DESIGN DOCUMENT – ASSIGNMENT 3

Name: Esther Osammor

Program Flow:

1. Start Program
2. Define PokemonLine Structure
3. Set successQuery = 0;
4. Print instructions for user to enter file name
5. Input InputFileName
6. Open file for reading
7. While file == NULL:
   7.1. Print out instructions for user to input file name again due to error or press 3 to exit 7.2. Input inputFileName
   7.3. If inputFileName = 3:
      7.3.1. End Program
   7.4. Open file for reading
8. While true:
   8.1. Create thread, menu
   8.2. Pause main program and wait for menu to finish executing
      8.4.1. Print out menu for user to choose from
      8.4.2. Input userSelection 8.3. Continue main program 8.4. If userSelection == a:
      8.4.1. Print out instructions for user to input the pokemonType they want to search for
      8.4.2. Input userSearch
      8.4.3. Set foundFlag = 0
      8.4.4. For each line in file:
         8.4.4.1. If userSearch matches pokemonType:
            8.4.4.1.1. Create/resize dynamic array of PokemonLines, pokemonLineArr
            8.4.4.1.2. Store line in pokemonLineArr
            8.4.4.1.3. foundFlag = 1
      8.4.5. If found == 1:
         8.4.5.1. successQuery += 1;
      8.4.6. go to next iteration of loop (continue)
   8.5. If userSelection == b:
      8.5.1. Print out instructions for user to input the name of the file they want to store found results in
      8.5.2. Input saveFileName
      8.5.3. Open saveFile for writing:
      8.5.4. If saveFile == NULL:
         8.5.4.1. Print instructions for user to try again due to error
         8.5.4.2. Input saveFileName
         8.5.4.3. Open saveFile for writing
      8.5.5. For all pokemonLines in pokemonLineArr:
         8.5.5.1. Write pokemonLine to saveFile

8.5.6. Create/Resize dynamic array of fileNameArr
8.5.7. Store saveFileName to fileNameArr
8.5.8. Close saveFile
8.5.9. Continue

8.6. If userSelection == c:
8.6.1. Print successQuery
8.6.2. For each saveFileName in fileNameArr:
8.6.2.1. Print saveFileName
8.6.2.2. Free memory for saveFileName
8.6.3. Deallocate memory for fileNameArr
8.6.4. Deallocate memory for userInputFile
8.6.5. Deallocate memory for userSelection
8.6.6. Deallocate memory for pokemonLineArr
8.6.7. Close file
8.6.8. End Program

How to Run Program:

Open directory in terminal and input 'gcc -o main main.c pfile.c -lpthread' to the console. Once compiled, input './main' to the console to run the program.

QUESTIONS:

1. How many C and header files will the program have and why?
   The program will contain two c files and one header file. One of the c files will be main.c(contains the main function) and the other will be pfile.c (pokemon file) which will contain all the implementations of functions defined in the header file. It will have these number of files to split up the code, enhancing both clarity and problem solving (easier to debug if code if split up into parts).

2. How does the program implement the functional and non-functional requirements? The program implements the functional requirements quite well. The user can input a file that contains pokemon csv data. It prompts the user to try again if the file in not found, the user is able to search through the file for a specific pokemon attribute and when it is found, all lines in the file containing that attribute is stored in memory, and later a file of the user's choice upon their request to do so. The program also makes use of a thread to display the menu right away after the user chooses an option from it.
   The non-functional requirement on performance is also implemented using a thread. The pausing of the main program until the thread that deals with the menu and its input is finished using pthread_join ensures that the program is always responsive to user input

3. What C language features and libraries should be used to implement the requirements and why?

The stdio.h and stdlib.h libraries should be used in order to use basic functions such as printf(), scanf(), exit(), etc. which are necessary for any c program to run. The string.h library should also be used in order to work with strings. An example of when this was used in the program was when the program had to save a line from the file into a pokemonLine structure. The strcpy() function from string.h had to be used to set the string attribute of the structure to equal a line from the file. The pthread.h library should also be used in order to allow threads to run in the program.

4. Which data will be stored in memory? How will these data be protected from corruption? Data such as the pokemonLine structure will have to be stored in memory, other data stored in memory include: character arrays for strings and lines, char variables to store user selection, arrays created to store structures and filenames, integers used to track how many elements in the arrays are created. These data can be protected from corruption by using mutexes and semaphores in the event of using threads that run concurrently and modify the same data.