Systems and Networks	C Bootcamp Project 1: Basic C Usage
CS3600 Fall 2014	July 28, 2014

This C bootcamp project is due at 11:59:59pm on September 4, 2014 and is worth 1.25% of your grade. You must complete it by yourself, and no slip days may be used.

1 Preliminaries

As this is the first project of the semester, there are two steps that you *must* complete prior to turning in this assignment. If you do not complete these, the course staff will not be able to determine whose submitted project is whose, and you will receive a 0 for the project. That would be unfortunate.

1.1 Registering with the turnin system

We will be using the CS3600 turnin system in order to submit assignments this semester. If you have not already done so, you must register for the system with your student ID. To do so, `ssh` into any CCIS Linux machine and execute

```
bash$ /course/cs3600f14/bin/register-student ID
About to register user 'USER' with student ID 'ID'.  Is this correct? [yn]
```

where `ID` is your student ID, with any leading 0s removed. For example, if your NEU student ID is 003044, you would enter

```
bash$ /course/cs3600f14/bin/register-student 3044
```

Double-check that the userid and student ID are correct; if so, type `y` and `Enter`. If not, type `n` and `Enter`. If it was successful, you will see the message

```
bash$ /course/cs3600f14/bin/register-student ID
About to register user 'USER' with student ID 'ID'.  Is this correct? [yn] y
User 'USER' is now successfully registered for CS3600 with student ID 'ID'.
bash$
```

If you see any other message (in particular, a message with "Error" in it), it has not succeeded. Email the instructor with the error message if you are not able to diagnose the problem.

1.2 Registering with Piazza

We will be using the Piazza bulletin board system for questions, problems, and announcements throughout the semester. You will be expected to check it at least every few days, participate actively, and use it as a first place to post questions related to the projects or homeworks in this course. However, you have to register for it. If you have not done so, please do so by going to `http://piazza.com/northeastern/spring2014/cs3600` and registering as a student.

2 Description

There are several goals for this assignment. First, you will get a chance to familiarize yourself with the C programming language. Second, you will learn how to use the CCIS Linux resources, and with the turnin and grading scripts we will be using for the semester. Your assignment is to complete four C problems. For each of the problems, we give you starter code and ask you to fill in various functions using the skills you learned in the first day of C Bootcamp. Note that you should **only** make changes inside of the functions marked with TODO; you should not change any other code.

2.1 SSH

Your project is designed to be run on the CCIS Linux machines; you're welcome to work on your own machine, but it must compile and run without any problems on CCIS Linux machines when graded. If you are not familiar with SSH, you can either work on one of the Linux machines in the WVH 102 lab, or you can learn about SSH (Google is your friend). Free SSH clients exist for Windows (e.g., PuTTY), Mac OS X (built-in Terminal application), and Linux (built-in terminals).

2.2 Starter code

The starter code for the assignment is available in `/course/cs3600f14/code/c-project1`. You must use this code as a basis for your project. Provided is a `cp1-problem[1,2,3,4].c` code and a `Makefile`. The `Makefile` is configured to correctly compile your code and to test your code on sample input.

To get started, you should copy down this directory into your own local directory. You should first `ssh` over to any CCIS Linux machine (e.g., `bubbles.ccs.neu.edu`). We recommend doing the following, making a directory in your CCIS home directory for all of your CS3600 work, and then copying down the C Bootcamp Project 1 code:

```
bash$ mkdir ~/cs3600
bash$ cp -r /course/cs3600f14/code/c-project1 ~/cs3600
bash$ cd ~/cs3600/c-project1
```

3 Completing your project

This project consists of four problems, each located in `cp1-problem[1,2,3,4].c`. Each file contains documentation for what is expected, and you should only add code where a TODO line exists. If you are unsure of any specifications (e.g., what to do in corner cases), first consult the documentation before asking on Piazza.

3.1 Compiling your code

You can compile the starter code by running `make`. This will try to compile all four of your homework problems, in order. If the compiler raises an error, it will print the error to the terminal. A successful compile looks like

```
bash$ make
gcc -std=c99 -O0 -g -Wall -pedantic -Werror -o cp1-problem1 cp1-problem1.c
gcc -std=c99 -O0 -g -Wall -pedantic -Werror -o cp1-problem2 cp1-problem2.c
gcc -std=c99 -O0 -g -Wall -pedantic -Werror -o cp1-problem3 cp1-problem3.c
gcc -std=c99 -O0 -g -Wall -pedantic -Werror -o cp1-problem4 cp1-problem4.c
bash$
```

At this point, you will have executable files `cp1-problem1`, `cp1-problem2`, ... in your directory. An unsuccessful compile might look something like

```
bash$ make
gcc -std=c99 -O0 -g -Wall -pedantic -Werror -o cp1-problem1 cp1-problem1.c
cp1-problem1.c: In function ÔmainÕ:
cp1-problem1.c:32: error: too few arguments to function ÔfactorialÕ
cp1-problem1.c: At top level:
cp1-problem1.c:55: error: conflicting types for ÔfactorialÕ
cp1-problem1.c:14: note: previous declaration of ÔfactorialÕ was here
make: *** [cp1-problem1] Error
bash$
```

You will have to learn how to diagnose and fix C compile errors (note that the line numbers are included in error messages, and a short description of the error is provided). If you can not understand the cause of an error message, try searching on Google or posting on Piazza.

