# COMP9319 2023T2 Assignment 1: LZW Encoding and Decoding

Your task in this assignment is to implement an LZW encoder and its decoder with 15-bit 32768 dictionary entries (excluding those entries for the individual ASCII characters), called `lencode` and `ldecode`, in C or C++. After the dictionary is full, no new entries can be added. You may assume the source file may contain any 7-bit ASCII characters.

Both `lencode` and `ldecode` accept two commandline arguments: a path to the input file, followed by a path to the output file. The output file generated by `ldecode` should be identical to the corresponding source file of its encoded input, similarly, its encoded input file should be obtainable and generated by `lencode` using that corresponding source file as its input file. This is illustrated by the following example:

```
%grieg> lencode ~cs9319/a1/test1.txt test1.encoded
%grieg> ldecode test1.encoded test1.decoded
%grieg> diff ~cs9319/a1/test1.txt test1.decoded
%grieg>
```

In the encoded file, each ASCII character is stored in one byte with the most significant bit marked as 0, and each dictionary index address is represented as two bytes with the most significant bit marked as 1. When an index is read from the encoded file, the most significant byte will be read first (so we can determine if we are reading an ASCII or an index). The example used in the next section will elaborate on this further.

## Examples and Testing

Some example source files and their encoded files are available at `~cs9319/a1` of any CSE Linux machine. A sanity test script is also available there.

For example, consider a source file test1.txt:

```
cs9319@grieg:~/a1$ cat test1.txt
^WED^WE^WEE^WEB^WET
```

You can inspect its expected LZW encoded file test1.lzw using xxd:

```
cs9319@grieg:~/a1$ xxd -b test1.txt
00000000: 01011110 01010111 01000101 01000100 01011110 01010111  ^WED^W
00000006: 01000101 01011110 01010111 01000101 01000101 01011110  E^WEE^
0000000c: 01010111 01000101 01000010 01011110 01010111 01000101  WEB^WE
00000012: 01010100                                               T
cs9319@grieg:~/a1$ xxd -b test1.lzw
00000000: 01011110 01010111 01000101 01000100 01011110 01010111  ^WED^W
00000006: 01000101 10000000 00000100 01000101 01011110 01010111  E..E^W
0000000c: 01000101 01000010 10000000 00000100 01010100           EB..T
cs9319@grieg:~/a1$
```

I inspected both the source file and its encoded file using `xxd` to show you their differences. In particular, the binary value `10000000 00000100` that corresponds to the index value of 4 replaces `^WE` twice in the encoded file.

You can figure out the corresponding index values by tracing the LZW algorithm presented in the Slide 7-17 of Lecture Week 2. Following the same variable tracing as the Slides, i.e., (p, c, output, index, symbol), the content of test1.lzw can be derived similarly and illustrated in the following table:

```
NIL ^
^   W   ^   0  ^W
W   E   W   1  WE
E   D   E   2  ED
D   ^   D   3  D^
^   W
^W  E   ^W  4  ^WE
E   ^   E   5  E^
^   W
^W  E
^WE E   4   6  ^WEE
E   ^
E^  W   E^  7  E^W
W   E
WE  B   WE  8  WEB
B   ^   B   9  B^
^   W
^W  E
^WE T   4   10 ^WET
T   EOF T
```

When compared with the Slide 7-17 of Lecture Week 2, the above table has two differences. Firstly, the dictionary index starts from 0 instead of 256, as the ASCII character code and dictionary index no longer share the same address space. Secondly, note the output of row 7 is the actual characters instead of a dictionary index (i.e., 256 in the Lecture Slide or 0 in the above table). Since two bytes are used to represent an index, storing an index instead of two characters is not space saving. Therefore, an index to a dictionary entry will only be output when its entry is holding at least 3 characters. As a result, the encoded file is always smaller than or equal to the size of its source file. When their sizes are equal, the encoded file is in fact identical to its source file (for example, refer to test2.txt and test2.lzw in ~cs9319/a1).

To run the sanity test script on `grieg`, simply go inside the folder that contains your program source files and type: ~cs9319/a1/autotest that will run tests based on example files provided there. You should run the sanity test script before you submit your solution.

# Documentation and Code Readability

Your source code may be manually inspected. Marks may be deducted if your code is very poor on readability and ease of understanding.

# Marking

This assignment is worth 15 points, all based on auto marking.

Your solution will be compiled and tested on `grieg` (i.e., `grieg.cse.unsw.edu.au`), a particular CSE Linux machine. It may call and use any C or C++ functions or libraries available on `grieg`.

You should submit totally two C or C++ files (and must be named using the file extension .c and .cpp respectively), one for encoding and one for decoding. Your submitted files will be compiled using the following commands depending on their file extensions:

```
%wagner> gcc -o lencode lencode.c
%wagner> gcc -o ldecode ldecode.c
```

or:

```
%wagner> g++ -o lencode lencode.cpp
%wagner> g++ -o ldecode ldecode.cpp
```

Any solution that fails to compile on `grieg` with the above commands will receive **zero points** for the entire assignment. Your submission will not be tested with huge files, but it is expected to work properly and correctly for files up to a few hundred kilobytes.

# Performance

Any single test (e.g., `lencode test1.txt test1.enc`) that takes more than 5 seconds on `grieg` will be terminated, and your solution will receive **zero points** for that test. So, your dictionary lookup cannot be too slow (e.g., linear search may not make it).

# Submission

**Deadline: Monday 26th June 12:00pm AEST (noon)**. The penalty for late submission of assignments will be 5% (of the worth of the assignment) subtracted from the raw mark per day of being late. In other words, earned marks will be lost. **No assignments will be accepted later than 5 days after the deadline.** Please read the Assignments section in the course outline for details.

Use the give command below to submit the assignment or submit via WebCMS3:

```
    give cs9319 a1 lencode.c ldecode.c
```

or:

```
    give cs9319 a1 lencode.cpp ldecode.cpp
```

Please use classrun to make sure that you have submitted the correct file(s).

```
    9319 classrun -check a1
```

Note: Unfortunately the give and classrun commands are not available on grieg, but are on any other CSE linux machine, so you'll have to use these other machines to run the give/classrun command.

# Plagiarism

The work you submit must be your own work. Submission of work partially or completely derived from any other person or sources is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

Please read the Student Conduct section in the course outline for details.