CSC 310, Spring 2002 — Assignment #2

Due at **start** of tutorial on March 8. Worth 10% of the course grade.

Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own.

Your task for this assignment is to write programs to compress and decompress web pages, which are written in HTML. An HTML document is composed of text, but with various special commands that control formatting and specify links to other documents. Because of these special commands, and because the content of web pages tends to be of certain types, a specialized compression program designed specifically for web pages might be useful. You are to write one that uses adaptive models described below, and which encodes using arithmetic coding.

Procedures written in C for performing arithmetic coding and for maintaining symbol frequencies are supplied to you. They may be obtained from the directory `/u/radford/310` on CDF, or from the course web page. It is probably easiest for you to just copy these files to your own directory, and then work with them there. The file `Documentation` in this directory explains how to use these procedures.

You should write three C programs: A program `html-encode` that encodes an HTML file, a program `html-decode` that decodes a file encoded by `html-encode`, and a program `html-freq` that collects statistics from a number of HTML files for use in encoding (and decoding) other HTML files.

These programs should operate as described below. Square brackets enclose optional arguments; dots indicate possible repetition.

```
html-encode [ -h ] [ -p ] [ stats-file ] < HTML-file > encoded-file
```

This program reads an HTML file from standard input, and writes an encoded version of this file, hopefully of smaller size, to standard output. The optional `-h` and `-p` arguments control how many contexts are used. If `-h` is specified, separate models are kept for normal text, text inside an HTML directive, and text inside a string inside an HTML directive. If `-p` is specified, characters are coded using frequencies conditional on the preceding character. It is legal to specify both or neither of `-h` and `-p`. The optional `stats-file` may contain initial frequencies to use; if it is omitted, frequencies are initialized to one.

```
html-decode [ -h ] [ -p ] [ stats-file ] < encoded-file > HTML-file
```

This program reads a file encoded by `html-encode` from standard input, and writes the decoded version of this file to standard output. The meaning of the optional `-h`, `-p`, and `stats-file` arguments is as for `html-encode`. For decoding to be successful, the *same* options must be given to `html-decode` as were given to `html-encode`.

`html-freq [ -h ] [ -p ] stats-file HTML-file ...`

> This program reads all the HTML files given as arguments, collecting statistics on frequencies in different contexts, and writes these statistics to the `stats-file` given. This file may be given as an argument to `html-encode` and `html-decode` for use in compressing other HTML files. The meaning of the optional `-h` and `-p` arguments is as for `html-encode`. When `html-encode` and `html-decode` use the `stats-file` produced, they must be passed the *same* options as were passed to `html-freq` when this file was created.

The contexts used when `-h` is specified are defined as follows. Initially, we are in context 0. When a '<' character is encountered in context 0, we switch to context 1. We switch back from context 1 to context 0 when a '>' character is encountered. If we encounter a double-quote character, '"', when in context 1, we switch to context 2. We switch from context 2 back to context 1 when a second '"' character is encountered. Seeing a '>' character when in context 2 does *not* cause the context to change.

When neither `-h` nor `-p` is specified, a single context is used (ie, a single "frequencies" structure, for the "freq" module). If no `stats-file` is given, the `html-encode` program should then operate the same as the demonstration `encode` program. When `-h` is specified, but not `-p`, there will be three contexts. When `-p` is specified, but not `-h`, there will be 256 contexts, one for each possible preceding (8-bit) character. (The context for the space character should be assumed at the start of a file.) When both `-h` and `-p` are specified, there will be $3 \times 256 = 768$ contexts.

The `stats-file` produced by `html-freq` will contain the "frequencies" structures for whatever contexts where specified. These structures can be written out in binary format using the `fwrite` procedure (and later read using `fread`).

You should test your program using the HTML files in the directory `/u/radford/test` on CDF. Your tests should first, of course, confirm that encoding a file and then decoding it results in the original file. (The `cmp` program is useful for this.) After that is established, you should see how large the encoded files are, comparing the results using all the combinations of options (there are eight combinations in all). To create a `stats-file`, you should run `html-stats` on the entire set of HTML documents in the directory `/u/radford/train`. Finally, you should compare with how well the standard `gzip` program does. (If `gzip` is run with no arguments, it encodes standard input and writes to standard output; `gunzip` does the decoding.)

After assessing how well the programs described above work, you should invent some modification of the procedure yourself (eg, some different context, or larger number of contexts), and assess how well your new idea works. Finally, you should discuss reasons why the results came out the way they did, in terms of the characteristics of HTML documents.

You should hand in your program, properly formatted and documented (but not excessively documented), the results of your tests, and your discussion of the results, in paper form.