CIS 212: Project #3C
Assigned: Tuesday November 13th, 2018
Due Tuesday, November 20th, 2018
(which means submitted by 6am on November 21st, 2018)
Worth 4% of your grade

Please read this entire prompt!

As per usual, I encourage you to tackle this project in small pieces.. The worst approach to programming is to write a ton of code, compile it, and then run and watch it fail … it is so hard to debug. Instead, break the problem up into sub-problems, compile frequently, and test frequently.

The final goal of this project is to match up heart donors with heart recipients. This will be done with the queue data structure, and you must use queues. In the real world, I imagine there are a lot of restrictions about matching up donors and recipients. For this toy problem, we will assume that any woman can donate a heart to any other woman, and any man can donate a heart to any other man.

The file "3C_input" contains a sequence of events. Note that I expect the filename to be an input to your program; you should look for argv[1] and not hardcode "3C_input" as the filename. (check_3C ensures this through some trickery – take a look.) Also, please keep in mind that if you are running gdb with a command line argument, you need to do something special. See appendix below.

The very first line is "R:F:Madie Koening." This means the first event is that (person) Madie Koening, a female (F), needs to receive a heart (R).

The second line is "H:Sacre Couer." This means that the second event is that the hospital (H) named "Sacre Couer" is available to do a surgery.

The third line is "D:F:Susie Brown." This means that the third event is that (person) Susie Brown, a female (F), can donate a heart (D).

This is enough to do a transplant. We have a female donor, a female recipient, and a hospital. So your program should issue a match.

Your program would then go through the rest of the events in the input file. For each new event, your program should make a match right then, if it can.

Note there is an ambiguous case: assume we have a female donor and a female recipient, and also a male donor and a male recipient, but no hospital. Then assume we get a hospital event. The question is whether to do the surgery for the female or male. In this case, the rule is "ladies first."

Like the real world, the sooner you ask for a heart, the sooner you will get one. So if person A appears in the list before person B (and if A and B are the same gender), then A will receive a heart before B (or donate their heart before B).

Obviously, queues are a great data structure for this, as they are a mechanism for placing people in lines. Also, again, you must use queues.

Hint #1: you will need five queues to make this work. If you look in the starter code, you can see the names of my five queues.

Hint #2: you should break this up into three steps.

Step #1: implement a queue data structure, and methods to work on that queue. There is commented out code in the main function of the starter code (labeled Step #1) for testing your queue code. (Note: you must implement this yourself. I don't want anyone using implementations from other libraries.)

Step #2: read the events from the file. There are slides in Tuesday's lecture on fgets. That is the easiest way to do it. That said, keep in mind that you need to be allocating memory for each new line. In my solution, I wrote a function called "NewString" that did this allocation. It also removes the newline that fgets leaves in. I put NewString in my starter code, and you are welcome to use it. Note that the memory allocated by NewString will be leaked, and that is OK.

Step #3: implement the matching described above. You can make the following assumptions in parsing:

- The first character of a line can only be R, H, or D.
- The second character is always a colon
- If the event is a hospital, then the third character is the beginning of the hospital name.
- If the event is a recipient or donor, then the third character is always M or F, the fourth character is always a colon, and the fifth character is the beginning of the person's name.

== Success ==

Before you submit, make sure to test your code using the provided grader program script (check_3C) on the Virtual Box.

Upload your new file, proj3C.c

== Running gdb with a command line argument ==

Remember! … this will not have the intended effect:

```
gdb a.out 3C_input
```

In this case, gdb will assume 3C_input is an argument to gdb. It will look at the argument, decide it is worthless, and discard it.

Instead, you want to do this:

```
(shell prompt:) gdb a.out
(gdb prompt:) run 3C_input
```

This will successfully pass the argument to your program as you run.

Otherwise, you will likely have a NULL argument for argv[1], and so your fopen will fail, and then you will end up debugging why fopen fails with a NULL filename … which is not the same problem you have outside gdb.