THEORY OF INFORMATION, ARCHITECTURE OF COMPUTERS AND OPERATING SYSTEMS (TIACOS)
Bioinformatics, ESCI
DAC, UPC

DataLab **Spring 2023**

Objectives

The purpose of this assignment is to become more familiar with bit-level representations of characters, integers and floating point numbers. You'll do this by solving a series of programming "puzzles." Many of these puzzles are quite artificial, but you'll find yourself thinking much more about bits in working your way through them.

Skills

- Manipulate character, integer and float codifications.
- Be able to program bit level manipulations in Python.
- Debug simple Python scripts.

Handout Instructions

> This wording has a companion file, `datalab-files.tar.gz`, with a Python script `datalab.py` that you must edit and deliver, and a directory with binaries tests[1] that will help you during the lab.

Start by copying `datalab-files.tar.gz` to a (protected) directory on a Linux machine in which you plan to do your work. Then give the command

```
$ tar -xvzf datalab-files.tar.gz
```

This will cause a number of files to be unpacked in the directory. The only file you will be modifying and turning in is `datalab.py`.

The `datalab.py` file contains a skeleton for each of the 8 programming puzzles. Your assignment is to complete each function skeleton using only *straightline* code for the integer puzzles (i.e., no loops or conditionals) and a limited number of Python arithmetic and logical operators. Specifically, you are *only* allowed to use the following operators:

[1]Choose version according with your operating system.

```
not  ~ & ^ | +  << >>
```

A few of the functions further restrict this list. See the comments in `datalab.py` for detailed rules and a discussion of the desired coding style.

The Puzzles

This section describes the puzzles that you will be solving in `datalab.py`.

Table 1 lists the puzzles in rought order of difficulty from easiest to hardest. The "Rating" field gives the difficulty rating (the number of points) for the puzzle, and the "Max ops" field gives the maximum number of operators you are allowed to use to implement each function. See the comments in `datalab.py` for more details on the desired behavior of the functions.

Name	Description	Rating	Max ops
`tmin()`	Smallest two's complement integer.	1	4
`negate(x)`	Return `-x` without using `-` operator.	1	5
`bitXor(x,y)`	`x ^ y` using only `&` and `~`.	2	14
`getByte(x,n)`	Extract byte `n` from word `x`.	3	6
`byteSwap(x,n,m)`	swaps the nth byte and the mth byte.	4	25
`floatIsEqual(f,g)`	Compute `f == g` for floating point arguments `f` and `g`.	3	25
`floatAbsVal(f)`	Return bit-level equivalent of absolute value of `f`.	2	10
`floatPower2(x)`	Return bit-level equivalent of 2.0^x for integer `x`.	4	30

Table 1: Datalab puzzles. For the floating point puzzles, value `f,g` are the floating-point numbers having the same bit representation as a 32 bits integer.

For the floating-point puzzles, you will implement some common single-precision floating-point operations. For these puzzles, you are allowed to use standard control structures (conditionals, loops), and you may use `integer` data types, including arbitrary integer constants. You may not use any unions, structs, or arrays. Most significantly, you may not use any floating point data types, operations, or constants. Instead, any floating-point operand will be passed to the function as having type `integer`, and any returned floating-point value will be of type `integer`. Your code should perform the bit manipulations that implement the specified floating point operations.

The included program `fshow` helps you understand the structure of floating point numbers. You can use `fshow` to see what an arbitrary pattern (integer numbers) represents as a floating-point number:

```
bash$ ./fshow-linux 2080374784

Floating point value 2.658455992e+36
Bit Representation 0x7c000000, sign = 0, exponent = f8, fraction = 000000
Normalized.  1.0000000000 X 2^(121)
```

You can also give `fshow` hexadecimal and floating point values, and it will decipher their bit structure.

Evaluation

Your score will be computed out of a maximum of 32 points based on the following distribution:

20 Correctness points.

10 Performance points.

2 Style points.

Correctness points. The puzzles you must solve have been given a difficulty rating between 1 and 4, such that their weighted sum totals to 20. You will get full credit for a puzzle if it works, and no credit otherwise.

Performance points. Our main concern at this point in the course is that you can get the right answer. However, we want to instill in you a sense of keeping things as short and simple as you can. Furthermore, some of the puzzles can be solved by brute force, but we want you to be more clever. Thus, for each function we've established a maximum number of operators that you are allowed to use for each function. This limit is very generous and is designed only to catch egregiously inefficient solutions. You will receive two points for each correct function that satisfies the operator limit.

Style points. Finally, we've reserved 2 points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Deliverable

Make a tar file with your `datalab.py` files and upload it via Moodle.

```
bash$ tar cvf datalab-handout.tar  datalab.py
```

Advice

You'll find it helpful to work through the functions one at a time, testing each one as you go.

References

[Randal E. Bryant, David R. O'Hallaron] *Computer systems: a programmer's perspective* Prentice Hall, 2015. Chapter 2.
https://upfinder.upf.edu/permalink/34CSUC_UPF/u34j1i/alma991002268649706710

[Python 3.9] *The Python Language Reference*
https://docs.python.org/3/reference/index.html