ECE 36800 – Data Structures
Programming Assignment 4 - Part 1

Purpose of Assignment 4:
Build a Huffman tree for encoding and decoding a text file.

Goal of Part 1:
Build a Huffman tree and generate Huffman codes for 26 English alphabets.

What to submit:

1. `huffman1.cpp`: Your test program to construct the Huffman coding tree. Start with the given file and add your implementation.
2. A word document `proj4p1.docx`: This file should include the printout of your program with `probability.txt` as the input text file. Include a copy (or screenshot) of the printout of the program run.
3. Push all your files under the "proj4/part1" directory before the deadline.

Other available files:

1. `node.h`: The header file for the new node class. You may add new functions to `node.h` or add a new `node.cpp` file if needed.
2. `probability.txt`: The 26 English alphabetic letters and their probabilities are listed.
3. `Makefile:` rules to compile the source files for your convenience.

Guideline:
There are two steps in your program:

- Step 1: Read from the file (`probability.txt`) the average occurrence frequencies of 26 English alphabetic letters
- Step 2: Construct a Huffman coding tree and assign Huffman codes for these letters.
  Note: Please have your program print out the code for each letter on screen.

Explanation/Suggestions of Implementation:

- After reading from `probability.txt` file, 26 nodes (each one corresponding to a letter) should be created. At the same time, a list of node pointers called `node_list` is created, where each node pointers refers to each of the 26 nodes. This list can be sorted in ascending order of probability using its member function `sort`, with a Boolean function `comp_prob` to compare elements. The `comp_prob` function is provided in `node.h`.
- After the sort and merge iterative steps, a Huffman coding tree should be built, where the leaf nodes are the 26 nodes with corresponding letters and probability values. The internal nodes in the tree are created as the result of the `combine` function (in `node.h`) during the merge process. This tree can be accessed through the `root` node pointer.
- The assignment of codes to the nodes on the tree can be done with a single traversal of the tree. This can be easily implemented using a recursive function.

- Please create a string array, e.g., `string codearray[26]`, to store the code string of each English character: the code for 'a' stored at index 0, and code for 'b' stored at index 1, etc. In this way, the code for each letter can be retrieved through direct indexing into the array.
- For the convenience of grading, please have your program print the code strings of all the letters.
- In the end, the code calls the recursive function `clean(root)` to release the memory space of all nodes on the tree.

Grading Policy:

Part 1 counts for 50% of the overall points in Programming Assignment 4.

Please make sure your program compiles successfully. If not, 25 points will be deducted.

1. **Executability** (5%):
   - Runtime errors: You program must not have runtime errors (e.g., code crash, infinite loop, reading uninitialized memory, accessing the content of a NULL pointer, etc.).
2. **Programming style** (5%):
   - Code efficiency: Code should use the best approach in every case.
   - Readability: Code should be clean, understandable, and well-organized. Please pay special attention to indentation, use of whitespace, variable naming, and organization.
   - Documentation: Code should be well-commented with file header and comments.
3. **Program Specifications/Correctness** (40%):

   Please refer to the Grading Criteria table for details. Specifically, your program should behave correctly, adhere to the instructions, and pass the test program.

file	**item**	**weight (%)**
huffman1.cpp (and node.h/node.cpp if needed)	construct Huffman coding tree	15
	generate and print out Huffman codes	15
proj4p1.docx	running result	10
	total	**40**