CS 580U, Fall 2022
Program 1

September 19, 2022

1 Instructions

1.1 Deadline

Due: Monday Sep 12 11:30:00 AM EDT 2022 (i.e. at the start of class), as measured by the Brightspace timestamp.

1.2 Before You Begin

All programs will be tested on the Classroom machines in the N01 classroom and/or on Remote. If your code does not run on the system in this lab, it is considered non-functioning *even if it runs on your personal computer*. You can write your code anywhere, but always check that your code runs on the lab machines before submitting.

1.3 Testing Your Code

The Makefile includes a target `make test` which will build an executable linked to the CUnit library (see subsection 3.1). You can then run this executable, which will give you an output like that of Listing 1.

Listing 1: Sample output from the test executable before any functions are implemented.

```
     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: Suite_1
  Test: test of hello ...Hello world!
passed
  Test: test of next_multiple ...FAILED
    1. test.c:33  - CU_ASSERT_EQUAL(next_multiple(is[idx], js[idx]),_next_multiple(is[idx], js[idx]))
    2. test.c:33  - CU_ASSERT_EQUAL(next_multiple(is[idx], js[idx]),_next_multiple(is[idx], js[idx]))
    3. test.c:33  - CU_ASSERT_EQUAL(next_multiple(is[idx], js[idx]),_next_multiple(is[idx], js[idx]))
  Test: test of ftoc ...FAILED
    1. test.c:42  - CU_ASSERT_EQUAL(ftoc(fahrenheits[idx]),_ftoc(fahrenheits[idx]))
    2. test.c:42  - CU_ASSERT_EQUAL(ftoc(fahrenheits[idx]),_ftoc(fahrenheits[idx]))
    3. test.c:42  - CU_ASSERT_EQUAL(ftoc(fahrenheits[idx]),_ftoc(fahrenheits[idx]))
    4. test.c:42  - CU_ASSERT_EQUAL(ftoc(fahrenheits[idx]),_ftoc(fahrenheits[idx]))
  Test: test of fibonacci ...FAILED
    1. test.c:50  - CU_ASSERT_EQUAL(fibonacci(ns[idx]),_fibonacci(ns[idx]))
    2. test.c:50  - CU_ASSERT_EQUAL(fibonacci(ns[idx]),_fibonacci(ns[idx]))
    3. test.c:50  - CU_ASSERT_EQUAL(fibonacci(ns[idx]),_fibonacci(ns[idx]))
```

```
    4. test.c:50  - CU_ASSERT_EQUAL(fibonacci(ns[idx]),_fibonacci(ns[idx]))

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      4      4      1      3        0
             asserts     14     14      3     11      n/a

Elapsed time =    0.671 seconds
```

1.4 Submitting Your Program

The Makefile includes a target `make submit` which will create a .tar file named "USERNAME.tar", where USERNAME is your LDAP username (e.g. mcole8). Make sure you have saved all files, run `make clean && make check`, and observe that a directory has been created (which you can later delete by running `make clean` again), as well as a .tar file with the appropriate name. Upload this tar file this assignment on Brightspace by the deadline.

1.5 Grading

The Makefile includes a target `make check` which will build the test executable and run it. Results will be both printed to the screen by the executable, and recorded to a file called **feedback.log** by the Makefile. You will receive points according to the following rubric:

Task	Points	Description
testHELLO	15	No partial credit
testNEXT_MULTIPLE	25	5 points per test case
testFTOC	25	5 points per test case
testFIBONACCI	25	5 points per test case
Coding Standards	10	At grader discretion

Table 1: Grading rubric for Program 1.

The coding standards document is located at `https://brightspace.binghamton.edu/d2l/le/content/146052/viewContent/518161/View`

2 Questions

Implement the following four functions in **functions.c** using prototypes located in **functions.h**:

1. `void hello(const char* msg);`
2. `int next_multiple(int start, int factor);`
3. `float ftoc (float fahrenheit);`
4. `int fibonacci(unsigned int term);`

Hello Accept an argument `msg`, and pass it to a call to `printf()` as the format string. You can assume that the string msg will not have any format specifiers within it.

Next Multiple Given a starting integer, and a factor integer, give the next largest integer which is a multiple of factor. If the starting integer is already a multiple of the factor integer, return the next largest multiple.

`start`	`factor`	`int next_multiple(int start, int factor)`
12	7	14
12	5	15
12	3	15

Fahrenheit to Celsius Given a temperature in Fahrenheit F, return the corresponding temperature in Celsius using the formula:

$$C = \frac{(F - 32)}{1.8}$$

`fahrenheit`	`float ftoc (float fahrenheit)`
32.0	0.0
212	100.0

Fibonacci Sequence The Fibonacci sequence is defined using the following piecewise function:

$$f_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f_{n-1} + f_{n-2} & \text{if } n \geq 2 \end{cases} \tag{1}$$

Using a for loop, given a term as input, give the sequence member corresponding to that term. For example:

`term`	`int fibonacci(unsigned int term)`
0	0
1	1
2	1
10	55
19	4181

3 Commentary

3.1 Testing

This programming assignment uses the CUnit testing framework, which means there's a lot of setup, testing, and teardown code that I've provided to you in advance. If you take a look at it, you might be intimidated. Don't worry – at this point you only need to recognize a bit of it.

Let's reconsider Listing 1. Note that each function which you will implement has an associated test function, and that when each of these functions are tested, there are one of two outcomes:

- Passed, as in the test of `hello`.
- Failed, as in the test of `next_multiple` and others. Note that if one test case fails, the test is marked as a failure, however if you pass some of the test cases you would earn partial credit proportional to the number of cases that passed.

Furthermore, at the very end, there is a run summary counting the number of successful tests and test cases (asserts).

Although you can see all of the boilerplate code needed to get this executable, we should focus on two specific blocks. First, consider Listing 2. For the test of hello(), all that must happen is the function should return. If it does, the test passes! For the test of next_multiple(), the process is slightly more complicated. Here, we set up a list of input values, then in a for loop, we call both your implementation of the function and the reference solution function. If they match, the assertion passes.

Listing 2: Extract from test.c

```
23 /* Test of hello */
24 void testHELLO(void){
25   hello("Hello␣world!\n");
26   CU_PASS("Successfully␣called␣hello()");
27 }
28
29 /* Test of next_multiple */
30 void testNEXT_MULTIPLE(void) {
31   int is[] = {0, 100, 365, 12258, 996};
32   int js[] = {1, 10, 7, 28, 4};
33   for (int idx = 0; idx <= 2; idx++) {
34     CU_ASSERT_EQUAL(next_multiple(is[idx], js[idx]),
35                     _next_multiple(is[idx], js[idx]))
36   }
37 }
```

Next, consider the changes to the Makefile, shown in Listing 3. Note that we had to both compile the source file **test.c** and link it to the library using the `LDLIBS` variable.

Listing 3: Extract from Makefile

```
 1 CFLAGS=-g -Wall -std=c99
 2 LDFLAGS=
 3 LDLIBS=-lcunit
 4 ARFLAGS= rcs
11 test: test.c functions.c solutions.a
12         $(CC) $(CFLAGS) $^ -o $@ $(LDFLAGS) $(LDLIBS)
```

If you have any other questions about CUnit, feel free to ask. However keep in mind that we will answer a lot of questions as we talk about the preprocessor, pointers, and structs during lectures later in the semester. Hopefully by the end of the course you will be able to fully understand test.c! In the meantime, feel free to take the source at face value.