ENGR 101 W20 C++ Exam: Code Writing (55 points)

Congratulations! You've been accepted to join the pilot team at gLyft, carrying passengers to their destinations all across the galaxy.

Recall the output format from Project 6 (Fig. 1, left). As you stop at each planet along your route to let off passengers, you also need to pay a fee to refuel your ship. The fee depends on the type of spaceport at each planet, which is indicated by the letter used to represent that location in the map (Fig. 1, right).

Sample Map Format (`map_public.txt`)

```
.....
.XX.S
.....
.....
EQ...
..A..
X....
..P..
.....
.....
.....
.....
T....
Start at 2 5
Go to Etheria and Eternia at 2 3
Go to Domeo And James Juliett at 2 2
Go to New Earth (Planet Bob) at 5 2
Go to The Library at 6 3
Go to The Discworld at 8 3
Go to Jupiter Two at 7 1
Go to Fhloston Paradise at 13 1
End at 5 1
```

Sample Fees Format (`fees_public.txt`)

```
X 792.4
T 55.55
Q 875.3
M 609.2
P 2341.
B 342.1
L 112.7
A 9349.
```

Figure 1. (left) Sample output from Projet 6. (Right) Sample input file containing the refueling fees for different spaceport types.

For example, in the map output shown in Fig. 1, there are spaceport types `X`, `Q`, `A`, `P`, and `T`. Note that several planets on the trip may have the same spaceport type -- in this case, three planets have spaceport type `X`. (Note that `S` and `E` do not represent spaceports -- those are the "Start" and "End" of the trip, respectively. Spaceport types will never be represented by `S` or `E`, for this reason.)

The fees for each spaceport type are contained in an input file that contains the letter representing that type and the associated fee amount.

You would like to compute the total amount you need to pay in fees for a particular trip.

Logistics
Your submission to autograder.io consists of 4 files.
Download starter versions of these files from the course Google Drive.

- **`fees.h`**
  Contains the definition of the `Spaceport` struct and prototypes for library functions in `fees.cpp`. Do not modify this file, except the definition of the `Spaceport` struct.
- **`fees.cpp`**
  Contains implementations of library functions that act as helpers for the driver.
- **`calcFees.cpp`**
  Contains the driver program (i.e. the `main()` function).
- **`honor_pledge.txt`** (worth 1 point)

Part A (2 points): Your program should use a `struct` to hold information about a particular spaceport type and its fee. The `Spaceport` struct has two member variables:

1. **`spaceportType`** - The letter used to represent the spaceport type.
   Example values: `'X', 'Y', 'A'` (Note that `'S'` and `'E'` are not allowed.)
2. **`fee`** - The amount that must be paid to refuel at that spaceport type.
   Example values: `100, 229.99`

Task**:** Fill in the `Spaceport` struct definition in the `fees.h` file.

Grading**:** The autograder will verify your `Spaceport` struct has the right members and types.

Part B (10 points): Complete the `readMap` helper function in the `fees.cpp` file, which reads map data from an input stream. The data format is the same as the output format from Project 6 (see example at right). The map portion of the input contains each planet's spaceport type in a grid. Start and end locations (`'S'` and `'E'`) and empty locations (`'.'`) should be ignored. The `readMap` function populates a vector with a list of the spaceport types found in the map, in the same order as they appear in the stream input. If there are duplicates (e.g. `'X'` in the example at the right), these duplicates will also go into the vector (i.e. you visited more than one spaceport of that type). The list portion of the input should be ignored (e.g. stop reading when you get to "`Start`").

```
void readMap(istream &in,
             vector<char> &stops);
```

Example: `map_public.txt`

```
.....
.XX.S
.....
.....
EQ...
..A..
X....
..P..
.....
.....
.....
.....
T....
Start at 2 5
Go to Etheria and Eternia at 2 3
Go to Domeo And James Juliett at 2 2
Go to New Earth (Planet Bob) at 5 2
Go to The Library at 6 3
Go to The Discworld at 8 3
Go to Jupiter Two at 7 1
Go to Fhloston Paradise at 13 1
End at 5 1
```

Task**:** Complete the implementation of the `readMap` function in the `fees.cpp` file.

Grading**:** The autograder will unit test your `readMap` function on several different sets of input.