If you wish to only compile one of the homework problems (rather than them all), you can run

```
bash$ make cp1-problem2
gcc -std=c99 -O0 -g -Wall -pedantic -Werror -o cp1-problem2 cp1-problem2.c
bash$
```

3.2 Running your code

One you have successfully compiled your project, you will have executable files `cp1-problem1`, `cp1-problem2`, ... in your directory. You can run each of them by running

```
bash$ ./cp1-problem1 <args>
```

You can also delete any compiled code and object files by running `make clean` (this will not delete your source code, don't worry).

3.3 Testing your code

We have included basic test scripts to check the output of your code against our reference solution. To run the test script, simply type

```
bash$ make test
```

This will compile your code and then test all four problems against the reference solution. If any errors are detected, the test will stop and an error will be printed. For example, you might see something like

```
bash$ make test
./test cp1-problem1 cp1-problem2 cp1-problem3 cp1-problem4
Testing cp1-problem1
  Trying with './cp1-problem1 1'                              [FAIL]
    Expected output:
Success: The factorial of that number is: 1

    Received output:
Success: The factorial of that number is: 0

make: *** [test] Error 255
bash$
```

This indicates that the test with input `./cp1-problem1 1` was expected to return 1, but instead returned 0 (note the difference between the expected output and the received output). We include a few sample tests, but these are by no means exhaustive. We expect that you will create additional test to ensure that your programs behave as expected.

3.4 Debugging your code

You should also strive to learn how to use the GNU debugger. We'll go over this more in class later in the week, but if you'd like to get started, a good quick reference is available at `http://www.csl.mtu.edu/cs2141/www/gdb.html`, with many more resources available online.

4 Implementation hints

You should develop your client program on the CCIS Linux machines, as these have the necessary compiler and library support. You are welcome to use your own Linux/OS X machines, but you are responsible for getting your code working, and your code *must* work when graded on the CCIS Linux machines. If you do not have a CCIS account, you should get one ASAP in order to complete the project. Your code must be `-Wall` clean on `gcc`. Do not ask the TA for help on (or post to the forum) code that is not `-Wall` clean unless getting rid of the warning is what the problem is in the first place.

You will want to make sure that you break your program up into modules, such that each module represents a sensible type abstraction. If you have questions on how to do this, please come see me or the TA.

5 Submitting your project

Once you are done, you should submit your (thoroughly documented) code. You can do so by running the `/course/cs3600f14/bin/turnin` script. Assuming you created a `c-project1` directory as described in Section 2.2, you can simply run

```
/course/cs3600f14/bin/turnin c-project1 ~/cs3600/c-project1
```

If you placed your code in a different location, you can turn it in by running

```
/course/cs3600f14/bin/turnin c-project1 <dir>
```

Where `<dir>` is the name of the directory with your submission. Regardless, when run, you should see output that looks like

```
bash$ /course/cs3600f14/bin/turnin c-project1 ~/cs3600/c-project1
  Added file Makefile (157 bytes)
  Added file cp1-problem1.c (1392 bytes)
  Added file cp1-problem3.c (2163 bytes)
  Added file cp1-problem2.c (2594 bytes)
  Added file cp1-problem4.c (2163 bytes)
Successfully submitted c-project1 for user amislove (confirmation ZiwKE5).
Submitted a total of 5 files (8469 bytes) in 0 directories.
```

The script will print out every file that you are submitting, make sure that it prints out all of the files you wish to submit. Finally, make sure you see the "Successfully submitted" link at the end. You should also receive an email confirmation of your submission. If you receive any other error messages, email the instructor with the error message if you are not able to diagnose the problem.

6 Advice

A few pointers that you may find useful while working on this project:

- Check the Piazza forum for question and clarifications. You should post project-specific questions there first, before emailing the course staff.
- Make *sure* you see the "Successfully submitted" message when submitting—if you do not see that and see an error message instead, do not ignore it!
- Finally, get started early and come to the TA/instructor lab hours. You are welcome to come to the lab and work, and ask the TA any questions you may have.