Keefer Rourke
Mr. Fryer-Davis

15 June 2015
ICS4UI

Text Compression and Decompression – Testing

While this project was not completed in its entirety, a significant portion of the project has been completed and is functional. The programme, when executed in “compression” mode, successfully reads in a text file, builds a frequency table and Huffman tree, generates variable length codes, and creates a binary file containing the generated data. It however cannot decompress the data when executed in “extraction” mode.

Note that this programme was developed and tested exclusively on GNU/Linux, using the C++ standard library.

Compression mode

Example programme usage in compression mode:

```
huffpuff -c [inputfile] (outputfile)
```

As above, my programme (dubbed 'huffpuff') will read the contents of the text 'inputfile' and write the encoded contents to the binary 'outputfile'. During development, the integers that were being written to the file were printed to the console, so they could be verified as the contents of the binary file using a hex editor/viewer such as `GHex`.

This was tested thoroughly using several files of different sizes ranging from 0 B in size to 72 kB in size. These files have been included in the project archive under the “test-files” directory.

In test cases where the input file is 0B, the program will print an error and quit, as it cannot operate on an empty file.

Example test cases:

On empty file:

```
./huffpuff -c test-files/empty.txt test-files/empty.bin
      Error: test-files/empty.txt is an empty file.
```

On non-existent file:

```
./huffpuff -c nonexistingfile.txt
      Error: nonexistingfile.txt is an empty file.
```

On 1 byte file:

```
./huffpuff -c test-files/nl.txt test-files/nl.bin

      : 0                                       <-- this is the newline character
      : 1                                       <-- this is the EOF character
      Successfully opened test-files/nl.bin
      bitsize of header: 19
      wrote header to file
      bitsize of encoded text: 2
      wrote encoded text stream to file
```

In this test case, and others where the input file size is very small, the output file is much larger than the input file because the Huffman tree must also be stored in the output file. The import thing here though is that the file contents are compressed from 1 byte to 2 bits.

Note that this file, though when opened appears empty, contains one empty line, which takes up one byte. It was created with`touch test-files/nl.txt; echo “” > test-files/nl.txt` on Ubuntu 15.04.

On the GPLv3 (35.1 kB):

```
./huffpuff -c LICENSE.txt test-files/license.bin

      [variable length codes here]
      Successfully opened test-files/license.bin
      bitsize of header: 769
      wrote header to file
      bitsize of encoded text: 162029
      wrote encoded text stream to file
```

In this test case the input file (35.1 kB) is compressed to the file 'out.bin' with a size of 20.4 kB, approximately 42% savings.

On dummy text (71.9 kB):

```
./huffpuff -c test-files/lorem_ipsum.txt test-files/lorem_ipsum.bin

      [variable length codes here]
      Successfully opened test-files/lorem-ipsum.bin
      bitsize of header: 419
      wrote header to file
      bitsize of encoded text: 307806
      wrote encoded text stream to file
```

In this test case the input file (71.9 kB) is compressed to 38.5 kB, approximately 46% savings. This file was generated from <http://lipsum.com>.

On existing binary file (nl.bin from above):

```
./huffpuff -c test-files/nl.bin test-files/out.bin

      [variable length codes here]
      Successfully opened test-files/out.bin
      bitsize of header: 69
      wrote header to file
      bitsize of encoded text: 34
      wrote encoded text stream to file
```

In this test case, the input file is an existing binary file. `huffpuff` can't distinguish if the file has already been compressed, and instead reads in the binary (interpreting it as ASCII) and compressed it again. Presumably this could then be decompressed twice to get back to the original file, or safeguards could be put in place (only accepting certain mime-types for instance) to forbid this behaviour.

Keefer Rourke
Mr. Fryer-Davis
15 June 2015
ICS4UI

Extraction mode

Example program usage in compression mode:

```
huffpuff -x [inputfile] (outputfile)
```

As above, when the program is called in this fashion, it should read the contents of the binary input file, decode them, and write the decoded text stream to a new plain-text file. However, since this is not yet implemented, when the program is called with these parameters, it does nothing. The outline of what should happen is described in the source file 'puff.hpp'.

If this function were implemented it would be tested by running the program first in the compression mode to produce a compressed binary file. The program would be then be run as

```
./huffpuff -x input.bin output.txt
```

making sure to specify a new unique output file name, to produce what should be an exact copy of the original file.

The correctness of this decompression can be easily compared either by manual inspection for small files, or through the use of a standard utility such as `diff` from GNU diffutils.
If the command `` `diff original-file.txt extracted-file.txt` `` returns nothing, then the files are the same, and the extraction method of `huffpuff` works properly.

A full overview of this testing process is as below:

```
./huffpuff -c input.txt out.bin
./huffpuff -x out.bin extracted_text.txt
diff input.txt extracted_text.txt
```

The test would be considered a pass if diff returned nothing to the console, and a fail if it returned something to the console.

Improper usage

If the programme is used improperly, for instance if bad arguments are provided, the programme is directed to the print the usage of the programme to the console and then terminate. First the programme checks for number of arguments, if there are too few or too many, it quits. If the number of arguments passes, then the validity of the arguments is checked.

Examples of improper usage and respective error messages:

```
./huffpuff                              <-- no arguments provided
      Error. Invalid number of arguments.
./huffpuff -c                           <-- too few arguments provided
      Error. Invalid number of arguments.
./huffpuff -x input output thirdfile    <-- too many arguments provided
      Error. Invalid number of arguments.
./huffpuff -s input output              <-- invalid argument provided [-s]
      Error. Bad arguments.
```

Usage message printed after any of the above errors is as follows on the next page:

Keefer Rourke
Mr. Fryer-Davis

15 June 2015
ICS4UI

```
NAME
   huffpuff - a Huffman coding implementation

SYNOPSIS
   huffpuff [-c] [--compress] [-x] [--extract] [--decompress]
   [--inflate] file ...

DESCRIPTION
   Compress plain-text files, and decompress Huffman binary files
   created by this programme.

OPTIONS
   Mandatory arguments are as follows, plus the input file name.

   -c, --compress
       compress a plain-text file to a smaller binary file

   -x, --extract, --decompress, --inflate
       decompress a binary file back to plain-text

   Optionally an output file name can be specified (see usage)

USAGE EXAMPLES
   huffpuff -c inputfile.txt
   huffpuff -x inputfile.bin
   huffpuff --compress inputfile.txt outputfile.bin
   huffpuff --inflate inputfile.bin outputfile.txt
```