CS 199 (133), Summer 2021
Programming Project #8: Huffman Coding Bonus (4 points)
Due Friday, August 27, 2021, 11:59 PM

There is an extra credit option for this assignment that is worth a measly 4 points. In other words, this isn't intended as an opportunity for you to increase your grade. It is intended as an extra coding exercise for those who are interested in exploring how to make their Huffman program behave better.

If you do the extra credit option, you are still required to complete the standard `HuffmanTree` and to submit it along with your `HuffmanNode`. So if you work on this, do so only after you have completed the standard assignment. To keep things clear, for this part of the assignment you should create a class called `HuffmanTree2`. You can copy your `HuffmanTree` class and modify it appropriately to get the initial version of this class.

The main goal of this variation is to eliminate the code file. When you use a utility like zip, you don't expect it to produce two output files (a code file and a binary file). You expect it to produce one file. That's what we'll do in this variation. To do so, we'll have to be able to include information in the binary file about the tree and its structure.

In all, you will have to include the following two new functions in your class along with the other functions we had in `HuffmanTree`:

Member Function	**Description**
`HuffmanTree2(IBitStream* input)`	Constructs a Huffman tree from the given input stream. Assumes that the standard bit representation has been used for the tree.
`void writeHeader(OBitStream* output)`	Writes the current tree to the output stream using the standard bit representation.

In the original `HuffmanTree` we had a function called `createEncodings` that would create and return a `map` of the characters to their encodings. However, it was `huffmanMain`, not `HuffmanTree` that would write the codes to an output file.

In this version of the program, instead of writing the codes to a separate file, `huffmanMain2` will call a new function you will write in your `HuffmanTree` class called `writeHeader`. The idea is to write to the `IBitStream` a representation of the tree that can be used to reconstruct it later. As we did with the `QuestionTree` in assignment 4, we can print the tree using a preorder traversal. For a branch node, we write a 0 indicating that it is a branch. We don't need to write anything more, because the branch nodes contain no data. For a leaf node, we will write a 1. Then we need to write the ASCII value of the character stored at this leaf. There are many ways to do this. We basically need to write some bits that can be read later to reconstruct the character value. The value will require up to 8 bits to write. Although we could write a smaller number in less bits, write out every number in exactly 8 bits so that you will know how many bits to read when you decode the file.

When encoding, `huffmanMain2` produces a binary file that first has a header with information about the tree and then has the individual codes for the characters of the file. When decoding the program has to use this information to reconstruct the original file. It begins by calling the constructor listed in the table above, asking your class to read the header information and reconstruct the tree. Once the tree has been reconstructed, the program calls your decompress function from the original assignment to reproduce the original file.

You will need `huffmanMain2.cpp`, and examples of encoded input files called `short.bonus` and `hamlet.bonus` which you can find linked from the project page of the course website. `huffmanMain2` should produce exactly the same output when used with your version of `HuffmanTree2`. There is a separate turn-in for the bonus in which you can submit your `HuffmanNode` and `HuffmanTree2` `cpp` and `h` files (you shouldn't need to make any changes to your node class for the bonus, but it will be easier for us to grade if you submit both when you turn it in).