CS111 - Project 0: Warm-Up

INTRODUCTION:

Every quarter a few students come into this course without basic C software development skills, and invest a great deal of time and frustrating effort in the first two projects before concluding they will fail the course and must drop. We have created this simple warm-up to determine whether or not students are prepared to work on C programming projects. Most students should find this project to be completely trivial (20 minutes of work). If you do not find this project to be trivial, you may want to reconsider whether or not you are ready to take this course.

RELATION TO READING AND LECTURES:

None. This project requires only skills that incoming students should already possess.

PROJECT OBJECTIVES:

- ensure students have a working Linux development environment.
- ensure students can access and understand Linux API documentation.
- ensure students can code, compile, test and debug simple C programs.
- introduce and demonstrate the ability to use basic POSIX file operations.
- introduce and demonstrate the ability to process command line arguments.
- introduce and demonstrate the ability to catch and handle run-time exceptions.
- introduce and demonstrate the ability to return informative exit status.
- demonstrate the ability to construct a standard Makefile.
- demonstrate the ability to write software that conforms to a Command Line Interface (CLI) specification.

DELIVERABLES:

A single tarball (`.tar.gz`) containing:

- a single C source module that compiles cleanly (with no errors or warnings).
- a Makefile to build the program and the tarball.
- two screen snapshot(s) from a gdb(1): one showing a segfault and associated stack-trace, the second showing a breakpoint and variable inspection.
- a README file describing each of the included files and any other information about your submission that you would like to bring to our attention (e.g. limitations, features, testing methodology, use of slip days).

PROJECT DESCRIPTION:

1.(if you do not already have one) bring up a Linux development environment. If you do not already have one and do not know how to do this, the local Linux

Users Group can help you. Your development environment should include (at least):

- gcc
- libc (e.g. glibc or libc6-dev)
- make
- gdb

If do not want a personal Linux development environment, you can also work directly on a SEASNET server.

2.(if you are not already familiar with them) study the following manual sections:

- POSIX file operations ... open(2), creat(2), close(2), dup(2), read(2), write(2), exit(2), signal(2), and this brief tutorial on file descriptor manipulation.
- perror(3) ... a function that interprets the error codes returned from failed system calls.
- getopt(3) ... the framework we will use for argument handling in all projects for this course.

Feel free to seek out other examples/tutorials for these functions, but make sure you cite those sources in your README.

3.write a program that copies its standard input to its standard output by read(2)-ing from file descriptor 0 (until encountering an end of file) and write(2)-ing to file descriptor 1. If no errors (other than EOF) are encountered, your program should exit(2) with a return code of 0.

Your program executable should be called lab0, and accept the following command line arguments (in any combination):

- --input=filename ... use the specified file as standard input (making it the new fd0).
  If you are unable to open the specified input file, report the failure (on stderr, file descriptor 2) using fprintf(3) and perror(3), and exit(2) with a return code of 1.
- --output=filename ... create the specified file and use it as standard output (making it the new fd1).
  If you are unable to create the specified output file, report the failure (on stderr, file descriptor 2) using fprintf(3) and perror(3), and exit(2) with a return code of 2.
- --segfault ... force a segmentation fault by setting a char * pointer to NULL and then storing through it.
- --catch ... use signal(2) to register a SIGSEGV handler that catches the segmentation fault, logs an error message (on stderr, file descriptor 2) and exit(2) with a return code of 3. (Note: catch should not trigger the segmentation fault.)

4.create a Makefile that supports the following targets:

- (default) ... build the lab0 executable.
- check ... runs a quick smoke-test on whether or not the program seems to work, supports the required arguments, and reports success or failure.

Please include a brief description (in your README) of what checks you chose to include in your smoke-test.

•clean ... delete all makefile-created files to return the directory to its just installed state.

•dist ... build the distribution tarball.

5. run your program (with the --segfault argument) under gdb(1)
   - •take the fault
   - •get a stack backtrace
   - •take a screen snapshot (to be included with your submission)
6. run your program (with the --segfault argument) under gdb(1)
   - •set a break-point at the bad assignment
   - •run the program up to the breakpoint
   - •inspect the pointer to confirm that it is indeed NULL
   - •take a screen snapshot (to be included with your submission)

VALUE:

Points for this project will be awarded as follows:

Value	Feature
	packaging and build
5%	untars expected contents
5%	clean build w/default action
2%	correct `make check`
1%	correct `make dist`
2%	reasonablness of `README contents`
5%	reasonablness of smoke test
	input/output features
10%	correctly copy input to output
10%	correctly implements `--input`
10%	correct handling of non-existant input file
10%	correctly implements `--output`
5%	correct handling of un-openable/creatable output file
5%	implements combined `--input` + `--output`
	fault handling
5%	generate (and die from) SIGSEGV
5%	catch and report SIGSEGV
	gdb use
5%	screen shot showing taking of segfault within gdb
5%	screen shot showing backtrace from segfault
5%	screen shot showing breakpoint stop before fault
5%	screen shot showing inspection of null pointer

SUBMISSION:

Your tarball should have a name of the form lab0-yourStudentID.tar.gz. and should be submitted via CCLE.

We will test it on a SEASnet GNU/Linux server running RHEL 7. You would be well advised to test all the functionality of your submission on that platform before submitting it.