University of Toronto
CSC228S, Winter 2001

Assignment 1

Deadline A :	*noon, Thursday January 25, 2001.*
Deadline B :	*noon, Thursday February 08, 2001.*
Total Weight:	*10% of your course grade. Part A is worth a small fraction of this total weight.*

Overview

This assignment will teach you about file input and output in C++ and about working with binary files. Your job is to write a program which takes a text file and produces a new, compressed, version of that file. Your program will also be able to uncompress and reproduce the original file.

Compression technique

There are many ways to compress a file. The technique you will use is to convert one character of the input file at a time from its ASCII representation into a new binary code. One possible encoding is shown in the table below:

character	code
A	0
B	101
C	100
D	1101
E	1100
eof	111

(This encoding is only appropriate for files that contain nothing but the upper case letters A through E.) Notice that while the character 'A' in an uncompressed file takes `sizeofchar()` bytes of space in the file, with the encoding above, it takes only a single bit of space.

Representing an encoding

Your program does not have to create the encoding, but will read it from an input file. This file will be a text file with the following format:

> Each character which may occur in the uncompressed file is followed by a sequence of characters 0 and 1 that represent its code. This is followed by a pipe character `|` as a delimiter. At the beginning of the file, separated from the first character by the delimiter, is the code for eof.

For example, we would represent the encoding shown above with the following file called `encoding1.txt` which is also provided on the website.

```
111|A0|B101|C100|D1101|E1100
```

Note that this is a text file; each of these symbols is stored as a character. That means each "1", for example, is stored as the number that is its ASCII value, and takes up `sizeofchar()` bytes in the file.

Your program must store the encoding table at the beginning of the compressed file so that the decompression can be done without separate access to the encoding used. You should use this same format.

An instantaneous encoding

Notice that in our example encoding, no code is the beginning substring of any other code. For example, `A` is a single `0`, and no other code begins with `0`. This means that we can decode without backing up in the file. For example, an input file that contained "ABACADABA would be represented in the compressed file as this sequence of bits:

```
010101000110101010
```

Try decompressing this by hand to see that it can be done without backing up. An encoding with this property is called "instantaneous". You may assume that your program will only be given instantaneous encodings to work with.

Doing the uncompression

You will probably choose to have a class that handles the conversion back and forth between characters and their code. When uncompressing, how do you know how many upcoming bits represent the next character, and should hence be passed to the appropriate conversion function? Remember that the codes are of varying lengths.

To solve this problem, write a function which receives a string and returns a boolean indicating whether or not that string is a valid code. If the code is valid, your function can also set one of the arguments to be the decoded character. If the code isn't valid, your program can read another bit from the file and try again. Your conversion function can do its task by by searching through a table of all the codes.

This isn't a very time-efficient way to do the decompression but it will be fine for this assignment.

Command line arguments

Your program will not be interactive. All input will be given on the command line and output will be to a file. Do not add any prompts or extra messages. (The `-v` option below is a small exception to this rule.)

Your program must take up to 4 command-line arguments. The first argument, `-v`, is optional. It indicates that your program should run in *verbose* mode. In verbose mode, your program will print to standard error a message that indicates the compression ratio achieved (i.e., the original file size divided by the new file size.) If present, the `-v` must be first.

The second argument, either `-c` or `-u`, is required. It indicates whether your program is to *compress* or *uncompress* a file.

In *compress* mode, an argument must follow the `-c`, and it must be the name of an existing file which contains the encoding to use for the compression.

The final argument is required, and is a string representing a filename. In *compress* mode, it represents the name of the file to be encoded. The resulting output should be written to a new file with the same name with an added `.BIN` extension. Encoding the input file `a.out`, for example, would create an output file `a.out.BIN`. This filename must end in "`.BIN`". In *uncompress* mode, this string represents the filename of the file to be decompressed. The resulting output should be written to a new file with the same name but without the `.BIN` extension. In *uncompress* mode there is no need to name a file containing the encoding since it has been included in the compressed file.

