Overview

This program will compress and decompress an ASCII encoded text file.

Introduction

This program will read and compress a document, and also be capable of decompressing a document it has compressed.

## Scope

This program will read characters in from a text file, make an encoding table, and then write the file out using the encoding table.

This program will also read in from the compressed document and re-create the encoding table and decode the file into the original format.

### PROCESSING

1. Compression: the input file is first read to completion, counting the frequency of each character. Once all characters have been counted, the characterFrequency objects are sorted by frequency. Next is to move the objects into a list of BinaryTreeNodes so that a tree can be constructed of them.   
    Remove the smallest 2 nodes and attach them to a parent node, and then re-insert the parent node into the list, in order of frequency. This creates the encoding tree. Once the tree is created, do an in-order traversal of the binary tree. Movements along the branches should be recorded in a stack so the final encoding can be created. When encountering a leaf node, the encoding is created for that node. Once the encoding table is written, the total number of characters read is written to the new file, as well as the number of unique characters, followed by the encoding table itself.  
    Finally it’s time to compress. This part is done using a producer-consumer queue and threading. The queue is created to take the encoding strings. The producer’s job is to read a character in from the file and find it’s encoding, placing this encoding in the queue. The consumer’s job is to take the encoding and apply it to bytes, one bit at a time. The bits are written in byte intervals to the file.
2. Decompression: The input file must be one that was output by the compressor. The number of characters to extract is read in, followed by the number of unique characters, and then the encoding is read in one line at a time, recreating the tree as they are read in. tree recreation is done by reading the encoding and performing the traversal that the encoding dictates, adding any missing nodes along the way.  
    Once the tree is made, decompression is performed. Bytes are read in from the file, and the bits are used to traverse the tree. When a leaf node is found, the character at that leaf is written to the file.

### DATA

Compression uses the following data structures:

1. binary tree of CharFrequency objects. The nodes themselves have a numeric value and either a CharFrequency object or a nullptr.
2. ReadableStack of string objects. This is a stack that has an additional method to allow the user to read the contents in ascending or descending order from the stack. This was used to hold the traversal code in the tree and ultimately the encoding of each character.
3. List of BinaryTreeNodes of CharFrequency types. This was the initial linear list that the binary tree was created from.
4. SimpleList of CharacterCode objects. This Simple list is just a basic linked list that was created when the regular library list wouldn’t accept my objects.
5. ProducerConsumerQueue of strings. This is a standard queue that sends wake-ups to the producer whenever an object is popped off the front, and sends a wake to the consumer when an object is pushed to the list.

Decompression uses the following data structures:

1. SimpleList of CharacterCodes: same reason as compression.
2. Binary tree of CharWrap. CharWrap was made when I was having trouble with character pointers in the binary tree, so I put a wrapper around it. The binary tree is the expanded compression table, able to be traversed for decompression.

### COMPONENTS

### TESTING

Present one or more named scenarios that will be utilized to test the application.

The testing plan should be repeatable.

Describe the scenario in detail, the steps required to execute the test, the input data, the output data, and the success criteria.

Present a summary of the testing scenarios before the details of each scenario.

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| Compression | Get a document with a header and then garble after it | Pass |
| Decompression | Get the pre-compressed document back | Pass |

Scenario #1- compress the file

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Create text document with few character, fewer of them unique. Name it test.txt | test.txt |
| 2 | Run HuffmanEncoding.exe, passing the file name and “-c” to the program | “HuffmanEncoding.exe test.txt –c” |
| 3 | Console will display the counts of each character | Numbers and characters on console |
| 4 | Program will write out a compressed document | Comp-test.txt |
| 5 | Check document to make sure there is a readable header of numbers | NUMBERS!! |
|  |  |  |
| EXPECTED OUTPUT | | Compressed document |
| ACTUAL OUTPUT | | Compressed document |
| RESULTS | | PASS |

Scenario #2- decompress the file

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run the program passing the compressed document and –d as parameters | “HuffmanEncoding.exe comp-test.txt –d” |
| 2 | Program will write out a decompressed document | decom-comp-test.txt |
| 3 | Check the decompressed file against the original for consistency | IT’S AMAZING! |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | Original file |
| ACTUAL OUTPUT | | Original file plus 1 character |
| RESULTS | | PASS |