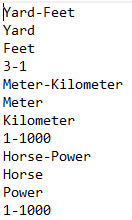
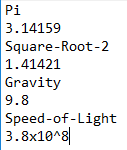
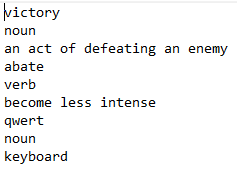
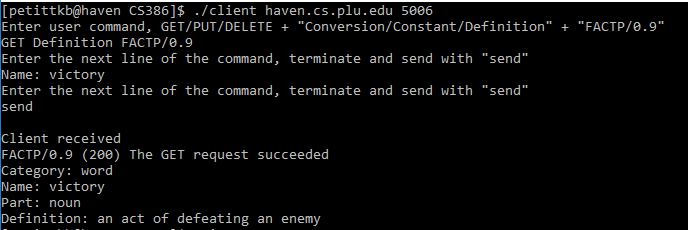
**Team Members:** Robin Naggi, Nick Sundvall, Keith Petitt

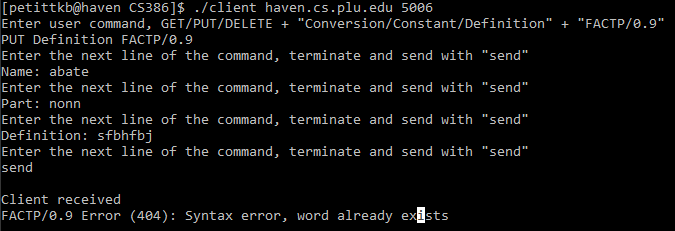
**Client**

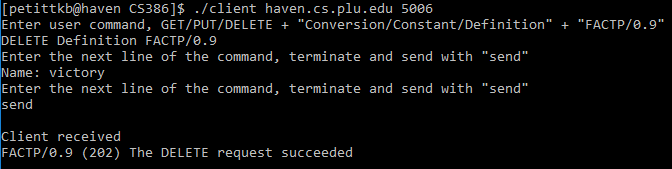
To start the client it takes two arguments after the ./client, first being the host IP and then the port number. If the input does not match what's expected the user will send a “wrong input” message and they will have to try again. After this first step the will be prompted to enter a method name with the category and “FACTP/0.9”. Then you’ll be prompted to enter the next line of commands first being the name. After the first command entry and beyond you can enter “send” at any moment to send your command to the server for evaluations. The error or success in your command will be determined by the server and will be sent as a message “FACTP/0.9” followed by “error code” or “success code” of the command. Following are the examples but not limited to all examples of the client sending and receiving messages corresponding to the various command inputs.

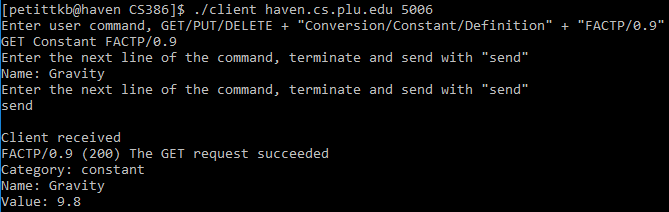
words.txt constant.txt conversion.txt

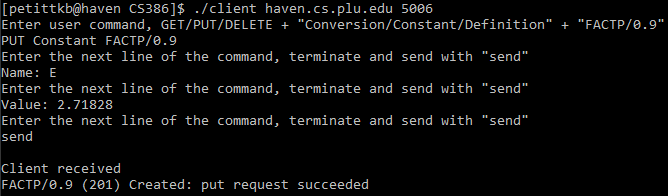
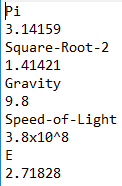