Erroneous input

You should handle the following situations:

- Incorrect use of command-line arguments. (For example, asking to compress but not providing the encoding file.)
- Compressing or uncompressing a file that doesn't exist.
- Compressing or uncompressing when there is already a file with the same name as the necessary output file.
- Compressing with an encoding file that doesn't exist.

In all cases, just print an appropriate message and stop the program. Do not prompt the user for different input.

You should not try to handle input files or encoding files that do exist but have inappropriate contents. In particular, you may assume that:

- When encoding, every character in the input file has an encoding listed in the encoding file.
- The encoding file describes an instantaneous encoding.
- When decoding, the `.BIN` file begins with a header describing the encoding used, in the correct format.

You may also assume that the codes given in the encoding file are less than 32 bits long.

How to write individual bits to a file

Although you can think of a file as a continuous string of bits, in C++ you are limited to writing and reading complete bytes at a time. Using `fwrite` you can write in binary the contents of a memory location but you must write an integer number of bytes. Some of your codes will be less than one byte long (perhaps even only a single bit) and others will be longer than a single byte. To confuse matters even further, a given binary code may straddle two bytes. The figure below shows the two bytes that would result if we compressed a file containing ABCD, using the encoding shown on page 1:

0	1	0	1	1	0	0	1		1	0	1	1	1	1	0	0

Notice that the code for D starts in byte 0 and finishes in byte 1. Also notice that the eof symbol `111` follows the symbol for D and that the rest of the final byte is padded with zeros. The eof symbol tells us that those trailing zeros don't represent 'A's but are just padding.

This *packing* of bits into bytes before writing to the file is perhaps the hardest part of this assignment. Make sure you understand how to work with bitwise operators by looking at the sample code provided on the web-site.

With `fwrite` you can write any integer number of bytes at once. so we aren't limited to packing and writing a single byte. You should use a buffer that is 10 times `sizeofchar()` bytes long.

Note that although your program will work on multiple platforms, files you compress on one machine using this approach may not decompress properly on another machine which has a different value for `sizeofchar()`.

Using od when debugging your program

Working with binary files can be frustrating because you can't just print the files to the screen or create them with your favorite editor. You should use the unix tool **od** to view human-readable versions of your compressed files. And you must use `od` to produce printouts of your testing for your assignment report. We will provide some tips on using `od` on the website.

Relevant Readings

We will cover some of the course material relevant to this assignment in class, but you are responsible for reading many of the details yourself in the text. Here are the sections of the text which you should begin reading now.

6.1	Data compression including run-length encoding (just for general background)
4.1.6	Binary files and using od
2.8.3	I/O redirection in unix
2.2	Opening files
2.4	Reading and writing files

What to hand in for the first deadline ("A")

There is no specific programming requirement for Part A; It is merely a step on the way to part B. You should have made a good start on the design, and have some code working. You may need to create simple driver programs to test your code at each phase of development. Hand in a printout of your code (including any driver programs), and some test runs that demonstrate what works. Although your program will be designed to run on very large files you can test it using relatively small files. Explain your test results with brief, hand-written comments on your printout.

Make your deadline A submission **as brief as possible,** and do not hand in a report.

What to hand in for the second deadline ("B")

For deadline B, you are to hand in your assignment in the form of a report, as described in the 228 Course Guide. The style of a 228 report is different from what you are used to from past csc courses, so be sure to re-read the relevant sections of the Guide before getting far into the assignment. Your report must include all of your code and all relevant input and output test files. You can use a script file to demonstrate the execution of your programs (see `man script`).

Take care to meet the specifications as given in this handout and in any clarifications on the web-site. Notice also that many decisions are not specified. In these cases you must make decisions yourself. Remember to justify these decisions including explaining the alternatives you did not choose. This is a very important part of your report.

Your report should be thorough, well-written, concise, and honest. It should not be more than about five pages long, not including code and tests. Make sure that you run your report through a spelling checker.

How to package up your assignment submissions

See the course web site for important information about how we expect you to package up your assignment for submission and how to do the electronic submission.

A note about C++

We do not expect you to use any advanced features of C++ for this assignment. In particular, no inheritance or templates should be necessary for a good design.