Assignment — C Messages

CS 265 Advanced Programming Techniques

Overview

Write a C ***program*** to verify messages.

Your task is to process a stream of messages line by line (either over **`stdin`** or read from a file, see Input section below). If the entire message matches any of the following patterns, you should print the message, followed by a space, followed by the text `OK`. If the text on the line matches none of the following patterns, you should print the message, followed by a space, followed by the text `FAIL`.

To Do

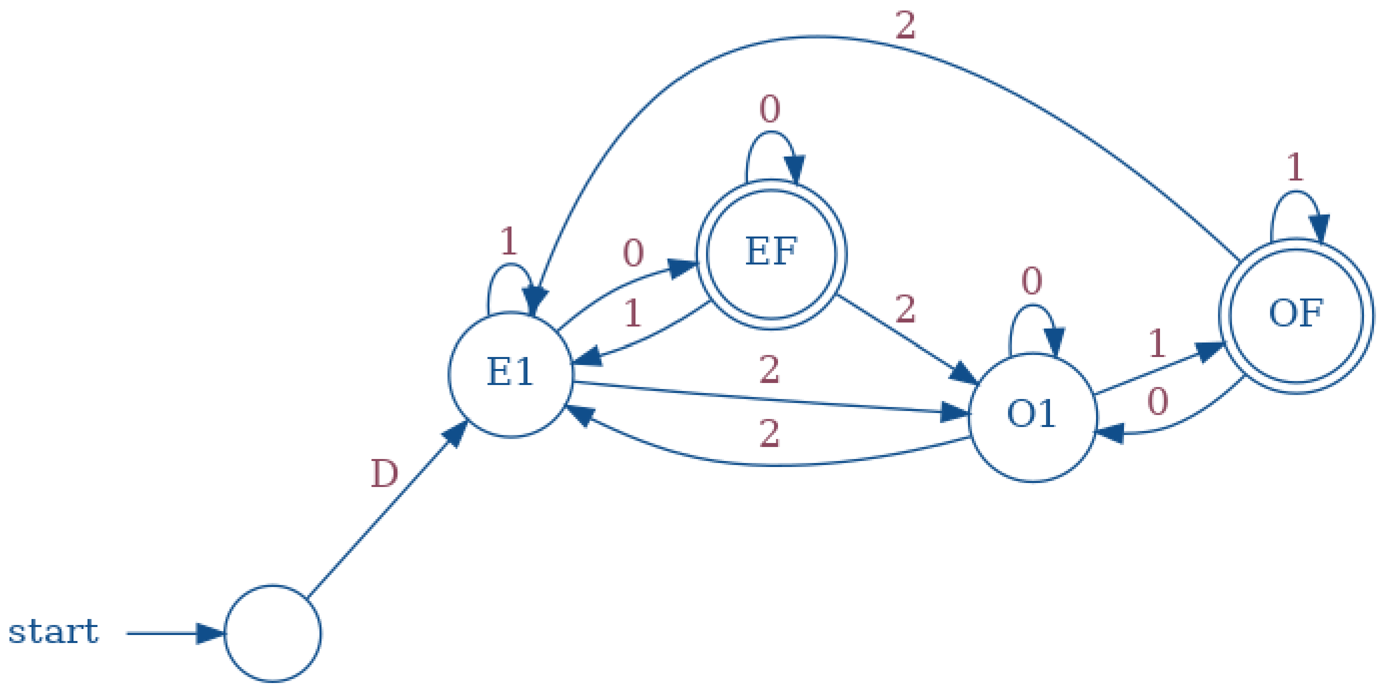
We have developed a protocol for reading messages from a device. There are different length messages. Each message has rules, described below. Messages are separated by a newline. Your job is to verify that the stream is valid, that it contains only valid messages.

Each message is followed by a description, some examples, and a *Deterministic Finite State Automaton* (*DFA*) which recognises the message. Before input is read, you are in state 1. We transition states on each character read. If, at the end of input, we are in an *accepting state* (double circle), then the message is valid. If there is no transition from a state on the current input, the string fails.

