CS302

Assignment 4: Prefix Code

Description

Ah yes, compression algorithms, this allows us to decrease the amount of space needed to store data without losing anything. For this assignment you will be implementing Huffman's Algorithm that computes the prefix code for a set of letters and their frequencies. You will need to create the following binary tree class to implement the algorithm.

```
class binTree
{
  struct binTreeNode
  {
    char letter;
    int frequency;
    binTreeNode * left, * right;
  };

public:
  binTree();
  binTree(char, int);
  binTree(binTree*, binTree*);
  ~binTree();
  int getFrequency() const;
  std::string getPrefixCode(char);
private:
  std::string getPrefixCode(binTreeNode*, char);
  void destroyTree(binTreeNode*);
  binTreeNode * root;
};
```

Each member contains/performs the following

- `struct binTreeNode` - represents a node of the binary tree, each node contains a letter along with its frequency and two children pointers
- `binTreeNode * root` - a pointer that points to the root node of the binary tree
- `binTree::binTree()` - default constructor that sets `root = nullptr`

- `binTree::binTree(char letter, int frequency)` - constructor that allocates a `new binTreeNode` object to the root pointer, and sets the node's letter and frequency with the two parameters and this would be a leaf node, set both of the children pointers with `nullptr`
- `binTree::binTree(binTree * t1, binTree * t2)` - constructor that combines two trees together, this function allocates a `new binTreeNode` to the root pointer, set its letter with `'\0'` (a null character) sets its left child pointer with the root of `t1`, sets the right child pointer with the root of `t2`, and its frequency is the sum of the frequencies of the roots of `t1` and `t2`
- `binTree::~binTree()` - destructor, calls `destroyTree(root)`
- `void binTree::destroyTree(binTreeNode * r)` - a recursive function that deallocates the tree in a postorder fashion
- `int binTree::getFrequency() const` - returns the frequency of the root
- `std::string binTree::getPrefixCode(char letter)` - returns the binary string that represents the prefix code of the letter passed in as a parameter, this function makes a call to `getPrefixCode(binTreeNode*, char)` function and returns its result
- `std::string binTree::getPrefixCode(binTreeNode * r, char letter)` - a recursive function that finds the leaf node with the matching letter and ultimately returns the the prefix string up the recursive call tree

Input

You will prompt the user for an input file and using an `std::ifstream` variable read from a file. The file will contain 30 lines where each line contains an integer (representing a letter's ASCII value) and an integer (representing its frequency) separated by a blank space. You letters you will need to decipher are letters a-z, a dot, a comma, a space, and a new line.

Then you will prompt the user for another input file, the file contains compressed data that you will need to decipher/decompress into the original format. The input file will be one huge line full of 1s and 0s.

Output

For each input, output the deciphered message to the console.

Contents of Main

In main, when implementing Huffman's Algorithm, you will need to store an array of binary trees, in order to avoid needing to implement a deep copy of a binary tree would be to store pointer of binary tree objects (since you will need to remove binary trees and combine them and put them back into the list), thus the following would be a good idea to declare either a `vector` or `list`

```
std::vector<binTree*> forest;
std::list<binTree*> forest;
```

The name forest makes sense since a forest is just a set of trees 😄 Initially you will have 30 pointers that point to a binary tree object (each object will contain a single node with a letter and frequency read from the file), and while the forest contains more than 1 pointer to a binary tree, you find and remove two pointers from the list (one with the smallest root frequency and the other the second smallest root frequency), then you combine them using one of the constructors of the `binTree` class and push this pointer to the back of the forest. After running this 29 times you would have `forest[0]` is the pointer that points to the complete Huffman Tree.

Then you need to get the prefix code of each letter using `forest[0].getPrefixCode(letter)` and store them into a list of size 30. And now you traverse the compressed file and decipher it and output the results to the output file.

Specifications

- Do not use global variables
- The only STL you can use in this program is `std::vector` or `std::list` to store a list of `binTree` pointers, no other STL is allowed
- As always, document your code

Sample Run

```
% g++ main.cpp binTree.cpp

% ./a.out

Enter filename: letters.txt
Enter encrypted file: have_fun.txt

glhf
```

Submission

Submit the source files to code grade by the deadline, just submit binTree.cpp and main.cpp, no need to submit the header file

References

- Supplemental Video https://youtu.be/jhplrdQfm00
- Link to the top image can be found at https://www.flaticon.com/free-icon/morse-code_179895