The project which we chose for SEng 440 was audio compression and decompression, which was later expanded in a specification session with Dr. Sima. This expansion revealed that we were to perform audio compression and decompression on a wave file using mu-law compression, implemented in software by the use of bit shifting operations. By the end of the project, there are to be two versions of main source code that will exist: one that serves as the original created solution and one that is its optimized enhancement. The optimization will utilize techniques taught in the SEng 440 lectures as well as refactoring in order to optimize the project performance.

Currently, we have finished the creation of the unoptimized original solution for the project and have tested it out with a wave file. It has been verified that the source code is indeed able to compress and then decompress the wave file into a new output file, which can then be played for comparison with the original file.

The source code is written in the C programming language to ensure that it will be able to work on the ARM virtual machine that has been specified for use with this project. Several C libraries are included in the project to make use of the which they provide.

The source code uses the argument line to take the file in as input while also creating a wave contents struct object in which to store the parsed components of the file. The fopen() operation is used to open the wave file as this allows it to be associated with a stream that is accessed by a pointer – thereby allowing all subsequent operations done with the file to be pass by reference. Since the file has been associated with a stream, the fread() operation is used to actually parse it and then store the components in wave contents object.

The parsing of the file is done piece by piece, using the amount of bytes associated with each component of a wave file to move through it. The file itself is split up into three components called chunks: the RIFF chunk, the FMT chunk and the Data chunk. Once these are parsed, the data samples – which exists as PCM data in the file is parsed.

The assert() operation is used throughout the file parsing to ensure that the data within the file is proper and not corrupted. While there are other checks on the file for validity, the assert() checks add redundancy for general purpose operation in the event that a file whose contents are unknown is to be tested in the program. In addition, it also makes sure that the file is being correctly parsed and that it is indeed a wave file.

The compression and the subsequent decompression of the file are done in functions separate from the parsing, with the former making use of the wave contents object.

The compression function uses a for() loop to iterate over the indices of the data parsed and stored in the wave contents object. It then uses a nested for() loop to use bitwise and bit shifting operations to compress the data sample. The compressed samples are written to a new file which upon the completion of the for loops will existed as a compressed wave file.

The decompression function takes the compressed wave file as the basis for creating the decompressed file. In a process similar to that of the compression function, this function uses a loop to go through the file values and put them in a new file. In this instance, it is a while() loop that does so, decompressing an 8-bit sample of the file into 16-bits.