# CS320 Assignment #4

## Purpose

This assignment is designed to help you experience Lua and C and implementing Multilanguage projects.

## Requirements

This assignment consists of six major requirements.

1) Developing solutions to the three problems below.
2) Ensuring that your programs run correctly using the tools **on rohan**.
3) Documenting your solutions correctly.
4) Organizing your *handin* directory and submissions correctly.
5) Turning in a correctly documented hard copy.
6) Your hard copy must have the rubric stapled as your first page.

## Problem

1) There has been a recent surge in applicants to your software engineering firm. Your boss has recently instated a simple programming test to weed out the applicants who can't program. Your job is to automate the task of running the submitted programming tests. To accomplish this you know that you can write a shell script that will take the applicants file as a command line argument and a 'Correct' output file as another command line argument and compare the output from the applicants program with the correct output. Unfortunately for you all of the submitted programs are written in Lua and your boss won't let you install Lua on your server. To get around this you know that you can write your own Lua execution environment in C. The final step will be to write some Lua code to verify that the whole setup is working.

   First, we must be able to execute some Lua code. To do this we are going to build a basic Lua interpreter. We know that we do not have to have a full-fledged interpreter, and that it only has to tell the lua environment to do the file that we will have passed as the first command line argument after the environment has been setup. Something along the lines of a very basic environment as described in the lua documentation[1] should do it. Probably wont need anything loopy and a single luaL_dofile[2] call will be sufficient.
   This C program will be your Assignment #5-1.
   Unfortunately compiling this thing is quite a task, as nothing that it uses is standard or installed on the server. Fortunately your buddy has done most of the hard work for you getting this to

[1] http://www.lua.org/pil/24.1.html **NOTE:** We are using lua 5.3 on the server and 5.1 should be the minimum target.
[2] http://www.lua.org/manual/5.2/manual.html#luaL_dofile We can ignore any errors.

work. (If you want to figure out how to build lua on Rohan yourself it really isn't very hard, but the compiler flags below should still target my directory for your final bash script).

To get this bad boy to compile there are going to be a total of four(4) new compiler flags and one old one. It is ugly, long and complicated. I will try to explain what is going on after:

```
gcc <FILE.C>  -l lua –l m –l dl -L <Directory to Lua Buil> -I
<Directory to Lua Buil>
```

This should all be on one line, but it doesn't fit. Basically it looks like this:

```
gcc <FILE.C>  -llua -lm –ldl -L <LDIR> -I <HDIR>
```

The <FILE.C> is the file we want to compile. The `–L <LDIR>` bit tells the compiler where to find the compiled library files such that the linker can find the symbols and use the right memory addresses for them. The -llua portion tells the linker to do this. The -lm portion tells it to link to the default math library. The `–I <HDIR>` portion tells the compiler where to find those fancy headers we found/used in the reference manual. The -R <SODIR> bit tells the compiler to add some extra fancy flags to our executable. These flags tell the executable where to find non-default (not installed) dynamically linked libraries (.so on *nix & .dll on windows). This is one of those nifty flags that allows us to ship programs that rely on specific libraries that might not be installed on a target system as long as we bring our own version of the library with our executable code.

Second, we are going to need a simple lua program to test our C code. This Lua file should be the test that your boss has set out for your applicants. What was that test? Oh, right. FizzBuzz from 1 to 100. You don't know what FizzBuzz is?[3]
This Lua script should be your Assignment #5-2.
You should be able to test your Lua code by running your C program above with the lua file as the first argument.

Third, we are going to write a shell script that will compile our C code and then execute the file passed as the first command line argument. If the output from the code does not match the contents of the file that is passed as the second argument then your script should print "Failed Test" otherwise it should print "Passed Test".
This script should be your Assignment #5-3.

Yes. The Assignment header prints are going to be ridiculous.

[3] http://c2.com/cgi/wiki?FizzBuzzTest There are so many solutions to fizzbuzz around you might find one to copy… but seriously…. This problem is really easy. Just do it.

2) You must ensure that your code can be run on rohan using the shell tools of your choice. You may develop your solution on any machine you desire, as long as the final solution works on rohan.
3) Your solution must have a complete comment header as is detailed below. During runtime, each of your solutions to section 1 must output a correct **title string** as the first line printed. It should be in this format:

   `Assignment #1-1, <NAME>, <MASC>`

   Each problem should follow this format, with the second number incremented for each problem in part 1. For example the second problem in part one should have the title string:

   `Assignment #1-2, Scott Lindeneau, masc1234`

4) You must place a copy of your solutions inside a folder named **`handin`** in your home directory. You assignment will be placed in a folder inside of the handin directory named according to the assignment. For example, for Assignment #3, you will place your solutions in `~/handin/assignment4/`. Your files **MUST** be named **`prog4_1.c and prog4_2.lua and prog4_3.sh`** and must not be modified after the turn in time. The modified timestamp for each file in your **`handin`** folder will be used as the submission time. If it is after the due date, it will be counted as late. **DO NOT** turn in any compiled files.
5) You **MUST** bring a hardcopy of your program to class on the due date. Your hardcopy must be stapled (if more than one page) and it **MUST** be correctly documented as per section 3. Late hardcopies will be considered late turn in for the assignment.
6) You **MUST** staple a hardcopy of the attached Rubric to your physical copy as the **TOP** page of your submission.

## Example Output

Testing just 5-1 with a test.lua file which only contains the line: print("Hello")

./a.out test.lua
Assignment #4-1, Scott Lindeneau, slindeneau@gmail.com
Hello

Testing 5-2 with 4-1 as the interpreter.
./a.out prog4-2.lua
Assignment #4-1, Scott Lindeneau, slindeneau@gmail.com
Assignment #4-2, Scott Lindeneau, slindeneau@gmail.com
1
2
Fizz
4
Buzz
…. (Output continues to completion)

Testing 4-3 with 4-2 and 4-1.
./prog4-3.sh prog4-2.lua correct.output
Assignment #4-3, Scott Lindeneau, slindeneau@gmail.com
Passed Test

(You will need to generate a valid correct.output file for your own tests)

## Additional Details

- You do not need to do input checking. You can assume that the input value is a valid input.

## Late Policy

Late programs will be accepted with a penalty of 5% per day for seven days after due date.

## Cheating Policy

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. (Remember that you can get help from me. This is not cheating, but is in fact encouraged.) I will examine your code carefully. Anyone caught cheating on a programming assignment or on an exam will receive an "F" in the course, and a referral to Judicial Procedures.

## Extra Credit

The problem should be titled **Assignment #4-4** and saved in a file **prog4-4.c**

Extra credit is graded on a complete/incomplete basis and will be added to your class total at the end of the semester.

Write a simple lua interpreter that takes commands from standard in (no command line arguments). This interpreter must be able to accept multi-line `for do end` loops and `if then else end` statements. You don't need to go full blown lexical analysis / parser on it. Just solve those two problems.