PokemonCatalog.cpp Description

A Pokedex is a catalog for pokemon where information about a pokemon can be stored.

- The name (or key) into Pokedex must be 1 word (case sensitive) name of pokemon
- The information (info) associated with each name can be any text

name	*Info*
kakuna	This is a pokemone that looks like a cocoon
pikachu	Everyone knows this pokemon
bulbasaur	it can survive for days just on sunlight
oddish	This grass pokemon looks like a radish

(a pokedex example shown above)

Protocol or Input Format

1. Each command will/must be entered on separate lines.
2. All commands are case sensitive.
3. Only the following commands will be supported (with exactly 1 blank space between command, name, and info)

Command	Description
PUT *name info*	If *name* is invalid report, "406 Not Acceptable" and do not add/update *info*. If name is *valid*, Add/replace *info* (0 or more characters) in Pokedex for pokemon named *name*. Print "201 Created" as result message.
GET *name*	Prints *info* for the pokemon with given *name* one 1 line and print "200 OK" on second line. If *name* is not found report error: "404 Not found".
FIND *sub*	Report *info* for all pokemon whose *name* contains the substring *sub*. One entry is printed per line. At the end of the list print "200 OK".
DELETE *name*	If *name* is found, then remove entry for pokemon with given *name* and print "200 OK" (to indicate operation was performed). If *name* is not found, then report error message "404 Not Found"
QUIT	Stops the program
	The above set of commands constitutes the base case for this Program.
SAVE	Save pokedex to the file "./pokedex.txt" (this is the exact path you must use). The file must contain 1 pokemon entry per line with *name* and *info* separated by exactly 1 blank space.
LOAD	Load pokedex (replacing all entries) from file "./pokedex.txt" (this is the exact path you must use). This is the file written by your SAVE operation.
Other methods, structure, etc.	If the user enters an invalid command report error message "400 Bad Request". Note that the error messages are loosely modeled after HTTP status code: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Functional Testing

Input and expected output files are supplied:

base_case_inputs.txt

base_case_outputs.txt

load_test_inputs.txt

load_test_outputs.txt

expected_pokedex.txt

pokemons.txt

save_test_inputs.txt

save_test_inputs.txt

Using diff commands to compare resulted output files with expected output files to see if the program's outcome is what you expected. If so, diff command will result nothing which indicates the resulted output is what we expected.

Base case test:

```
$ ./homework4 < base_case_inputs.txt > my_base_case_outputs.txt
$ diff my_base_case_outputs.txt base_case_outputs.txt
```

Save test:

This test expects your program to work for the base case and then tests for correct operation of the `SAVE` command. Note the 2 different checks here:

1. The first `diff` checks if the output from your program is consistent
2. The second `diff` checks if the data is saved in correct format

```
$ ./homework4 < save_test_inputs.txt > my_save_test_outputs.txt
$ diff my_save_test_outputs.txt save_test_outputs.txt
$ diff pokedex.txt expected_pokedex.txt
```

Load test:

This test expects your program to work for the base case and then tests for correct operation of the `LOAD` command. Note that the supplied `expected_pokedex.txt` is used as input:

```
$ cp expected_pokedex.txt pokedex.txt
$ ./homework4 < load_test_inputs.txt > my_load_test_outputs.txt
$ diff my_load_test_outputs.txt load_test_outputs.txt
$ diff pokedex.txt expected_pokedex.txt
```