Project #3 – run-length encoding

Version 2.0.1: ©2021-03-03

Course	INFO-1156 Object-Oriented Programming in C++
Professor	Garth Santor, Janice Manning, and Lynn Koudsi
Assigned	March 8th 2021
Due	March 29th 2021 @ 11:59pm
Weight	7%

Project Description

Create two C++ 17[1] console[2] applications; one that encodes a file using RLE compression, and another that decodes the compressed file.

Program Interfaces

The encoder should have the following command-line interface:

```
rle [--debug] [--help] inputfile [outputfile]
```

Where:

- `--debug` indicates the output file is in text format.
- `--help` shows the command line options.
- `inputfile` is the name of the text or binary are file to be encoded (could have any name/extension)
- `outputfile` is the name of the file containing encoded result. If a name is not provided, it will be the original filename with ".rle".

The decoder should have the following command-line interface:

```
rld [--debug] [--help] inputfile [outputfile]
```

Where:

- `--debug` indicates the output file is in text format.
- `--help` shows the command line options.
- `inputfile` is the name of the RLE encoded file to be decoded. If there is no ".rle" extension then an output filename must be provided.
- `outputfile` is the name of the file containing the decoded result. If a name is not provided, it will be the original filename with ".rle" stripped from the name.

Note that you cannot have both binary and text switches simultaneously.

- `--help` indicates that a help message should be printed.

Technology

Run-Length Encoding compresses long runs of characters by writing the file information as a byte count followed by the character byte. The counts are stored as unsigned char (uint8_t) and the character values are stored as unsigned chars.

For example:

[1] Must be compiled with /std:c++17
[2] Windows platform

hello.txt	
hello	x68x65x6Cx6Cx6F

Is encoded with the command:
`rle hello.txt`
producing the file

hello.txt.rle
x01x68x01x65x02x6Cx01x6F

The file:

repeating.bin	
aaaaaabbbb	x61x61x61x61x61x61x62x62x62x62

Is encoded with the command:
`rle repeating.bin`
producing the file

repeating.bin.rle
x06x61x04x62

Since the repeat count is only a single byte, the largest count that can be stored is UCHAR_MAX which is 255. Therefore, repetition counts that exceed 255 must be done in groups of 255.

For example:

512a's.txt	
aaaaaaaaaaaaa…a	x61x61x61x61…x61

Is encoded with the command:
`rle 512a's.txt`
producing the file

512a's.txt.rle
xFFx61xFFx61x02x61

rltest

A test program has been provided that will run your programs against a series of test reporting which tests pass, and which tests fail. The nature of the failure will also be reported. Place the program into your debug and/or release directories. Help is provided with the '--help' switch.

Grading Criteria	Max		Actual	
Requirements	**Weight**	**Points**	**Awarded**	**Grade**
Test Cases				
#0: rle trivial.txt trivial.txt.rle	2%	1	1	2%
#1: rle simple.txt simple.txt.rle	10%	1	1	10%
#2: rle typical.txt typical.txt.rle	10%	1	1	10%
#3: rle oneK.txt oneK.txt.rle	3%	1	1	3%
#4: rle oneKone.txt oneKone.txt.rle	2%	1	1	2%
#5: rld trivialcoded.txt.rle trivialcoded.txt	2%	1	1	2%
#6: rld simplecoded.txt.rle simplecoded.txt	5%	1	1	5%
#7: rld typicalcoded.txt.rle typicalcoded.txt	4%	1	1	4%
#8: rld oneK.txt.rle oneK.txt	3%	1	1	3%
#9: rld oneKone.txt.rle oneKone.txt	2%	1	1	2%
#10: rle default.txt	5%	1	1	5%
#11: rld defaultcoded.txt.rle	5%	1	1	5%
#12: rle binary.bin binary.bin.rle	10%	1	1	10%
#13: rle binary.bin.rle	10%	1	1	10%
#14: rle --debug typical.txt typical.txt.rle	4%	1	1	4%
#15: rld --debug typicalcoded.txt.rle trivialcoded.txt	4%	1	1	4%
#16: rle --help	2%	1	1	2%
#17: rld --help	2%	1	1	2%
#18: rle (no args)	2%	1	1	2%
#19: rld (no args)	2%	1	1	2%
#20: rle nosuchfile	2%	1	1	2%
#21: rld nosuchfile	2%	1	1	2%
#22: rle existingfile.txt .	2%	1	1	2%
#23: rld existingfile.txt.rle .	2%	1	1	2%
Non-functional requirements				
Multi-file solution	3%	1	1	3%
Penalties				
Penalties from *C & C++ Grading Guide* ***v2.2.0***	**-5%**	1	0	0%
Late submission:	**-10%**	1	0	0%
Total				**100%**

Difficulties
Moderate
Harder
Hardest

Submission Requirements

1. Submit **entire Visual Studio project directory** to Fanshawe Online
    a. Delete ***all*** debug and release directories.[i]
    b. Submit in a .ZIP, .7z archive file.

---

[i] Alternatively, you can 'clean' your project for submission by downloading 'vsclean' a Visual Studio Solution Cleaner from www.gats.ca .