Huffman Code Trees
CS101 Project 4
Due 4/15

Phase 1 (85 points):
You are to write a program that reads a plain text file, computes a Huffman code tree for that text file, and writes out the encoded version of the text.

- Your program should read the text from the file given as a command-line argument. You may assume that no more than 100,000 characters are in the file.
- You should compute the number of occurrences of each character in the file, and each character (and its frequency) should be placed in a new tree node.
- You should build a min-heap containing these nodes.
- Build the Huffman Code Tree using the heap. As you create new internal nodes, give them a unique integer label.
- Write the pre-order traversal of the Huffman code tree to the file "preorder1.txt" and the in-order traversal to the file "inorder1.txt". As you write out the nodes in the traversal, write each node on a line. If the node is an internal node, then write an 'i' followed by the integer for that node. If the node is a leaf, write an 's' followed be the ascii values of the letter(s) represented by that node.
- Construct a table containing the encoding for each character, storing the encoding as a string.
- Encode the original text, writing the encoded version to "code1.txt". This file should be ASCII '0' and '1' characters (much easier to debug)
- Lastly, compute and write out the number of bits (0s and 1s) in the file above to the standard output.

Requirements:

- You should build all the data structures that you use yourself. You must create a binary heap data structure that uses an array and implements insert and extract-min that run in O(lg N) time.

Phase 2 (25 points):

Modify your phase 1 program as follows:

- We will allow pairs of characters to be treated as a new kind of character.
- We will try a brute force approach to this problem: In order of decreasing frequency, for every adjacent pair of characters that appear in the text file, try to encode the text using that pair as a new character. If it reduces the size of the encoded output, then use that character pair. Repeat this for every pair of characters, keeping those that help.
- Repeat the output above, using the files preorder2.txt, inorder2.txt and code2.txt.
- Repeat the output to standard output containing the number of bits in the final code2.txt.

- The only include files allowed are string, iostream and fstream.
- Your makefile should build the executable named “encode”.
- Zip all of your source code and makefile into a single .zip file for submission.
- You must use good object based organization, i.e. use classes in an appropriate way.

Example
Suppose that the file “foo.txt” contains the following text:
ALLALABAMAFOOTBALL

Then executing:
encode foo.txt
should produce output files such as the following. Note that this is only an example of a correct output. The tree and codes produced by your program would likely be different.
Standard Output (to the screen):
Phase 1 code length: 45 bits
Phase 2 code length: 27 bits

preorder1.txt:
i 133
i 132
s 76
s 65
i 131
i 130
s 66
s 79
i 129
i 128
s 84
s 70
s 77

inorder1.txt:
s 76
i 132
s 65
i 133
s 66
i 130
s 79
i 131
s 84
i 128
s 70
i 129
s 77

code1.txt:
010000010001100011110111011011011100100010000

Here is a simple example for phase 2 this one is deliberately simple so that you can easily do this one by hand:

Suppose that the file infile.txt contains:
ooaooboo

then executing encode infile.txt should produce:

Phase 1 code length: 10 bits
Phase 2 code length: 7 bits

Preorder2.txt:
i 129
i 130
s 98
s 97
s 111 111

inorder2.txt:
s 98
i 130
s 97
i 129
s 111 111

code2.txt:
1011001

For the ALLALBAMAFOOTBALL text above,
The code length is reduced to 27 bits. This happens as follows.
The pair AL appears 3 times. Using the letter pair AL reduces the total encoding to 44 bits.

There are a number of pairs that all occur only once. Let's consider these in alphabetical order.

Using the pair AB reduces the code length to 43 bits.

Using the pair AF reduces the code length to 40 bits.

Using the pair AM reduces the code length to 35 bits.

Using the pair OO reduces the code length to 32 bits.

Using the pair TB reduces the code length to 27 bits.
There are no more pairs that are possible in the message at this point.

This results in the output files below. Since this is a Huffman code tree, these files are not unique. You will notice that decode reproduces the original text.

The preorder2.txt file:

```
i 128
i 129
i 130
s 65 66
s 65 77
i 131
s 65 70
s 79 79
i 132
i 133
s 84 66
s 76
s 65 76
```

The inorder2.txt file:

```
s 65 66
i 130
s 65 77
i 129
s 65 70
i 131
s 79 79
i 128
s 84 66
i 133
s 76
i 132
s 65 76
```

The code2.txt file:

```
11101110000010100111001101
```