CSE 2421 – Systems 1

Spring Semester 2024

Programming Assignment #2

- **This assignment is worth 13pts.**
- **You must upload the zipped solution folder to Carmen, as solution.zip.**
- **The deadline for this assignment is February 12th 11:59pm ET.**
- **Deductions for late submissions apply.**
- **The main topic of this assignment is bit handling.**

Preliminary Instructions:

- Download the entire folder a2 from Carmen.
- You can debug and do the preliminary implementation of your code in your own laptop or desktop, but the final testing of your code must be done and work in stdlinux or coelinux.
- After downloading the folder a2 (or the file a2.tar.gz), copy the entire folder to your favorite OSU linux cluster (stdlinux or coelinux).

Instructions:

You should exclusively use the **`unsigned int`** data type in this assignment. You should only work in **bits.c**.

[2 pts] Implement function **`count_bits`**. The function should return the minimal number of bits necessary to represent a given input. The following table provides a few inputs and outputs of this function:

Input (Decimal)	Output
0	1
1	1
2	2
10	4
22	5

[2 pts] Implement a function **`count_bit_type`**. This function takes two 32-bit unsigned int arguments. The first one is an arbitrary number. The second argument can be zero or one, that is, a single digit bit pattern. The function should return the number of bits of the first argument that match the bit value of the second argument. The following table provides a few inputs and outputs of this function:

Input (Decimal)	Output
0 0	32
0 1	0
1 1	1
3 1	2
511 1	9

[3 pts] Implement a function **`find_pattern`**. This function takes two 32-bit unsigned int arguments. The first one is "the haystack", the number in which you will repeatedly search for a given bit pattern. The second argument is "the needle", the decimal number representing a bit-pattern that you will search for in the haystack. The function should return the number of times that the "needle" pattern is found in the "haystack" number. You can assume that patterns cannot overlap. The following table provides a few inputs and outputs of this function:

Input (Decimal)	Output
15 1	4
51 1	4
51 3	2
7347424 7	3
84055200 5	4
252165120 15	3
511 3	4

[2 pts] Implement a function **`invert_32bit`**. This function takes one 32-bit unsigned int, and returns a new 32-bit unsigned int where the lower 16 bits of the output correspond to the upper 16 bits of the input, and the upper 16 bits of the output correspond to the lower 16 bits of the input. The following table provides a few inputs and outputs of this function:

Input (Decimal)	Output
983040	15
47	3080192
228	14942208
255	16711680

[2 pts] Implement a function **`translate_bits`** which takes a filename (char*). In this function you only have to add the necessary file handing. The functions you developed previously are already being called. **`translate_bits`** should open a file with the given name. The function will then read several test cases from the input file. First, it reads an integer representing the number of test cases to process. Each test case will consist of four unsigned int, which you have to read from the file (a FILE *). We will call these four numbers arg1, arg2, arg3 and arg4 (but in the code you will find them as *val*, *needle*, *min_bits*, *min_matches*, respectively). For each test case, **`translate_bits`** will use **`invert_32bit`** on arg1 when two conditions are met: a) the number of bits of arg1 equal to 1 is greater or equal to arg3, and b) the number of times the binary pattern represented by arg2 found in arg1 is greater or equal to the value of arg4. Both conditions are necessary for arg1 to be translated. For the first condition you should use **`count_bit_type`**, while for the second condition you should use **`find_pattern`**. The following table provides a few inputs and outputs of this function:

Input	Output
2 51 3 3 1 119 7 4 2	 3342336 7798784
1 660 5 3 2	 43253760

1 660 5 2 3	660
1 5780 5 4 3	378798080
3 1638 3 5 3 1638 3 5 4 1638 3 7 2	107347968 1638 1638

[2 pts] Add descriptive comments to your code. Use descriptive variable names.

What is provided and what to do:

- A Makefile is provided.
- You should implement the functions in the order that they are requested. First count_bits, then count_bit_type, then find_pattern, then invert_32bit, and last, translate_bits.
- Several test harnesses are provided. These are the files test-*c. You can add more test cases, but do not change the calls to the functions in bits.c they are meant to test. You can add more 'test_one' calls, but it's up to you to determine if the output is correct or not.
- You should test the functions individually. The provided Makefile allows you to test each function independently. Read the Makefile to see what is being done. For example, to test count_bits do:
  make count-bits.test
  ./count-bits.test
- All other functions can be tested in a similar fashion.
- You can use "make clean" to remove object files and *.test files.
- **Write your code exclusively in the file bits.c.**
- Three text files with multiple test cases are provided for testing your implementation of translate-bits. You can write additional text files with more test cases.
- Do not leave additional printf calls or any extra printing message in bits.c. The only printing message should be the one that the instructor left uncommented.
- When you are ready to submit your work, run the script ./test-all.sh (may need to do "chmod +x test-all.sh"). Take a screenshot of the output. It should show your username, the linux hostname (stdlinux or coelinux) and the time stamp.
- Submitting your work: Create a folder called "solution" in your local machine. Copy the bits.c file with your work and the screenshot file to the folder. Compress the folder and upload it to Carmen.

Where to upload:
Upload via Assignments->Assignment 2 in Carmen.
Do not confuse with Files->Assignments.