COMP 2401 - Assignment 2 - Design Document

Grant Li - 1012558982

Pseudo Code:

Main.c:

```
link to header file
func main(){
        create shared data struct to be used by all threads

        get user to enter the file they want to query from
        save it to data.readFileName

        initiate mutex

        start menu(data)
        get total number of queries from menu()

        destroy mutex

        print(ending messages)

}

func menu(data){

        open file
        loop while user input file is not valid{
                if not valid, ask for a valid one or to quit
        }
        close file

        create variables for threading and queries numbers

        run loop while user doesn't want to exit{
                print(menu options)
                get input

                switch statement for input{
                        option 1 for searching for a Pokemon type{
                                get user input for Pokemon type into data.PokeType
                                create thread and put it into a thread array
                                use the new thread to start getPokemon(data)
                                break
                        }
                        option 2 for saving queried Pokemon into a file{
                                get user input for save file name into data.writeFileName
                                create thread and put it into a thread array
                                use the new thread to start savePokemon(data)
                        }
                        option 3 for exiting program{
                                loop to detach all the threads
                                free all allocated memory from data structs
                                return number of queries
                        }
                }
        }
}
```

FileIO.c:

```
link to header file
func getPokemon(data){
        create instance of shared data struct from data parameter

        create temp pokemon variable to represent each line of the input
        open the user query file from data.readFileName

        scan the first line to avoid csv column names
        while not at the end of the file{

                if data.flag is 1 {close thread}

                scan a line's data into the temp Pokemon variable

                if the temp Pokemon has the desired type{
                        unlock mutex using data.mutex

                        resize pokemon query array from data.pokeDeck
                        copy temporary pokemon info into new element of data.pokdeck

                        lock mutex
                }

        }
        close file
}

func savePokemon(data){
        create instance of shared data struct from data parameter

        open file from data.writeFileName
        loop through all pokemon in array{

                if data.flag is 1 {close thread}
                write down Pokemon's attributes into the file using a format
        }
        close file
}
```

Coding choices and why:

I created 2 .c files and a header file to separate the functions I would need to use multiple times and the functions that only need to be called once every time the program is launched. This is because the functions that are called multiple times are done through multithreading, and I separated the ones that needed to be multithreaded from the ones that didn't in order to organize files to be edited more efficiently. A single header file links the 2 .c files. The header also contains struct templates to hold Pokemon data from the user and file queries, as well as the data shared between the structs.

To account for the NFR - performance, I multithreaded the program so that after the user inputs a choice in the menu, they are immediately returned to the menu while the searching or saving is done in the background. This would make the program faster for the user because they wouldn't have to wait for the program to finish searching or saving to be brought back to the menu for their next choice.

In the program, <pthread.h> was used in order to multithread the program for performance. <string.h> was used to compare the file query to the user query in order to find a Pokemon with the desired type. <stdio.h> was used to interact with the user and to access files. <stdlib.h> was used to dynamically create and resize variables to fit with the user and file queries, specifically dynamic input from both.

In terms of data storage locations, the lists of Pokemon and threads were stored in the heap because the list of queried Pokemon could grow whenever the user wants, subsequently, more threads would have to spawn from this process and the saving process. Thus the list to keep track of threads was dynamic as well as the list of Pokemon. Data containing the name of the file queried and written into were stored in the heap, because of unknown user input sizes. This is the same case with the type of Pokemon specified by the user.

The shared data would be protected from corruption by the use of mutexes. When a thread wants to edit the data that is shared by all threads, it will lock the mutex, make its edits, then unlock them to ensure only one thread can change the shared data at a time.