

Project 2

CDA 4630/5636: Embedded Systems
Spring 2015 (**Instructor: Prabhat Mishra**)

Posted: March 3, 2015

Total: 10 points

Due: March 20, 2015, 11:30 PM

Assume that the dictionary can have eight entries (index 3 bits) and the eight entries are selected based on frequency (the most frequent instruction should have index 00). If two entries have same frequency, priority is given to the one that appears first in the original program order. The original code consists of 32-bit binaries. You are allowed to use only four possible formats (encoding) for compression (as outlined below). Note that if one entry (32-bit binary) can be compressed in more than one ways, choose the most beneficial one i.e., the one that provides the shortest compressed pattern (least number of total bits). If two formats produce exactly the same compression, choose the one that appears earlier in the following listing (e.g., *direct matching* appears earlier than *2-bit mismatches*). Please count the starting location of a mismatch from the leftmost (MSB) bit of the pattern – the position of the leftmost bit is 0.

Format of the *Direct Matching*

00	Dictionary Index (3 bits)
----	---------------------------

Format of the *2-bit consecutive mismatches* – starting location is counted from left/MSB

01	Starting Location (5 bits)	Dictionary Index (3 bits)
----	----------------------------	---------------------------

Format of the *2-bit mismatches* anywhere – Mismatch locations (ML) are counted from left/MSB

10	1 st ML from left (5 bits)	2 nd ML from left (5 bits)	Dictionary Index (3 bits)
----	---------------------------------------	---------------------------------------	---------------------------

Format of the *Original Binaries*

11	Original Binary (32 bits)
----	---------------------------

You are expected to implement the dictionary-based compression and decompression functions using C, C++ or Java. Submit a working prototype that will take any 32-bit binary (0/1 text) file and compress it to produce a output file that shows compressed patterns arranged in a sequential manner (32-bit in each line, last line padded with 1's, if needed), a separation marker "xxxx", followed by eight dictionary entries. Your program should also be able to accept a compressed file (in the above format) and decompress to generate the decompressed (original) patterns. Please see the sample files posted in the course assignments webpage.

Command Line and Input/Output Formats: The simulator should be executed with the following command line. Please use parameters "1" and "2" to indicate compression, and decompression, respectively.

./SIM 1 (or **java SIM 1**) for compression

./SIM 2 (or **java SIM 2**) for decompression

Please hardcode the input and output files as follows:

- Input file for your compression function: **original.txt**
- Output produced by your compression function: **cout.txt**
- Input file for your decompression function: **compressed.txt**
- Output produced by your decompression function: **dout.txt**

Submission Policy:

Please follow the submission policy outlined below. There will be up to 10% **score penalty** based on the nature of submission policy violations.

1. Please submit only one source file. **Please add “.txt” at the end of your filename.** Your file name must be SIM (e.g., SIM.c.txt **or** SIM.cpp.txt **or** SIM.java.txt). On top of the source file, please include the sentence: “/* On my honor, I have neither given nor received unauthorized aid on this assignment */”.
2. Please test your submission. These are the exact steps we will follow too. **In case your command line option is different, please put that information on top of the file as comment.**
 - Download your submission from eLearning (ensures your upload was successful).
 - Remove “.txt” extension (e.g., SIM.c.txt should be renamed to SIM.c)
 - Login to **thunder.cise.ufl.edu**. If you don't have a CISE account, go to <http://cise.ufl.edu/help/account.shtml> and apply for one CISE class account. Then you use **putty** and **winscp** or other tools to login and transfer files.
 - Please compile to produce an executable named **SIM**.
 - gcc SIM.c -o SIM **or** javac SIM.java **or** g++ SIM.cpp -o SIM
 - Please do not print anything on screen.
 - Assume hardcoded input/output files as outlined in the project description.
 - Compress the input file (original.txt) and check with the expected output (compressed.txt)
 - ./SIM 1 (or java SIM 1)
 - diff -w -B cout.txt compressed.txt
 - Decompress the input file (compressed.txt) and check with the expected output (original.txt)
 - ./SIM 2 (or java SIM 2)
 - diff -w -B dout.txt original.txt
3. *In previous years, there were many cases where output format was different, filename was different, command line arguments were different, or e-Learning submission was missing, All of these led to un-necessary frustration and waste of time for TA, instructor and students. Please use the exactly same commands as outlined above to avoid 10% score penalty.*
4. **You are not allowed to take or give any help in completing this project.** *In previous years, some students violated academic honesty (giving help or taking help in completing this project). We were able to establish cheating in several cases - those students received “0” in the project and their names were reported to UF Ethics office. This time we will have **double penalty** for cheating.*

Grading Policy

The class assignments webpage has the sample input and output files. Correct handling of the sample input will be used to determine 60% of credit awarded. The remaining 40% will be determined from other input test cases that you will not have access prior to grading. The other test cases can have different types and number of 32-bit binaries (0/1 text). It is recommended that you construct your own sample input files with which to further test your compression and decompression functions. **Please note that the new test case will NOT test any exceptional scenarios that are not described in this document.**