Part C (10 points): Complete the `readFees` helper function in the `fees.cpp` file, which reads information about the fee for each kind of spaceport from an input stream into a vector of `Spaceport` structs. An example of the input format is shown at the right. Each line of the input contains a letter indicating a spaceport type and the fee to refuel at that type of spaceport, separated by a space. You may assume there are no duplicate entries in the input for the same spaceport type.

Example: `fees_public.txt`

```
X 792.4
T 55.55
Q 875.3
M 609.2
P 2341.
B 342.1
L 112.7
A 9349.
```

```
void readFees(istream &in,
  vector<Spaceport> &spaceports);
```

Task**:** Complete the implementation of the `readFees` function in the `fees.cpp` file.

Grading**:** The autograder will unit test your `readFees` function on several different sets of input.

Part D (10 points): Complete the `getFee` helper function in the `fees.cpp` file, which looks up the fee for a given spaceport type in a vector containing `Spaceport` structs (e.g. the kind of vector you might get from using `readFees()`). If the requested spaceport type is not represented in the vector, return an assumed default fee of 100.

```
double getFee(char spaceportType,
              const vector<Spaceport> &spaceports);
```

Task**:** Complete the implementation of the `getFee` function in the `fees.cpp` file.

Grading**:** The autograder will unit test your `getFee` function on several different sets of input.

Part E (22 points): Finally, write a driver program that computes the total amount you will need to pay in spaceport fees for a particular trip. The trip will be specific information coming from a map input file, and the fee amounts come from a fee input file (see examples of both formats above). The names of these files are specified by the user through `cin` when they run the program. The program outputs a report to `cout`, including the fees for each stop along the route and the total amount paid. We highly recommend you use the helper functions from earlier parts when writing your driver.

Task**:** Write a driver program in the `calcFees.cpp` file that behaves as described above.

Input Prompts: Your program should prompt the user to enter the names of the map and fees input files. Use exactly these prompts:

```
"Enter Map Filename: "
"Enter Fees Filename: "
```

Do **not** print a newline (i.e. `endl`) after the input prompts, since the user will already be pressing the `<Enter>` key when they submit their input. Do not include the quotation marks (`"`).

Error Checking: Your program must perform the following error checks:

1. Check that the map input file is opened successfully. If it is not, print:

   `"Error: <mapFilename> could not be opened."`

   to `cout` where `<mapFilename>` is the name of the specified map file. Do not include the quotation marks (`"`). Then, your program should `return 1;` from `main()` to indicate an error has occurred.

2. Check that the fees input file is opened successfully. If it is not, print:

   `"Error: <feesFilename> could not be opened."`

   to `cout` where `<feesFilename>` is the name of the specified map file. Do not include the quotation marks (`"`). Then, your program should `return 1;` from `main()` to indicate an error has occurred.

Perform the above checks in the specified order. Be sure to check the map input file can be opened before prompting for the next file. (Note that if both files happen to be missing, only the map file error message should be printed since that error check immediately returns from `main`. Note also that for the same reason, if the map file was missing, the prompt for inputting the fees filename would not be displayed.)

You do not need to check for any other errors. If the files open successfully, you can assume the input they contain is properly formatted as described above.

Output Format: Your program should output the spaceport types that are visited, their individual fees, and the total amount paid in fees for the trip. The example output shown at right corresponds to the desired output assuming the `map_public.txt` and `fees_public.txt` files are used. (The prompts and input from the user are also shown in the example.) Note that the order of spaceports in the output is the same as the order in which they are read by the `readMap` function, *not* the order they appear in the "Go to ..." portion of the map input.

Example: Public Test Output to `cout`

```
Enter Map Filename: map_public.txt
Enter Fees Filename: fees_public.txt
Spaceport Type X: 792.4
Spaceport Type X: 792.4
Spaceport Type Q: 875.3
Spaceport Type A: 9349
Spaceport Type X: 792.4
Spaceport Type P: 2341
Spaceport Type T: 55.55
Total: 14998
```

Grading**:** The autograder will run your program on several different sets of input and verify that it produces the correct output. It will also verify that your program produces the correct error messages and returns `1` from `main` if the specified input files cannot be opened.

Public Test Case: Sample `map_public.txt` and `fees_public.txt` files are provided for you to test your own code. This test is also present on the autograder.

To compile your code, use: **`g++ --std=c++11 calcFees.cpp fees.cpp -o calcFees`**

Run your code with: **`./calcFees`**