CSCI-650 **Project 3** **Huffman Encoding Trie**

In this project, you will implement the Huffman algorithm.

Input: A text file that includes alphabet characters, commas, semicolons, colons, spaces, periods (no new line characters, apostrophes).

Example:

```
abbccc dddd eeeee ffffff, ggggggg
```

Output:

1. A text output file with an encoding for each character in the file (printed in alphabetical order) in the format: <upper case character><space><bit encoding><endl>. For example (this is output file for the input file example above):

```
  011
, 10110
A 10111
B 1010
C 010
D 100
E 110
F 111
G 00
98
264
```

Note, the first line contains a space ' ' character in this example.

The last two lines are integers in the format: <int><endl>.

The first integer is the total number of bits required to encode the given input calculated by the formula

$$\min_{\substack{\text{binary trees } T \\ \text{with } |\Sigma| \text{ leaves}}} \sum_{c \in \Sigma} f(c) d_T(c)$$

where f(c) is a frequency of character ***c*** in the input file and d(c) is the length of the encoding for ***c*** (number of bits in the ***c***'s encoding). In our example, 98 is the total number of bits required to encode the given input file.

The second integer in the last line of the output file represents the total number of bits in the original file (each character is represented by 8 bits). This number is calculated by the same formula as above, except d(c) is the same for all characters and equal to 8. In our example, the total number of bits that original input file takes is 264.

For your output to match test output, convert all characters in the input file to the upper case using ***toupper***(char ch) function; and then output the characters in alphabetical order (apply alphabetical order to spaces, commas and other characters in the input text). Before building a Huffman encoding trie, convert all characters to upper case.

Specification Requirements:

1. To build a Huffman encoding trie, you will to use classes Node and Trie. The class Node will have public data members: Node **prev (left),* Node *next (*right*), int priority, and character *value.* The class Trie will have a private data member Node **root.* You can use any member functions for Trie class.

2. You must use command line arguments. For this use the following declaration for the main function:
int main(int argc, char *argv[])
The *argc* holds the total number of arguments including the name of executable, and the array *argv* holds strings, the command line arguments. A user should be able to run your program with the following command line:

```
./huffman  input.txt output.txt
```

Then argv[0] is "./huffman", argv[1] is "input.txt", argv[2] is "output.txt". In other words, you can access the input file inside of your program using code like:
string input = argv[1];

Submission: Submit the following files via *turnin* system:
trie.h
trie.cpp
main.cpp

Provided files:
node.h,
dlist.h
dlist.cpp
priority_queue.h