Computer Science 310

Lab #3

Due Date : Thursday, February 18th, 4 PM

30 Points

Create a Lab3 subdirectory in your class account, and store all code associated with this assignment in this directory. Name your source files cmp.cpp and compress.cpp and include executables named myCmp and myCompress.

The cmp Utility

We want to get some practice with some of the *Linux* systems utilities for *cosequential processing*. Cosequential processing is the coordinated processing of two or more sequential lists to produce a single output list.

`cmp` compares two files. It takes the filenames of these two files at the command line, and displays the byte and line number where they first differ to the screen. If both files are identical, it does nothing. If all of one file is identical to the first part of another, it reports that end-of-file was reached on the shorter file before any differences were found.

Here is a sample output for two files that differ.

```
file1 file2 differ: byte 92, line 5
```

And a sample output for `file1` with an empty file `file4`.

```
cmp: EOF on file4
```

You should support either/both of the options below prior to the file names.

- `-c` Output differing bytes as characters. This option displays the character that is different between both files. Both differing characters are shown first in octal and then as an actual character. For example,

  ```
  file1 file2 differ: char 92, line 5 is 117 O 101 A
  ```

- `-i N` Ignore differences in the first N bytes of input.

You should provide errors for the following.

- Requesting the opening of a file that doesn't exist.
- An illegal number of parameters on the command line.
- An illegal option not listed above.
- Providing the name of a directory rather than a file.

Compression Using Run-Length Encoding

Write two value-returning functions `compress` and `uncompress` that can be used to encode/decode an input string using run-length encoding. Assume that `ff` (hex) is the run-length indicator.

For example,

```
string myInput = "93 93 93 93 92 91 91 94 94 94 94 94 95 95 95 73 73 73 73 73 73 73";
cout << compress(myInput) << endl;
```

would output `ff 93 04 92 91 91 ff 94 05 95 95 95 ff 73 07`

Your `uncompress` function could take this output as input, and return the original input string. You may assume the input string for encoding or decoding is always in a valid format.

Written Exercises

Answer each of the following on a separate sheet of paper.

1. Create a Huffman code for the 421-character file described in this table.

Character	Frequency
a	27
b	18
c	28
d	45
e	90
f	97
g	93
h	23

   How many bits would we use to encode this file, using the Huffman code?

2. (a) Suppose you want to store the ranks of playing cards (`Two`, `Three`, `Four`, `Five`, `Six`, `Seven`, `Eight`, `Nine`, `Ten`, `Jack`, `Queen`, `King`, `Ace`). How many bytes would it take to represent the rank as a fixed-length sequence of characters, using the names shown above?

   (b) How many bits would it take to represent the rank in as few bits as possible? If we had to use an integral number of bytes, how many would we use?

Hand In

Hand in your written exercises at the start of class or upload to Canvas. Copy over your two coding exercises to your scratch area.

```
cp *.cpp /scratch/csc310/last_fm/Labs
```