Terminal Tinder Summary

Files:

heart.py --> uses the photos from heart_gif folder
sad.py --> uses the photos from the crying_gif folder

questionnaire.h
questionnaire.c

make_file.c (currently part of questionnaire.c may potentially make own file)
make_file.h (currently part of questionnaire.h)

make_match.c --> uses the text files from the database folder
make_match.h

Makefile

Steps:

- ◯ ***STEP 1*** - Make a heart.py file that will pop up a window displaying multiple heart images forming a gif, then finishing off with a final heart photo displaying the final match all while the song "Love Shack" is playing. Then this file will also email person1 and person2 the other's dating portfolio (`firstname_lastname.txt`)
- ◯ ***STEP 2*** - Make a sad.py file that will pop up a window displaying multiple crying images forming a gif, then finishing off with a final image indicating no match was made all while playing "I am so lonly".
- ◯ ***STEP 3*** - Make an questionnaire that prompts the current user to entire their information about themselves, and information about what they want in their match.
- ◯ ***STEP 4*** - Then save this current user's responses as their own file in the format of `firstname_lastname.txt`
- ◯ ***STEP 5*** - Also have their first and last name appended to the total_database.txt file
- ◯ ***STEP 6*** - Make the match from reading total_database.txt to get the name of the persons file you are wanting to read to attempt a match for the current user.

  ---------- *MORE INFO ON THIS* ----------
  - Note: It is important that you have your path set to the pwd for the user so that if this code is ever clones from gitub, it still works on anyone's machine. An example of this line in python is:

```
base_path = os.path.abspath(os.path.dirname(__file__))
base_path_to_database_folder = os.path.join(base_path,
database)
```

but I do not know the C code line off the top of my head. Then you will need to do a system join for the root_path to the `total_database.txt` or the `firstname_lastname.txt` of each user you look at. The reason why this is important is because if you hard code anything (even a simple / showing you are in a folder) then it will not work on other machines because each operating system is different. I am sure there is a C library you can use for this, but an example of this join in python is:

```
person1_file = os.path.join(base_path_to_database_folder,
(firstname1 + "_" + lastname1 + ".txt"))

person2_file = os.path.join(base_path_to_database_folder,
(firstname2 + "_" + lastname2 + ".txt"))

total_database_filename =
os.path.join(base_path_to_database_folder,
"total_database.txt"))
```

- You could use a data structure to read in the file `total_database.txt` line by line (keep in mind to NOT check the last like because the last line of `total_database.txt` is the current user's first and last name).
- Options for the data structures could be a Queue so whoever was there first will be more likely to match with the current user. Or, to be honest, I don't even think we need to pull it into a data structure at all because we could go about it in the following way:
  - Open `total_database.txt` file
  - Make sure to start at the top of the file (this way the oldest members already are being searched through first without needing a Queue format, and each time there is a new user entry, it would start from the top automatically -- `fseek(originalFile, 0, SEEK_SET);`)
  - Then parse the first line of the file by spaces and create the variables for who you are looking at
    - `Read line 1`
    - `firstname = line[0]`
    - `lastname = line[1]`
    - `filename = firstname + "_" + lastname + ".txt"`

    *** and you know you can do this because every file is going to be names the exact same way
  - Then you can keep `total_database.txt` file open but then open `firstame1_lastname1.txt` and compare the answers to the current user's file since we know exactly what lines will have what answers and all you need is to compare the first character of those lines since we had them enter a number

indicating their selection and I did error checking in questionnaire.c to make sure they are all entered as integers.
    - See the current example file `alyssa_kelley.txt`
    - Line numbers you need to look at (assuming the first line is referred to as line #1)
        - 17 (age -> needs to be compared to the other persons line 8)
        - 20 (gender -> needs to be compared with the other persons line 11)
        - 23
        - 26
        - 29
        - 32
        - 35
        - 38
        - 41
        - 44
- Then you can make a variable like `number_of_simularities` and use this as a counter if their answers match, and if it exceeds like 5 or something then they get matched with that person, and the matching process is over.
- If a match is found, then you can delete that line, and the very last line (aka the current user you are trying to find a match for) from `total_database.txt`. We can keep the two individual files for person1 and person2 but just delete them from the `total_database.txt`.

- ◯ ***STEP 7*** - Connect with the python scripts *(TWO OPTIONS ON HOW TO IMPLEMENT THIS)*
    - ◯ ***OPTION 1*** - We would then need this c file to SAVE the names of who gets matched with who for the heart.py file to work and email each other. This is the information that needs to get saved as environment variables which needs to be done in `make_match.c` (and i put the variabel names below each)

        - name of current user
            - PERSON1
        - name of person they matched with
            - PERSON2
        - email address of current user
            - PERSON1_EMAIL
        - email address of person they matched with
            - PERSON2_EMAIL
        - file name of current user
            - PERSON1_FILENAME

        - file name of person they matched with
            - PERSON2_FILENAME

- Here is a link with how to do this in C:
- ***https://stackoverflow.com/questions/3416638/set-environment-variables-in-c***

◯ ***OPTION 2*** - You can just invoke your make_match.c code to run the heart.py or the sad.py based on if they get a match or not, then you do not need to set any environment varibles, you would just pass them in as arguments

- Example:

```
    python3 heart.py PERSON1 PERSON2 PERSON1_EMAIL PERSON2_EMAIL
PERSON1_FILENAME
            PERSON1_FILENAME
```

- Here is a link with how to do this in C:

https://stackoverflow.com/questions/12142174/run-a-python-script-with-arguments

- Whatever works best for your `make_match.c` file works for me, this might be a better option because you can already use your saved variables from the match for the python arguments and do a simple if / else statement to indicate which python script to run

```
    if match == True:
        python3 heart.py <insert all the arguments>
    else:
        python3 sad.py
```

◯ ***STEP 8*** - Finalize the Makefile.

- The Makefile will need to do the following:
    - First run the command to display the logo
    - Compile and run the questionnaire.c
    - Compile and run the make_match.c
        - if make_match.c does not call the python script then call the correct python script.