# Programming Assignment: Compression v1

# Word Compression

This program focuses on maps, and file input/output.

### Introduction:

File Compression is an algorithm created for compressing text data to make a file occupy a smaller number of bytes. The MP3 file format is a compressed version of the original audio file, designed to reduce the file size for mobile devices and streaming. The MP3 algorithm is not loss-less, meaning that the audio quality is reduced as part of the process. Starting with 1 hour of high quality digital audio music, a lossless compression algorithm can reduce the file size by a factor of 2. Only redundancies in the file structure are removed, and all information can be re-built from the compressed file. An MP3, using medium level “bit-rates”, can reduce the original by a factor of 6. But as a ***lossy*** algorithm, in which some information is deleted, most listeners will be able to easily hear the difference.

Other types of compression are available for other types of media. Streaming video signals are often compressed, mostly for performance reasons. The field of Genetics also uses compression techniques to help manage the enormous amount of data created in mapping the Human Genome and performing genetic experiments. Note that compression is closely related to encryption, in which the intelligible details of a message are masked from unauthorized interceptors.

The idea of Word Compression is to strategically replace long words in a text document with shorter codes, in order to save space in your file. The ratio of bytes in your original file over the compressed is called the compression ratio. After successfully converting an input file this way, call it Word Compression Version 1.0, we will consider more strategic algorithms that can improve your compression ratio.

The steps involved in Word Compression Version 1.0, for converting a given text ***clean input*** file into a compressed ***output*** file are the following:

1. Examine the input file's contents and replace any word larger than 5 characters with a special String code that combines the String '\*' + an integer counter. The first large word will be replaced with \*0, the next with \*1, etc.
2. Each replacement must be stored in a map for future use. Note the map will contain no duplicates.
3. Re-examine the source file's contents, and write it to a new file. Replace each large word with the code version of that word from your map. Save your output file using the same name, inserting ‘.comp10’ between the filename and ‘.txt’.
4. Open your compressed file, decode the contents using your map, and print the original message to the console.

### Encoding a File:

For example, suppose we have a file named example.clean.txt with the following contents:

File Compression is an algorithm created for compressing text data

The encoding map is:

|  |  |
| --- | --- |
| LongWord | Code |
| 'Compression' | \*0 |
| 'algorithm' | \*1 |
| 'created' | \*2 |
| 'compressing' | \*3 |

File \*0 is an \*1 \*2 for \*3 text data

In the original file, the **clean** file has 57 characters. The compressed file named example.comp10.clean.txt contains only 27 characters, for a compression ratio of 2.11 to 1

mobydick.clean.txt starts with 947,301 characters, not including spaces. The compressed version named mobydick.comp10.clean.txt has 819,585 characters, for a compression ratio of 1.16 to 1