Project 2

CDA 4630/CDA 5636: Embedded Systems

Total: **10** points **Due**: April **2**, 2025 **11:30** pm

This is an **individual** assignment. You are not allowed to take or give any help in completing this project. Please **strictly follow the submission instructions** (outlined at the end of this document) and submit your source code in eLearning website before the deadline. Late submission (by email attachment to vtelukunta@ufl.edu) is allowed (up to 24 hours) with a 20% penalty (irrespective of whether it is late for 10 minutes or 10 hours). No grades for late submissions after 24 hours from the deadline. Please include the following sentence on top of your source code as a comment:

"I have neither given nor received any unauthorized aid on this assignment".

Bitmask-based Code Compression: In this project, you need to **implement both code compression and decompression using C, C++, Java or Python**. Please go through this document first, and then view the sample input/output files in the project assignment.

Assume that the dictionary can have sixteen entries (index 4 bits) and the sixteen entries are selected based on frequency (the most frequent instruction should have index 0000). If two entries have the 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 eight possible formats for compression (as outlined below). Note that if one entry (32-bit binary) can be compressed in more than one way, choose the most beneficial one i.e., the one that provides the shortest compressed pattern. If two formats produce the same compression, choose the one that appears earlier in the following listing (e.g., *run-length encoding* appears earlier than *direct matching*). If a 32-bit binary can be compressed using multiple dictionary entries by any specific compression format (e.g., *bitmask-based compression*), please use the dictionary entry with the smallest index value. Please count the starting location of a mismatch from the leftmost (MSB) bit of the pattern – the position of the leftmost bit is 00000.

Format of the *Original Binaries*

000	Original Binary (32 bits)

Format of the *Run Length Encoding* (RLE)

001	Run Length Encoding (3 bits)

Format of *bitmask-based compression* – starting location is counted from left/MSB

010	Starting Location (5 bits)	Bitmask (4 bits)	Dictionary Index (4 bits)

Please note that a bitmask location should be the first mismatch point from the left. In other words, the leftmost bit of the 4-bit bitmask pattern should be always '1'. The starting location (from the left) can vary between 00000 and 11101 (other choices are invalid since the bitmask will not fit).

Format of the *1-bit Mismatch* – mismatch location is counted from left/MSB

011	Mismatch Location (5 bits)	Dictionary Index (4 bits)

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

100	Starting Location (5 bits)	Dictionary Index (4 bits)

Format of the *4-bit consecutive mismatches* – starting location is counted from left/MSB

101	Starting Location (5 bits)	Dictionary Index (4 bits)

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

110	1st ML from left (5 bits)	2nd ML from left (5 bits)	Dictionary Index (4 bits)

Format of the *Direct Matching*

111	Dictionary Index (4 bits)

Run-Length Encoding (RLE) can be used when there is consecutive repetition of the same instruction. The first instruction of the repeated sequence will be compressed (or kept uncompressed if it is not part of the dictionary) as usual. The remaining ones will be compressed using RLE format shown above. The three bits in the RLE indicates the number of occurrences (000, 001, 010, 011, 100, 101, 110 and 111 imply 1, 2, 3, 4, 5, 6, 7 and 8 occurrences, respectively), excluding the first one. A single application of RLE can encode up to 8 instructions. In other words, up to 9 repetitions can be covered by RLE (first one non-RLE compression followed by up to 8 RLE compression). If you have more than 9 repetitions, you can apply RLE multiple times as long as it is profitable compared to other available options. While multiple combinations are possible, please cover 9 repetitions followed by the remaining ones. For example, if you have 21 repetitions, you should apply RLE three times as <1+8> + <1+8> + <1+2> (instead of <1+6> + <1+6> + <1+6>, or other possible compositions).

Your program should be able to handle 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 0's, if needed), a separation marker "xxxx", followed by sixteen 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 in the project assignment.

