EC 413 Computer Organization - Fall 2019
Lab 0: C Programming Language

Due: September 23rd, 2019

1 Introduction and Background

This lab will cover basic C programming, including compiling and running your code on the lab computers. Below are some directions for writing, compiling and running a C program.

1.1 Compiling C Code

1. **Compiling C Code**
   To set up your environment for the C programming language, you need the following two software tools available on your computer: (a) Text Editor and (b) The C Compiler. The files you create with your editor are typically called source files and they contain the program source code. The source files for C programs are named with the extension ".c". You can use any preferred text editor, namely Emacs, Vim, Gedit, etc.

   The source code needs to be "compiled", into machine language so that your CPU can actually execute the program as per the instructions given. The most frequently used compiler is the GNU C/C++ compiler (GCC). The version of GCC does not matter for this course. If you use any other compiler, **make sure your code compiles on the lab computers**. The labs will be graded on the machines in PHO 305.

   To compile a C code lab0.c, enter the following in the command line:

   ```
   gcc lab0.c -o lab0
   ```

   The command takes the input C code as an input and creates an executable binary lab0. If you do not specify an output binary file, it creates a default binary "a.out".

2. **Running Executables in Linux**
   Executable files (binaries or scripts) can be run as follows: Executable files (binaries or scripts) can be run from the command line in Linux. Entering the following command in the same directory as the binary will run the "lab0" binary generated by GCC:

   ```
   ./lab0
   ```

3. **Accepting User Inputs and Outputting Results**
   It is possible to accept user inputs when you run your binary. These inputs are called command line arguments. Command line arguments are allow users to input values such as file names so they do not need to be hard coded into the program.

   Use the following command to run your lab0 binary with an argument of 100:

   ```
   ./lab0 100
   ```

   Command line arguments are passed to your main function in C with the "argc" and "argv" arguments. There are plenty of examples online to show how to use these. If you have questions about them, ask the TA's in office hours.

2 Submission Requirements

Submit a zipped file named **bu_id**_lab0.zip with your code and compiled binaries in a directory named **bu_id**_lab0, where bu_id is replaced with your BU ID (including the 'U'). The code files must have the name **bu_id**_problem_**x**.c, where bu_id is replaced with your BU ID (including the 'U') and x is replaced with the problem number.

The binaries must have the name **bu_id**_problem_**x**.out where **bu_id** and **x** match the C code file. Your binaries must accept inputs from standard input, and print outputs on the standard output. This allows us to automatically grade your submissions. Submissions that do not follow these formats will not be graded.

In the provided zip file, you will find two C files (example1.c and example2.c), two binaries (example1.out and example2.out), two benchmark files (bench1.txt and bench2.txt), and a test.py python file. These files are to make testing easier for you as well as help you make sure your code will run with our automated grading.

If you run:

```
python test.py --problem_name example1.out --benchmark_path bench1.txt
```

you should get the following output:

```
example1.out: 1 -> 1 CORRECT!
example1.out: 2 -> 2 CORRECT!
example1.out: 8 -> 8 CORRECT!
example1.out: 7 -> 7 CORRECT!
```

As you can see, the example1.c code prints out the passed integer in the binary format, but reversed (left to right instead right to left).

For the following problems, you will create your solutions as specified above, compile the to specified binaries, and create benchmark files. Test.py script will run your code for each of the input $\rightarrow$ expected output pairs in the benchmark file, and notify you how many examples are correct. During grading, we will use the same script, but with a different benchmark file. You can use the provided C code to better understand how the inputs, outputs, and grading works. Example2.c illustrates how to pass arrays to your script, which will be needed for Problem 5.

Please note that if you run the executables that are not compiled on you computer, you will have to tell your Linux computer that it is safe to execute them. For example1.out, navigate your terminal to directory where example1.out is and execute the following command:

```
chmod +x example1.out
```

3 Problems

3.1 Problem 1

Write a C program that accepts an integer from standard input and prints it out as an unsigned binary number. Assume the input integer is 32 bits long. All 32 bits of the integer should be printed (including leading zeros).

Example input, output pairs:

- $1 \rightarrow$ 00000000000000000000000000000001
- $74 \rightarrow$ 00000000000000000000000001001010
- $1042 \rightarrow$ 00000000000000000000010000010010

3.2 Problem 2

Write a C program that accepts a string from standard input representing a signed 16 bit integer and returns the base10 value of the input.

Example input, output pairs:

- 00100010 11010111 → 8919
- 11111111 11110001 → -15
- 11111100 00000000 → -1024

3.3 Problem 3

Write a C program that accepts an 16 bit hex number from standard input and returns unsigned decimal number.

Example input, output pairs:

- ABAD → 43949
- 9AD7 → 39639
- 1234 → 4660

3.4 Problem 4

Write a C Program that accepts an unsigned integer *n* from standard input and outputs the *n*-th Fibonacci's number.

Example input, output pairs:

- 0 → 0
- 1 → 1
- 2 → 1
- 3 → 2
- 4 → 3
- 8 → 21
- 17 → 1597

3.5 Problem 5

Write a program that accepts an unsigned integer *n* through standard input. This argument determines the size of an array. Next, accept *n* more unsigned integers from standard input to fill the array. Sort the array ascending using the algorithm of choice. Afterwards, implement a filter function that accepts one argument and builds another array that contains only the elements greater than the given number. The argument of filter function should be passed as the last standard input of your program. Print the filtered array to the output.

Example input, output pairs:

- 5 1 5 8 7 2 5 → 7 8 // Accepts 5 inputs [1, 5, 8, 7, 2], sorts the array and returns the numbers greater than five SEPARATED WITH A SPACE.
- 6 4 5 7 1 3 4 3 → 4 4 5 7 // You should return both 4's