Section B (the pseudocode):
The server psuedocode:

```
function file_exists(location):
                    OPEN file at location
                    SET fp variable to the result of opening file
                    IF the file didn't open properly
                        RETURN -1
                    ELSE
                        CLOSE file we opened
                        RETURN 0
end
Function free_char_pointer(char_pointer):
            IF the char_pointer  is not null:
                        Free char_pointer
                        Set char_pointer to point to NULL
end
function check_pokemon_type(data_line, ideal_type, separator):
                    ITERATE through data_line string using strsep function
                    SET pokemon_type to the type value inside the comma separated string, data_line using strsep function
                    IF pokemon_type and ideal_type match
                        RETURN 0
                    ELSE
                        RETURN -1
end
Function server_read_pokemon(*arg):
            SET passed_in  variable to represent the dynamic array structure that is passed into the function through arg
            SET boolean counter variable to 1 to represent that the fact that the read thread is running
            SET pokemon_send_string to represent the string that contains all the pokemon of a certain type that will be sent to the client
            SET curr_pokemon_index to the index of the pokemon type that we want to read inside pokemon_types_array
            SET saved to store the amount of pokemon that are saved in this function
            OPEN the file that the user specified the name earlier
            LOCK the mutex
            IF thread_is_paused attribute in passed_in is set to 0:
              Pause the thread until a signal is given to unpause
            WHILE the file has not been completely read
                        IF num_lines > 0 THEN
                                    SET type_check_string to store a deep copy of the line being read
                                    ALLOCATE memory to the type_check_string
                                    COPY characters from line variable to type_check_string
                                    IF pokemon type at curr_pokemon_index match and type of the pokemon at the current line match using check_pokemon_type  function:
                                                Null-terminate the line string
                                                IF no pokemon have been saved so far:
                                                            ALLOCATE memory to pokemon_send_string
                                                            COPY line to pokemon_send_string
                                                ELSE:
                                                            ALLOCATE more memory to pokemon_send_string
                                                            CONCATENATE line to pokemon_send_string
                                                CONCATENATE the '|' character to pokemon_send_string
                                                INCREMENT saved variable by 1

                                    FREE type_check_string
            REALLOCATE more memory to pokemon_send_string to null-terminate the string
            SET pokemon_send_string_size to hold the amount of bytes pokemon_send_string takes up as a string
            SET number_of_pokemon_send_string to hold the amount of pokemon saved as a string
            CONVERT the length of pokemon_send_string to a string and store inside pokemon_send_string_size
            CONVERT saved variable to string and store inside number_of_pokemon_send_string
            IF thread_is_paused attribute in passed_in is set to 0:
              Pause the thread until a signal is given to unpause
            SEND pokemon_send_string_size, pokemon_send_stirng and number_of_pokemon_send_string to the client
            UNLOCK the mutex
            SET boolean counter variable to 0 to represent that the fact that the read thread is done
end
Function main():
            SET file_name to store the name of the file to be read from
            SET server_read_thread to be the thread to do reading operations
            OBTAIN name of the file to be read and put into file_name variable
            SET read_struct to be the structure that contains all the necessary variables to read and send data to the client
            ALLOCATE memory to the pokemon_types_array field of the read_struct structure
            CREATE the server socket
            BIND the server socket
            LISTEN for a client to connect
            INITIALIZE mutex and cond variable
            LOOP until stop signal is given
                        GET message from client
                        IF the message was pause:
                                    LOCK the mutex and set thread_is_paused field in readStruct to 0
                        If the message was unpause:
                                    UNLOCK the mutex, set thread_is_paused field in readStruct to -1, and send a signal to thread to unpause
                        IF the message was stop:
                                    EXIT the loop
                        ELSE:
                                    ALLOCATE memory inside pokemon_types_array field to store the new message
                                    COPY message into pokemon_types_array field
                        IF there is a new pokemon type that is stored inside pokemon_types_array_field
                                    CREATE a thread to read the file
            FREE all allocated memory inside the readStruct structure and file_name pointer
            CLOSE the server socket
end
```

