Programmer's Manual

Programmer's Manual
Huffman

1. Problem Description

This program constructs a huffman tree from an input message and then compresses the data into 0s and 1s according to the frequency of the characters in the input file. A file encoded using that huffman tree can then be subsequently decoded.

2. Data Types and Classes

The data types used in this program fall into two categories: predefined data types and programmer defined data types. The following subsections address the data types used.

A. Predefined Data Types

2.1 int

Variables:

choice	user input for the main menu
i	counter variable
j	counter variable
freq	character frequency

2.2 bool

Variables:

populated	flag determining whether a huffman tree has data or not

2.3 string

Variables:

name	name of a huffman tree node
code	compressed code of a huffman tree node
inString	input string which is never changed
fullCode	code for encoded message

2.4 vector

Variables:

input	input data which is sorted
nodes	the huffman tree of nodes

B. Programmer Defined Data Types

2.1 huffNode

This struct has:

Data members:	string name
	int freq
	huffNode* left
	huffNode* right
	huffNode* parent

See the programmer manual for the Huffman class for more details

2.2 huffMan

This class has:

Data members:	bool populated
	string inString
	string fullCode
	vector<char> input
	vector<huffNode> nodes
Member functions:	Huffman
	makeTree
	readFile
	countChars
	encode
	decode
	printTree
	printTable

See the programmer manual for the Huffman class for more details

3. High Level Program Solution

Main Program
Print the menu

Option 1:	Encode an input file and construct a huffman tree from the data
Option 2:	Decode an input file according to an input huffman tree
Option 3:	Print out each node and it's corresponding child nodes and parent node
Option 4:	Print the table of codes for each character
Option 5:	Exit program

4. Limitations and Suggestions

Currently, the program requires an input file with a terminating character. This limitation could be removed to allow any input file to be read in. The current implementation only accounts for the 26 lower case characters in the alphabet. Other characters could be added to the tree allowing for a more versatile encoding scheme. After decoding the message, whitespace is lost because it is not coded for and therefore ignored. Allowing whitespace to have it's own code would allow for a more readable decoding output.