1. Clear specification of how you plan to store the data from file within your program

```
struct Gene {
        string gene;
        Mutation mutation[];
};
```

```
struct Mutation {
        string m_mutation;
        int energy;
        Gene *source_gene;
};
```

A particular gene could have 0 or up to 5 possible mutations

2. Clear delineation between high-level tasks and sub-tasks (this can be accomplished simply by using indented lists)

- main
    - Prompt the user to enter the name of the input file
    - Read in gene data from file
    - queries
        - Create const variables for each query
        - Create variables for the queries that require multiple arguments
        - Prompt user for query input
        - If query matches to into that if statement, call appropriate function
- read_sample
    - Open file, if failed error message
    - Get first line, total number of genes in file
    - Allocate and populate memory for gene
    - Populate memory for mutations
- init_gene
    - Set gene and mutation values to empty
- populate_gene
    - Use pointer to read the input data from file. Each line of the input file contains information about a gene and its mutations.
    - Use stringstream to extract individual value from file
    - Iterate through genes
    - Get first gene
    - Get mutations and energy
    - Clear stream
- populate_mutations
    - Use pointer to read the input data from file. Each line of the input file contains information about a gene and its mutations.
    - Use stringstream to extract individual value from file
    - Iterate through genes
    - Get first gene
    - Get mutations and energy
    - Clear stream
- print_gene
    - Iterate through num_genes
    - Print name of gene
    - Print mutations and cost associated with the gene
    - If no mutations, print “none” under mutations header
- can_mutate
    - Iterate through num_genes
    - Iterate through mutations
    - If source gene matches gene in file
        - Look to see if target gene is associated with source gene
    - If source gene is not in file, or if target gene is not associated output

        - [Source] cannot mutate into [Target]
- mutation_w_energy
    - Iterate through num_genes
    - Iterate through mutations
    - If source gene matches gene in file and target gene is associated with source gene
        - Check if amount of energy can transform source gene

3. Subtasks (or possibly sub-sub-tasks) should be detailed enough to correspond to only a small chunk of code (typically between 5-10 lines)

- main
    - Prompt the user to enter the name of the input file.
        - Create string to hold filename
    - Read in gene data from file, store in int variable
    - Call read_sample function
    - Create variables for query arguments
    - Prompt user for query and arguments
    - While query != quit, keep prompting user for query
    - If query != quit, call function associated with query
- read_sample
    - Open file
        - Use the input file name to open the file in read mode.
        - If the file opening fails, output an error message.
    - Get first line, total number of genes in file
        - Read the first line of the file.
        - Extract the total number of genes from the first line.
    - Allocate and populate memory for gene
        - Allocate memory for an array of genes based on the total number of genes obtained from the first line.
    - Iterate through the file and populate the gene data structure:
        - Start a loop to iterate through the remaining lines of the file.
        - For each line:
            - Extract the gene information (gene, mutations, energy) from the line.
            - Assign the gene information to the corresponding element in the gene data structure array.
    - Reset the file stream for another pass
        - Close file
        - Open file
        - Overwrite infile >num_genes
        - infile.ignore
- populate_gene
    - Use pointer to read the input data from file. Each line of the input file contains information about a gene and its mutations.
    - Use stringstream to extract individual value from file
        - string gene_sequence, line_info;
        - stringstream sstream;
    - Iterate through genes
        - for(int i = 0; i < num_genes; i++)
    - Get first gene

        - Getline
        - sstream >> gene_sequence;
        - gene_data[i].gene = gene_sequence;
    - Get mutations and energy
        - Create int index for setting mutation values
        - gene_data[i].mutations[mutation_index].m_mutation = gene_sequence;
        - sstream >> gene_data[i].mutations[mutation_index].m_energy;
    - Clear stream
- populate_mutations
    - Create string variables for entire line, and each piece of information on line
    - Use stringstream for the first pass and skip over the gene_sequence
        - Create a string variable to hold line_info and first gene_sequence
        - Create sstream variable to pull from line and store in values from line
        - Walk over array
            - Get whole line from file stream
            - Load up stringstream with that line
            - Read not just gene_sequence but also the other strings in the line
            - While loop, as long as sstream is still populating line
                - If gene_sequence does not have mutations, condition will be triggered
            - Get mutation and energy values
                - Gene_data[i].mutations[mutation_index].m_mutation = gene_mutations;
                - gene_data[i].mutations[mutation_index].m_energy = gene_energy;
- print_gene
    - Iterate through num_genes
        - Print gene
    - Create bool to mark if gene has mutations
    - Loop over mutations
        - If mutations is not empty print mutation and energy value
        - Mark mutations true
        - Break from loop
    - If no mutations, print “none” under mutations header
- can_mutate
    - Create bool to mark if mutation is possible
    - Iterate through num_genes
        - Check if input gene is equal to gene in Gene struct
    - Iterate through mutations
        - If target gene is equal to mutation gene mark bool as true
        - Print that source gene can mutate to target gene
        - Break from loop

        - If source gene is not in file, or if target gene is not associated output, bool is false
            - Print [Source] cannot mutate into [Target]
- Mutation_w_energy
    - Create bool to see if mutation with energy is possible, set to false
    - Iterate through num_genes
        - Check if source gene is equal to gene
    - Iterate through mutations
        - If target gene is equal to mutation, check for energy value
        - Energy should be greater or equal to the energy needed to mutate
        - If conditions match mark bool as true
        - Print source gene can transform to target gene with energy
    - If source gene can transform to target gene but doesnt have enough energy
        - Print it can transform but not with input energy
        - Mark bool as true
    - If input doesnt fulfill conditions then bool is false
        - Print source gene cannot transform to target energy

4. For every subtask (or possibly every two subtasks) a brief description of how you will test that code before moving on

- main
    - After calling read_sample function, I will cout number of gene to make sure file is read properly
    - I will test the query functions by calling them in main and determining if outputs are correct
- read_sample
    - Open file
        - I will open the filename and check if it fails or not. I will have a cout error message if it fails and to test my code a success message if it properly opens
        - After allocating memory for gene, I couted the struct variables to make sure they were empty or 0
    - Get first line, total number of genes in file
        - Cout num in main
    - Allocate and memory for gene
        - Create init_gene function, cout first gene in main to determine if its empty
    - Populate memory for gene
        - Create populate_gene function
        - Cout the first gene in each line, do in main
    - Reset the file stream for another pass
        - To test resting the stream, I will get a line from the file
        - Line should be populated with the first line after the gene
- populate_gene
    - After using stringstream for the first pass and skip over the first line (which just has the number of genes), and setting each gene back in main we output the first gene to make sure we had properly set the first gene
- populate_mutation
    - To make sure we're getting the rest of the line, we'll cout our variables inside the while loop

5. Where appropriate, please cite the lecture, lab, or demo files that contain code examples relevant to a given task or subtask

I followed along the Monster Mash lecture when writing my code, and adapting my code accordingly. I also looked back at my code for phone_tree to replicate the query input.