

CS2604 (Fall 2001)
PROGRAMMING ASSIGNMENT #2: Huffman Encoder/Decoder
(Revised 12:30pm, September 27)

Due Wednesday, October 10 @ 11:00 PM for 100 points

Early bonus date: Tuesday, October 9 @ 11:00 PM for 10 point bonus

Late date: Thursday, October 11 at 11:00 PM with a 30 point penalty

Assignment:

You will write a pair of file compression programs based on Huffman coding trees. Typical encoding schemes like ASCII use fixed-length coding schemes with each character represented by a fixed number of bits. Instead, Huffman coding trees define variable length codes, where more frequently occurring characters are encoded with fewer bits than characters that occur less frequently. As a result, files encoded using Huffman coding trees often require significantly less space than those that use fixed-length coding schemes.

Invocation and I/O:

The programs are invoked as `huffencode infile outfile` and `huffdecode infile outfile`

`huffencode infile outfile` encodes the contents of *infile* using a Huffman coding tree and stores the tree and the encoded file in *outfile*. A Huffman coding tree should be created based on the contents of *infile*. The file must first be opened, its contents read, and a frequency count generated. The frequency count must then be used to generate a Huffman tree in memory. The basic encoding unit is a single byte. The input file might be, but need not be, an ASCII file. Thus, it will be necessary to represent all characters, including spaces, tabs, and non-printing characters. For verification purposes, your program should print to standard output the code and frequency for each character in the Huffman coding tree in ascending order by code value.

Upon completion, the output file *outfile* should contain a representation of the code list as well as the encoded input file. The encoded output should be stored in binary format to make use of the space gains from the encoding.

`huffdecode infile outfile` decodes the contents of *infile* and stores the decoded file in *outfile*. The format of *infile* will be the same as the format of *outfile* for `huffencode`: it will consist of the Huffman code list and the encoded version of the message. This program will first reconstruct in main memory the Huffman tree based on the code list in the file, then will decode the encoded information using the tree.

Upon completion, the output file should contain the decoded information. Of course, running `huffdecode` on the output file from `huffencode` should result in the original file.

Design and Implementation Considerations:

Before implementing Huffman coding trees, it would be wise to read Section 5.6 of the textbook. You may use the code from the textbook, though there is no requirement to do so.

Your Huffman coding trees should support binary data input, with the basic unit a single byte of data. In writing to disk, your program must store the encoded data in binary format as well. That is, don't store each 0 or 1 as a separate character.

To ensure that your program can decode files encoded by other programs (including the test files), the following standard ordering will be used for the output files:

- the number of codes (in decimal)

- the code values and their lengths
- the codes themselves (in binary)
- the number of coded characters (in decimal)
- the encoded message (in binary)

Note that the codes and the encoded message may overlap byte boundaries. Only the final byte should be padded with extra zeros.

You should use a four-byte integer to store the number of coded characters. For any condition where this is inadequate, an error message should be presented. The number of coded characters will be byte aligned, not word aligned.

See the sample input files and explanations for more details.

Programming Standards:

You must conform to good programming/documentation standards, as described in the Elements of Programming Style. Some specifics:

- You must include a header comment, preceding `main()`, specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Always use named constants or enumerated types instead of literal constants in the code.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation and blank lines to make control structures more readable.
- Precede each function and/or class method with a header comment describing what the function does, the logical significance of each parameter (if any), and pre- and post-conditions.
- Decompose your design logically, identifying which components should be objects and what operations should be encapsulated for each.

Neither the GTAs nor the instructors will help any student debug an implementation, unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code taken from the textbook. Note that the textbook code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point. You may not use code from STL, MFC, or a similar library in your program.

Testing:

Sample data files will be posted to the website to help you test your program. These are not the data files that will be used in grading your program. While the test data provided should be useful, you should also do testing on your own test data to ensure that your program works correctly.

Deliverables:

You will submit this project electronically. In particular, you will create a zip'ed archive file containing the following items (and nothing else):

- all source code files necessary to build an executable
- either the project workspace files (.dsw and .dsp) for Visual C++ users, or a makefile for g++ users.

Windows users should be sure to use a modern zip tool which preserves long file names. A suitable freeware command-line zip tool will be posted on the course website. UNIX users should submit a gzip'ed and tar'ed file.

Once you have assembled the archive file for submission, for your own protection, please move it to a location other than your development directory, unzip the contents, build an executable, and test that executable on at least one input file. Failure to do this may result in delayed evaluation of your program, and a loss of points.

You will submit your project to the automated Curator server. The instructions and necessary software are available at: <http://ei.cs.vt.edu/~eags/CuratorGuides.html>. If you make multiple submissions, only your last submission will be evaluated.

Pledge:

Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement in the header comment preceding the function main() in your program. The text of the pledge will also be posted online.

```
// On my honor:  
//  
// - I have not used C++ language code obtained from another student,  
//   or any other unauthorized source, either modified or unmodified.  
//  
// - All C++ language code and documentation used in my program  
//   is either my original work, or was derived, by me, from the source  
//   code published in the textbook for this course.  
//  
// - I have not discussed coding details about this project with anyone  
//   other than my instructor, ACM/UPE tutors or the GTAs assigned to this  
//   course. I understand that I may discuss the concepts of this program  
//   with other students, and that another student may help me debug my  
//   program so long as neither of us writes anything during the discussion  
//   or modifies any computer file during the discussion. I have violated  
//   neither the spirit nor letter of this restriction.  
//
```

Programs that do not contain this pledge will not be graded.