ECE/CS 438 Fall 2013
Programming Assignment 3
Due: By 7 p.m. on December 4, 2013

This assignment may be performed by a group consisting of at most two students.

The programming assignment consists of two independent parts.

Part I

The goal of Part I is to implement the CRC algorithm, and compare its performance with a table lookup-based scheme, as described below.

Write the code for an encoder procedure that generates CRC for a given generator and a given data. The usage for the encoder is as follows:

encoder g k d

where *g*, *k*, and *d* are all decimal numbers.

- *g* is the decimal equivalent of a 4-bit generator. Thus, if the generator is 1001 then *g* = 9. The most significant bit of the generator is always 1.
- *k* is the number of bits in the data input
- *d* specifies the decimal equivalent of the data input. Thus, if *k*=7 and *d*=35 then the data in binary is 0100011

Parameters *k* and *d* are optional. When *k* and *d* are specified, the encoder program should print the CRC computed using the specified generator and data.

Submit the code for your encoder along with compiling instructions, similar to previous programming assignments. You may use C, C++, Java or Python.

Performance evaluation part (i): Use your encoder procedure in a loop to compute CRC for a sufficiently large number of randomly generated data inputs, each consisting of 16 bits, and generator equal to 1001. Measure the time required per CRC computation on average. Compare this time with CRC computed using the table-based method below. During the performance evaluation, do **not** print the CRC to an output file (so that the delay in printing does not affect your measurements).

Table-based method: Pre-compute CRC for all possible 16 bit inputs and store the CRCs in a table. Thus, the table will have size 2^{16}. Use table lookup to determine the CRC for each randomly generated input (as above), and compute the average time required for this lookup.

Report the average cost of the above two algorithms.

Performance evaluation part (ii): Suppose that number of bits in the data is 8. For generator with 4 bits, the CRC will contain 3 bits, resulting in a 11 bit codeword. Suppose that each bit of this codeword is received in error independently with probability p (each bit may be erroneous independent of other bits). Empirically determine the probability that the receiver will detect presence of errors in a received codeword. To determine this probability, you may assume that the sender is sending the codeword 00000000 000 corresponding to data = 00000000. It turns out that the desired probability is independent of which codeword is transmitted. You should perform many experiments as follows: in each experiment, determine whether each bit of the codeword is in error (with probability p), and flip each erroneous bits in the codeword, if any. When a non-zero number of errors are injected into the codeword, run the decoder algorithm on the erroneous codeword, and determine whether the receiver detects presence of errors or not. You should repeat this experiments a sufficiently large number of times. Estimate of the desired probability as the fraction of codewords for which the decoder detects the presence of errors. Determine this probability for p = 0.1, 0.2, and 0.3.

Part II

Implement the internet checksum algorithm. You can find a description in Section 3.3.2 of the textbook, and also in a RFC linked from the Lectures page for the class. You may assume that the input to your algorithm consists of exactly 8 words, each containing 16 bits. Each such 16-bit word is specified using its decimal equivalent. For instance, if the 16-bit word in binary is 00000001 10000000, then the decimal equivalent is 384. As an example, the input (which should be read from the standard input) may be specified as:

```
2000  121  4  53  221  856  10005  9345
```

Your program should print the decimal equivalent of the 16-bit checksum.