The client pseudocode:

```
Function free_char_pointer(char_pointer):
            IF the char_pointer  is not null:
                        Free char_pointer
                        Set char_pointer to point to NULL
end
```

```
function check_valid_pokemon_type(input_type):
        IF input_type matches one of the pokemon types as formatted with the first letter uppercase and all others lower case
                Return 0
        Return -1
End
Function write_pokemon(arg):
              SET temporary variable to represent the dynamic array structure that is passed into the function through arg
              SET thread_is_paused inside temporary variable to 0
              LOCK mutex
              SEND message to server to pause
              OPEN a file using fopen() using name_of_file attribute of dynamic array structure
              LOOP through every pokemon saved successfully inside temporary
                CONVERT pokemon to a string using pokemon_to_line function
                WRITE string to file using fprintf function
              CLOSE file
              IF number_of_saved_files  attribute of temporary = 0
                ALLOCATE memory to store name of file that we just opened inside all_file_names array inside ExpandedThreadType struct inside the  temporary variable
                COPY file name of the file we just wrote to to the memory allocated
              ELSE
                IF name of file we wrote to has not already been written to using check_for_previous_file function
                        REALLOCATE more memory to all_file_names array inside ExpandedThreadType struct inside the  temporary variable
                        ALLOCATE memory to store name of file that we just opened inside all_file_names array inside ExpandedThreadType struct inside the  temporary variable
                        COPY file name of the file we just wrote to to the memory allocated
              SET thread_is_paused inside temporary variable to -1
              SEND signal to mutex in read_pokemon that the condition has changed
              SEND message to server to unpause
              UNLOCK mutex
end
function print_final_information(temporary):
        PRINT number of successful queries
        FOR each file created during this session
                PRINT file name
end
function line_to_pokemon(line, new_pokemon, separator):
       SET number, name, first_type, second_type, total_stats, health_points, attack, defense, speical_attack, special_defense, speed, generation and legendary to store their respective
information from comma separated info from line using strsep
       ALLOCATE memory to new_pokemon variable
       ALLOCATE memory to name, first_type and second_type attributes of new_pokemon

       SET name, first_type and second_type attributes of new_pokemon to contain the null terminating character using memset
       COPY memory addresses containing the characters in name, first_type, and second_type to their respective attributes inside new_pokemon
       FOR each of the remaining variables containing information about pokemon except the legendary variable:
         CONVERT those variables to int and store inside the corresponding attribute inside new_pokemon using strtol
       IF legendary variable is equal to False
         Set new_pokemon’s legendary attribute to store the letter n
       ELSE
         Set new_pokemon’s legendary attribute to store the letter y
end
Function add_pokemon(pokemon, pokemon_dynamic_array):
        SET new_pokemon_array to be a new dynamic array
        If information passed in is invalid
                Exit the function and return an error
        Allocate memory for the new_pokemon_array

        FOR pokemon in the old dynamic array
                Store all those pokemon in the new dynamic array

        Insert the pokemon at the right-most-index in new dynamic array

        FOR transactions in the old dynamic array
                Store all those transactions in the new dynamic array as long as they’re to the right of position_to_insert
end
Function pokemon_to_line(line_to_write, pokemon_to_write, separator):
              For each attribute inside pokemon_to_write except legendary attribute:
                Copy attribute into a string using sprintf
                CONCATENATE string to line_to_write string
                CONCATENATE separator character to line_to_write string
              IF legendary attribute = ‘y’
                CONCATENATE “True” string to line_to_write
              ELSE
                CONCATENATE “False” string to line_to_write
end
Function check_for_previous_file(temporary, check_file)
        For every file name stored inside the all_file_names attribute inside temporary
                IF the file name inside temporary = check_file
                        RETURN 0
        Return -1
end
Function read_pokemon(arg):
        SET dynamic_array to represent the dynamic array structure that is passed into the function through arg
        SET bytesReceived to represent the bytes received from the recv function
        SET buffer to store the first message sent from the server to the client
        LOCK the mutex
        IF thread_is_paused attribute inside dynamic_array is set to 0:
                WAIT until a signal is given to unpause the thread
        SET thread_is_running attribute inside dynamic_array to 0
        SEND the pokemon type inside all_types_being_read array inside dynamic_array structure at curr_type_being_read index to the server
