EC 413 Computer Organization - Fall 2017
Lab 1: C Programming Language

1 Introduction

In this lab, we are covering:

1. C programming language
2. **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[1], 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. If you use any other compiler, **make sure your code compiles on the lab computers**.

   To compile a C code lab1.c, enter the following in the command line
   » gcc lab1.c -o lab1
   The command takes the input C code as an input and creates an executable binary lab1. If you do not specify an output binary file, it creates a default binary "a.out".
3. **Running binaries in Linux**
   The executable can be run from command line as follows:
   »./lab1
4. **Accepting user inputs and output results**
   It is possible to pass some values from the command line to your C programs when they are executed. These values are called command line arguments and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

   To run the binary with an argument of 100, the following command has to be run
   »./lab1 100
5. **Binary, hex...**
   The problem set 1 introduced you to manually converting decimal to binary, binary to decimal. Now let us try writing C code to do this.

2 Grading

Submit a zipped file with your code and compiled binaries. The code files must have the name **bu_id**_problem_**x**.c, where bu_id and x are put in by the student and the binaries must have the name **bu_id**_problem_**x**.out. Your binaries must accept inputs from standard input, and print outputs on the standard output. This will allow us to automatically grade your submissions.

[1]It is widely known that Vim»Emacs.

In the provided zip file, you will find two C files (problem1.c and problem2.c), two binaries (problem1.out and problem2.out), two benchmark files (bench1.txt and bench2.txt), and a test.py python file. If you run:

```
python test.py --problem_name problem1.out --benchmark_path bench1.txt
```

you should get the following output:

```
problem1.out: 1 -> 10000000 CORRECT!
problem1.out: 2 -> 01000000 CORRECT!
problem1.out: 8 -> 00010000 CORRECT!
problem1.out: 7 -> 11100000 CORRECT!
```

As you can see, the problem1.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 → 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. Problem2.c illustrates how to pass arrays to your script, which will be needed for Problem 5.

3 Problems

3.1 Problem 1

Write a C program that accepts an integer from standard input, and prints out that integer as an unsigned binary number. Assume the input integer is 32 bits long. The output should have no leading zeros.

Test your code with the following examples:

- 72 → 1001001
- 274 → 100010010
- 1042 → 10000010010

3.2 Problem 2

Write a C program that accepts a string from standard input representing a signed 8bit integer, and returns the value of the input.

Test your code with the following examples:

- 01010100 → 84
- 11001100 → -52
- 11101110 → -18

3.3 Problem 3

Write a C program that accepts a decimal number from standard input, and returns the hex value of the input.

Test your code with the following examples:

- 74 → 4A
- 2047 → 7FF
- 123456 → 1E240

3.4 Problem 4 - Fibonacci Series

Write a C program that accepts an integer n from standard input, and prints out the n-th Fibonacci's number.

Test your code with the following examples:

- $4 \rightarrow 2$
- $8 \rightarrow 13$
- $17 \rightarrow 987$

3.5 Problem 5 - Binary Search Algorithm

Write a program that first accepts an integer n which determines a size of an array. Afterwards, accept n user inputs from standart input. Sort the list using an algorithm of choice. Afterwards, accept an integer x, and write a C function to search the array for the given element x in the array. Return the index of the first element x in the array. Implement this code using a recursive function. You can read about binary search algorithm here: https://en.wikipedia.org/wiki/Binary_search_algorithm.

Test your code with the following examples:

- 5 1 5 3 7 2 5 → 3 // accept 5 inputs [1, 5, 3, 7, 2], sort the array, return index of element 5
- 4 1 3 2 2 2 → 1 // remember to return the **first** element's index

4 Extra points

As TA's cannot award extra points, the following problems will get you "TA points". It is however strongly suggested that you complete these tasks, as we will repeat them in the next lab when using MIPS assembly. The value of TA points is to be determined by the professor.

4.1 Problem 6 - worth 10 TA points

Write a C program that accepts an decimal number from standard input as a string, and prints out the single precision floating point version of it. Separate the sign, exponent and mantissa with a single white space. Use the IEEE 754-1985 standard for encoding the number (https://en.wikipedia.org/wiki/IEEE_754-1985). This standard uses 1 bit for the sign, 8 bits for the exponent, 23 bits for the mantissa, and bias of 127.

- 3.1415926 → 0 10000000 10010010000111111011010
- 1.6180339887 → 0 01111111 10011110001101110111101
- 111.111111111 → 0 10000101 10111100011100011100100
- -0.0031415926 → 1 01110110 10011011110001100101101

4.2 Problem 7- Tower of Hanoi

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
- No disk may be placed on top of a smaller disk.

To start, accept the number of disks on the first rod, and print out the sequence of steps until all the disks are on the third rod and sorted. We will manually test this task, so text output is fine.

4.3 Problem 6

Choose whether you want 2 TA points, or 6 TA points. If more than 20% of the class chooses 6 TA points, everybody gets -1 TA point. Otherwise, everybody gets the number of TA points they asked for. Submit the requested number of points in a file "points.txt".