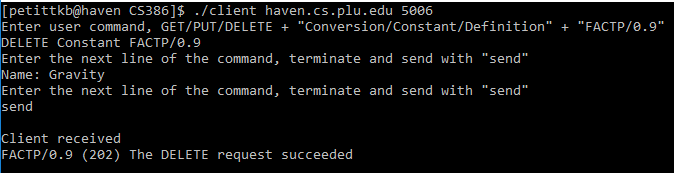
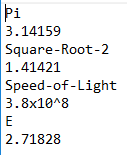












**Server**

Running the server doesn’t take any additional arguments, you just have to run it and then the server will be waiting listening for a connection. When the server receives the data that was input from the client after a connection has been made the server breaks up the char array using the strtok method. The delimiter used to break up the char array includes whitespace, newlines, tabs, and more. Every separate token from the input is put into an array. The server looks at the first token to determine what kind of request it has been set, either a “GET”, “PUT”, or “DELETE” and if it does not match any of those then it adds the proper error code to be sent back to the client. Then the server looks at the second token to determine if the category is a “Conversion”, “Constant”, or “Definition”.

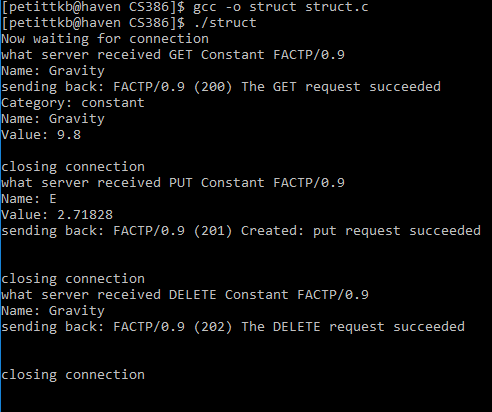
Based on these two pieces of information the server will call a method to either get the proper Factoid to return to the client, add a new Factoid, or delete a Factoid from the server. Also, before calling the method to do what the client requested, the server checks to ensure that the proper field names were included in the request, that the request was formatted properly, and that the values associated with the field names are not null. After completing the request from the client, the server sends back the appropriate code to let the client know if the request was successful, if the client did something wrong, or if an error occurred during the process. If the client performed a “GET” then the server will send back the Factoid requested at the end of the message.

The primary implementation for file reading and data storage comes in the form of structs. These structs each correspond to the three types of commands/categories in the FACTP/0.9 protocol. These are stored into arrays of structs corresponding to their data types; allWords (Definitions), allConstants (Constants), and allConversions (Conversions). Prior to initial connection with a client, the server first reads in the data from the text files into their respective struct arrays where values are stored accordingly. Each field in the structs points to a character array equal to its string of size[n]. One noteworthy aspect of these structs are their MAX\_INDEX values. These values function to emulate a dynamic array while allowing for a large static array to be declared at the prior to receiving the client. The MAX\_INDEX value will be modified throughout the server’s runtime. All functions on the server side other than the initial input processing utilize these structs.

The GET functions loops through each element in the appropriate struct and compares the desired string to the struct[index].name value. A flag-like value is assigned if a match is found, however, the method will continue to read through the entire struct despite a match being found. Afterwards, if the flag value was set, the values of the data type are appended to the return string sent back to the client.

The DELETE function works similarly to the GET function as it requires only the name of the item to be deleted. Again, the function loops through the struct until it reaches the end. If the value is found the server will move each value at each next index forward one, overwriting the current value. Then the struct utilizes its MAX\_INDEX value to simulate a fluctuating size struct by decreasing it by 1. Data isn’t necessarily deleted through this process, however, when writing the struct out to a file we loop through the struct up until it’s MAX\_INDEX value, essentially ignoring any extra inputs that were deleted previously.

The PUT function first reads through the entire struct exactly like the other two functions. If a matching value is found the method will return an error code stating that the value already exists within the struct. Otherwise, the function will simply increment the MAX\_INDEX by 1 and use the initial constructor method to set the value of the next index equal to the new struct created.



But wait, there’s more!

←------

Above is just a small example of how the server continues to run after the first client has sent it’s request and it has been completely processed.