        RECEIVE the amount of bytes the next message from the server will be and store the amount of bytes inside buffer variable
        CONVERT buffer to an integer and store conversion inside bytes_from_message variable
        SET pokemon_message to store the message containing all the pokemon from the server
        ALLOCATE memory with the bytes_from_message variable and malloc function to have enough memory to store the next message from server
        RECEIVE the string containing the pokemon data from the server and store message inside pokemon_message
        RECEIVE the number of pokemon in the pokemon_message and store the number_of_pokemon inside number_of_pokemon_message variable
        CONVERT number_of_pokemon_message variable to an integer and store inside number_of_pokemon variable
        LOOP through each pokemon inside pokemon_message string:
                CONVERT line containing pokemon information inside pokemon_message to a PokemonType with line_to_pokemon function
                STORE the pokemon inside dynamic_array
        INCREMENT number_of_successful_queries attribute of temporary variable by 1
        INCREMENT number_of_pokemon_sucesfully_Saved attribute of temporary by value in number_of_pokemon variable
        INCREMENT curr_type_being_read attribute inside dynamic_array by 1
        UNLOCk the mutex
        FREE allocated memory in the function
        SET thread_is_running attribute inside dynamic_array to -1
End
Function main():
        SET clientSocket to store the socket the client is communciating with
        SET gamer_choice, type_choice and user_file_name_choice variables to contain user input from the user
        SET read_thread and save_thread as threads
        SET dynamic_array to an instance of the DynamicArrayType struct
        ALLOCATE memory to dynamic_array, extra_pokemon_data struct, all_files_names array and all_types-being_read array
        CREATE the socket with the TCP protocol and set to clientSocket variable
        SETUP address of the client
        CONNECT to server
        STORE clientSocket variable inside dynamic_array
        LOOP until user exits the program:
                LOOP until user exits program
                GET user's choice and put into gamer_choice variable
                IF user selects choice a THEN
                        OBTAIN type of pokemon to be read and put into type_choice variable
                        CHECK if the type of pokemon is valid using check_valid_pokemon_type function
                        IF there is a pokemon already stored inside all_types_being_read THEN
                                REALLOCATE more memory for all_types_being_read
                        ALLOCATE memory inside all_types_being_read to store the new pokemon type
                        COPY the pokemon type to all_types_being_read array
                        Increase all_types_being_read_size attribute of dynamic_array by 1
                        JOIN the read_thread if it has already been run
                        SET read_thread to read pokemon from using read_pokemon function and pass in the dynamic_array
                ELSE IF user selects choice b THEN:
                        IF no info has been read THEN
                                GO back to menu
                        OBTAIN name of file to save data and put into user_file_name_choice variable
                        LOOP until the user enters a valid name:
                        IF user_file_name_choice is invalid and cannot create a file that can be opened:
                                FREE user_file_name_choice variable and prompt user to enter a new name
                        ELSE:
                                EXIT loop
                        STORE file name inside name_of_saved_file attribute of dynamic_array
                        JOIN the save_thread if it has already been run
                        SET save_thread to write pokemon to file using write_pokemon function and pass in the dynamic_array
                ELSE IF user selects choice c THEN:
                        IF the query thread is running
                                CANCEL the query thread from continuing to operate
                        JOIN save thread
                        PRINT the required information about saving and queries using the print_final_information function
                        FREE the gamer_choice, type_choice and user_file_name_choice pointers if they contain information
                        FREE all the file names stored inside all_file_names and then free the all_file_names pointer itself, if allocated
                        FREE all the pokemon types stored inside all_types_being_read and then free all_types_being_read pointer itself, if allcoated
                        FREE memory allocated to each pokemon's name, first_type, second_type, and the pokemon itself, if allocated
                        DESTROY mutex and cond variables
                        FREE memory allocated to the dynamic array that contained the pokemon, if allocated
                        FREE the extra_pokemon_data struct if allocated
                        FREE the dynamic array struct itself
                        SEND stop message to server
                        CLOSE the client socket
                        EXIT program
                ELSE:
                        PRINT message saying that they did not select any of the available options
                        GO back into the loop
end
```