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

./SIM 1 (or **java SIM 1** or **python3 SIM.py 1**) for compression

./SIM 2 (or **java SIM 2** or **python3 SIM.py 2**) for decompression

Please **hardcode** the input and output files as follows:

1. Input file for your compression function: **original.txt**
2. Output produced by your compression function: **cout.txt**
3. Input file for your decompression function: **compressed.txt**
4. Output produced by your decompression function: **dout.txt**

Submission Policy:

Please follow the submission policy outlined below. There will be up to 20% **score penalty** based on the nature of submission policy violations. Because we will be using "diff –w -B" to check your output versus the expected outputs, please follow the output formatting. Mismatches will be treated as wrong output and will lead to score penalty.

1. 1. Please develop your project in one source file since the submission website accept 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 or SIM.py.txt). On top of the source file, please include the following sentence using the comment feature of that language.

/* 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.

- Download your submission from eLearning (ensures your upload was successful).
- Remove ".txt" extension (e.g., SIM.c.txt should be renamed to SIM.c). We know that eLearning adds a number at the end of the file if you submit multiple times. My script will take care of it (so do not worry about that number inserted by eLearning).
- Login to **storm**.cise.ufl.edu or thunder.cise.ufl.edu using your Gatorlink login and password. Ideally, if your program works on any Linux machine, it should work when we run them. However, if you get correct results on a Windows or MAC system, we may not get the same results when we run on storm or thunder. To avoid this time waste, we strongly recommend that you should test your program on thunder or storm servers.
- Please compile to produce an executable named **SIM**.
  - gcc SIM.c –o SIM **or** javac SIM.java **or** g++ SIM.cpp –o SIM **or** g++ -std=c++17 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 or python3 SIM.py 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 or python3 SIM.py 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 the TA, instructor, and students.* **Please use the exact same commands as outlined above to avoid 20% score penalty.**

4. **You are not allowed to take or give any help in completing this project**. *In the previous years, some students violated academic honesty. We were able to establish violation in several cases - those students received "0" in the project, and their names were reported to Dean of Students Office (DSO). This year, I will include one grade level penalty (in addition to getting 0 in project 2), e.g., if your final grade is B, it will be B-. If the department of university impose additional penalty (e.g., getting a failing grade in the class) that will be imposed. If your name is already in DSO for violation in another course, the penalty for second offence is determined by DSO. In the past, two students from my class were suspended for a semester due to repeat academic honesty violation (implies deportation for international students).*

Grading Policy

The project assignment has the sample input and output files. Correct handling of the sample input will be used to determine 60% of the score. 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. You can assume that we will use less than 1024 32-bit binary (0/1 text) patterns in the new test file. **Please note that the new test case will NOT test any exceptional scenarios that are not described in this document**.

Accessing CISE Servers (storm or thunder) to run Project 1

Please ignore this section if you know how to use CISE servers to run programs.

1. Register for a CISE account using your Gatorlink account and password.

   https://register.cise.ufl.edu/

2. Download PuTTY or similar shell to run your project using Linux. If you are using MAC, you can use the Terminal client.
3. Download WinSCP or similar file transfer mechanism. You can use sftp command from your MAC terminal client.
4. Once you register (step 1), you will get an email from do-not-reply@cise.ufl.edu indicating "your account has been created". Unless you get this email, the subsequent steps will not work.
5. Open PuTTY and login to any of the CISE servers (storm.cise.ufl.edu or thunder.cise.ufl.edu) using your Gatorlink account and password. Create a directory for your project 1 (e.g., mkdir project2).
6. Open WinSCP and login to any of the CISE servers (storm.cise.ufl.edu or thunder.cise.ufl.edu) using your Gatorlink account and password. Transfer files from your Windows directory to the directory you created (e.g., project2).
7. Go to PuTTY again and run the command from that directory (e.g., cd project2, g++ ...., diff ...., etc.). If your implementation is correct, 'diff' command (e.g., diff -w -B cout.txt compressed.txt) should not provide any output.