clarity, this means that an assignment that is 3 days late would incur a 30% penalty to the grade **assessed**. Please submit this form along with your late submission. Otherwise, the team will receive a zero on that assignment.

In addition to the above submission requirements you must update your `README.md` file so that it specifies the current structure of your program. This is especially important since your final program will likely be much different than what you originally designed in Assignment 1.

Testing Your Submission

Your assignment 4 should have approximately the following structure, see the project structure section for more details:

example-project-directory/

```
├── CMakeLists.txt
├── README.md
├── googletest
├── header
│   ├── header_file_1.h
│   └── header_file_2.h
├── src
│   ├── src_file_1.cpp
│   └── src_file_2.cpp
└── unit_tests
```

```
        ├── test_file_1.cpp
        └── test_file_2.cpp
└── integration_tests
        ├── input_redirection_tests.sh
        ├── output_redirection_tests.sh
        └── pipe_operator_tests.sh
```

You should also have a .gitignore and a .gitmodules hidden file in your root directory. Your root directory should contain no C++ header or source files and your googletest directory should be added to your project as a submodule. While the googletest directory is not required to be present in the root directory it is suggested.

When testing your project we will run (approximately) the following commands on hammer:

```
git clone <assignment-repo-url>
cd <assignment-repo-url>
git checkout tags/hw4
git submodule init
git submodule update
cmake3 .
make
test -e rshell || echo "rshell executable missing, check submission instruction
section of the specifications"
test -e test || echo "test executable missing, check submission instruction
section of the specifications"
```

These commands should generate a `test` executable as well as an `rshell` executable in the root directory, otherwise will print that they are missing. **Please run these commands on hammer with a brand new clone** of your GitHub repository to ensure your shell compiles and builds properly before submission.

Collaboration Policy

- You **may not** copy any code found online, doing so will result in failing the course.
- You **may not** look at the source code of any other student.
- You **may** discuss with other students in general terms how to use the unix functions.
- You are **encouraged** to talk with other students about test cases and are allowed to freely share ideas in this regard.

Grading

10	Sufficient Unit Test Cases

10	Sufficient Integration Test Cases
10	Updated README.md
20	Input Redirection Implementation
25	Output Redirection Implementation
25	Piping Implementation
100	**Total**

Note: Your project structure and executable names are not explicitly listed in the grading schedule above but not following the specified file names and location may result in losing points because the automated grading system is unable to process your repository. Loss of points due to an issue specified above are final and failure to build and compile will result in a zero for the assignment.