Section C:
In Assignment 4, I've used 2 header files (client.h and server.h) and 2 C files (client.c and server.c). I have 2 header files and 2 C files to separate the functionality of both the client and the server. Since possibly there would be different people working on the client and the server, I've separated the functionality of the client and the server into two different files for encapsulation (the people working on the client only have to focus on the functionality of the client) and for security (the people working on one of the c files will not have to deal with someone else modifying their code). Not only that, but the server and client require different struct's and functions so having them separated creates easier to read code.

Section D:
I've implemented the Non-Functional Requirements of performance and maintainability.
The performance NFR is accomplished via multi-threading. From the client side, I have two threads (one dealing with reading operations/operations dealing with the server) and one dealing with saving operations to the disk. From the server side, I have one other thread (other than the main thread) to deal with reading the pokemon.csv. Having multiple threads that do the heavy-lifting operations of the program, it allows the main thread to be unencumbered and remain responsive despite the reading and saving operations going on. Also, the reading threads in both the server and client also deal with the inter-communication between the server and client, which makes the main threads from both programs more responsive.
The maintainability NFR is accomplished by separating the server (server.c and server.h) and client (client.c and client.h) functionality into their respective header files and C files. A Makefile is also provided to build the program, which makes it easy to compile and link files. The MakeFile also allows you to cleanup .o and executable files with the make clean function.

Section E:
The stdio, stdlib, string and pthread libraries are copied over and used from Assignment 3. To summarize, stdio is used for getting information from and to the terminal, to and from the user, stdlib is used for string conversion functions like atol and macros like EXIT_FAILIURE. String library is used for string functions to concatenate and copy like strcpy and strcat. Finally, the pthread library is used to create mutexes and threads to implement multithreading in Assignment 4. The new libraries which are included are: unistd.h, socket.h, in.h and inet.h. The unistd.h is used in the server for the sleep() function which causes the read thread to be suspended for 0.25 seconds. This allows data to flow between the client and server at a reasonable rate and for data to not be overwritten/corrupted when sent between the client and server. This is used in Use Case 3. The socket.h is very important since it allows sockets to be created and for information to pass between the server and the client via recv() and send(). It allows the server and client to be connected in Use Case 1 and Use Case 2, allows the client and server to send pokemon data back and forth in Use Case 3 and for the server and client to close properly in Use Case 5. The in.h and inet.h is crucial to setup the sockets and to implement the TCP protocol. In Use Case 1 and Use Case 2, it's used to open port 6000 and to connect through the IP, 127.0.0.1.

Like Assignment 3, I've made some assumptions about the implementation of the requirements outlines (and not outlined) inside the Assignment 4 specifications:

- When the client properly closes the client program, the client will send a stop request to the server, allowing the server to properly close.
- To make the program easier to run (and without the sudo command), I've configured the client and server to both listen on port 6000.
- To test the program, the tester should use two separate terminals.

To add more information about Use Case 3 (since it's the most complex Use Case in the specification), I've used the strcat function to concatenate all the pokemon that match the Gamer's type input into one giant string and to sent that string to the user in one go using the send() and recv() function respectively. I chose this method to send information between the PPS and PQC since it reduces the amount of calls of the send() and recv() function.