C++ Programming Methods

Assignment 2, Simple Compression

Bas Terwijn <b.terwijn@uva.nl>

In this assignment we will create a program to compress and decompress text files. Here we practice using file I/O and streams.

Passing arguments

To allow users to easily use our new MyCompress compression program we like them to be able to pass information to the tool on the command line which is common for unix-like programs (reading command line arguments is described on page 404 of *Absolute C++, fifth edition, Walter Savitch*). Our program should behave according to this man (manual) page:

```
Name
      MyCompress

Synopsis

      MyCompress <flag> [<input-filename>] [<output-filename>]

Description

      Compresses and decompresses text files.
      If no output filename is given it writes to standard output (stdout).
      If no input filename is given it reads from standard input (stdin).

Options for <flag>

      -h    give this usage information and exit
      -c    compress the input file and write result to output file
      -d    decompress the input file and write result to output file

Example Usage

      MyCompress -c input.txt output.txt  # compress input.txt to output.txt
      cat output.txt | MyCompress -d  # read from stdin, decompress to stdout
```

In order for your program to use the same code to both read and write to file and read and write to standard input and output you should use the *istream* and *ostream* classes as described on page 575 of *Absolute C++, fifth edition, Walter Savitch*. Class *istream* is the parent class of both *ifstream* and *cin* and as a result can be used to read input from both these subclasses and therefore can help you avoid writing code to read from each separately. Similarly *ostream* is the parent class of *ofstream* and *cout* and can be used to write to both these subclasses.

Compressing

The compression itself is done by run-length encoding. This encoding replaces any sequence of *k* characters, with *k* larger than 1, with that character directly followed by the number *k*. Also all

special characters, for example whitespace, tab and new-line characters, are treated this way. We will assume sequences are not longer than can be represented by an *int*. The decompression should reverse the compression so that the original text is recreated.

Problems occur when the original text has digits, as then the decompression does not know if a digit was added by the compression as part of a number *k* or that it was already there in the original text. To solve this problem the compression will add to the output a '\' (backslash) character before each digit it finds in the input text. It then also needs to add a '\' character to the output before each '\' character in the input. This gives the decompression enough information to uniquely recreate the original text.

To give an example of compressing and decompressing:

```
original:     “aaaaa bbbcde     111112345, single \ multiple \\\\”
compressed:   “a5 b3cde 5\15\2\3\4\5, single \\ multiple \\4”
decompressed: “aaaaa bbbcde     111112345, single \ multiple \\\\”
```

The program should read the input file one character at the time (using for example the istream::get() function). It should also write to the output file one character at a time possibly preceded by a '\' and followed by a sequence-length number. There is no need to have more than 2 distinct characters from the input file in memory at any time, this will only complicate things.

After processing a file both the compression and decompression should print the compression ratio calculated by *characters-in-output* divided by *characters-in-input* to inform the user of the compression result. This and optionally other information should be printed to *standard error* so not to corrupt an output file written to standard output.

Testing

To test your implementation you can see if decompressing a compressed file results in a file that is identical to the original. Using the shell command *diff <file1> <file2>* on a unix-like system, or comparing the hash values of the files will tell you if they are identical.

Besides this compare your results with these test files.

```
original.txt
compressed.txt
decompressed.txt
```

Submission

Submit your solution before the deadline to blackboard. Add to each solution:

- your name, student number and the name of the assignment

- references to the source of any algorithm or code that you did not create yourself

- operating system and compiler that was used to test the code