ECE220: Computer Systems & Programming **Fall 2020 ZJUI**
Lab 5 **Tuesday 20 October**

Decoding a Secret Message with Puzzle Pieces

In today's lab, your group must use bit manipulations and iterations to fit several pieces into a puzzle. The positions of the pieces in ten puzzles have been used to encrypt a secret message, so once you have a working solution, you can solve the puzzles and decrypt the message. The goal of this exercise is to help you formulate loops in C and to use bitwise operators to solve problems. For MP4, you must also be able to formulate loops, but solving the problem itself requires more mathematical analysis rather than manipulation of bits (using bits is too slow for MP4; an algebraic approach is better).

Begin by checking out the `lab5` subdirectory in your Subversion repository. The directory contains a copy of this document and a starting point for your solution in the `lab5.c` file. You must write the `fit_pieces` function. The directory also contains a header file, `lab5.h`, which contains a description of how your function works, a separate source file, `main.c`, which contains the C function main used to parse command-line arguments and pass them to your function, and a test file, `do_tests`, that executes your code on the ten puzzles.

The Task

Given three puzzle "pieces" as 32-bit unsigned values, your code must determine whether the "pieces" can be shifted left without overflow (loss of 1 bits) in such a way that the three pieces fit together to fill the 32-bit word with 1s. Let's use 8-bit pieces as examples. Imagine that we have 00001101, 00001011, and 00001001. Can you shift the pieces left so that the 1s fill the 8 bits?

If we shift the first piece left by 4 bits, shift the second piece left by 0 bits (no shift), and shift the third piece left by 2 bits, we obtain 11010000, 00001011, and 00100100, respectively. As you can see, these three bits contain no common 1 bits, and when we "fit" them together (bitwise OR), they form the all 1s pattern.

Note that shifts cannot be negative. If instead of 00001011, the second piece in our example were 00010110, the puzzle could not have been solved.

Also note that overflow is not allowed. Given pieces 00001111, 00000111, and 00000011, the puzzle cannot be solved. Since there are a total of nine 1 bits, one of the 1 bits would have to be discarded using a left shift in order to avoid overlap in the pieces, but discarding 1 bits is not allowed.

Finally, you may assume that none of the three given pieces is 0. If you want a challenge, do not make this assumption, but it will hold in all of the pieces that we give you.

To solve this problem, you should make use of nested loops that consider each possible shift value for each of the 32-bit pieces. Managing the loops will take some thought, so make sure that you consider how the loop control works carefully before trying to write the program.

If your function finds a solution, it should print the three shift values in order and return 1. In the example above, you could print something like "shift values were 4, 0, and 2." Only the shift values matter—the exact output format is not important.

If your function cannot find a solution, it should return 0, and the main function will print a message indicating the failure for you.

To compile your code, use the command:

```
gcc -Wall -o puzzle main.c lab5.c
```

which produces an executable called "**`puzzle`**."

The Secret Message

If you think that your code is working, you can run the tests by executing

```
./do_tests
```

You may get a permission denied message after checking out. If so, type "**`chmod +x do_tests`**" to enable execution permission on the test script, then execute it again as above.

All of the puzzles in the test file are solvable, so if any of the calls fail, you have a bug.

If all succeed, you now have a sequence of 30 integers, which you can interpret as follows: numbers from 1 to 26 should be replaced with letters A to Z, respectively; adjacent pairs of 0s (these may cross puzzles, so be sure to think of all 30 numbers as one long sequence) should be replaced with spaces; 0s by themselves should be eliminated. If your code is correct, you have just decrypted the secret message!

Congratulations!

Tell your friends before they can finish coding! (or be nice…)