Dee:

Starts with an **`D`** followed by an arbitrary string of digits {**`0 1 2`**}. If the number of **`2`**s is even, the string ends with a **`0`** . Otherwise, it ends with a **`1`** E.g.:

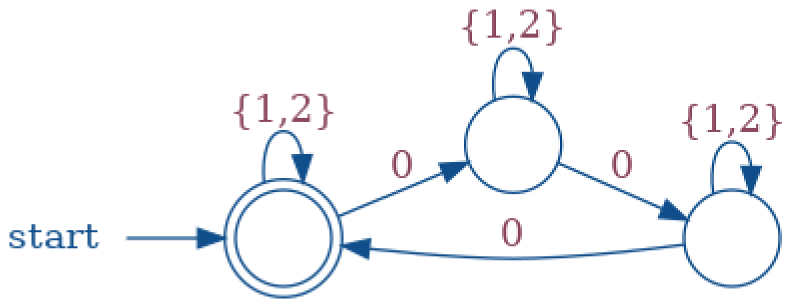
```
D021002010
D0
D22222221
```



Dum:

Starts with a `M`, followed by an arbitrary string of digits {`0 1 2`}. The number of `0`s must be a multiple of three.

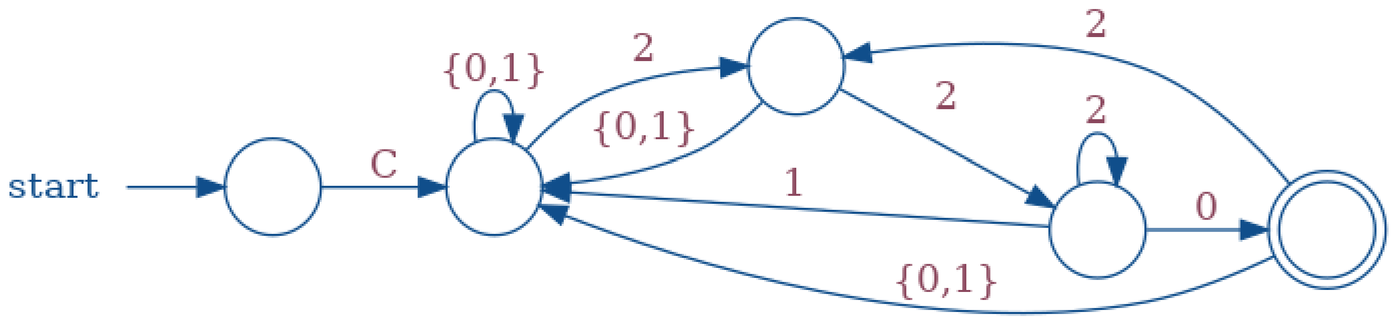
```
M2112122212112
M21002010010211
```



Cheshire:

Starts with a `C` , followed by an arbitrary string of digits {`0 1 2`}. Must end with the string `220` .

```
C220
C102012001201102012222222202222222220220
```



Tweedle:

Starts with an `T` , followed by a `0` or a `1`, finally, followed by a ***Dee*** or ***Dum*** string.

```
T0D021002010
T1D0
T0M2112122212112
T1M21002010010211
```

Input

You will read input from the filename provided as the first argument on the comand line. If you can not open it for reading, print a meaningful message, and quit. If no filename is provided, you will read `stdin` . **Note**, `stdin` is of type `FILE*`, and can be read exactly like a file. See `Labs/C/inputLines.c` for an example.

Output

Print out the message, followed by a space, then `OK` if the message is valid, `FAIL` otherwise. One per line.

For example, given this input:

```
D2221
D020
C210220
```

```
T1M000
d83833
```

You would print:

```
D2221 OK
D020 FAIL
C210220 OK
T1M000 OK
d83833 FAIL
```

Hints

You maybe find these helpful. If you do not, that is okay. They are not directives.

- Read each line, parse the string, or
- Use `fgetc()`, parse the input directly
- Make a function for each message

Style

We've covered style. Use judicious (not excessive) whitespace, proper file header coments, function header comments, other helpful (but not effusive) comments. Apply principles you've learned, to make your code legible.

Makefile

Submit a makefile called **`Makefile`**, that has, minimally, the following targets:

`view`
: Use a pager like `more` or `less` to show me all of your source code

`fsm`
: Compile all of your code into an executable called **`fsm`** .

  You can write your source in multiple files, and add targets as needed, which `fsm` might use to help build dependencies. Indeed, this is encouraged.

`clean`

This target will clean up any object files, if there are any, and any other intermediate files you create.. It does not need to remove the executable, I know where that is.

This target must exist, even if it does nothing. I don't want any errors.

So, this is sorta how I'll test your programs:

```
$ make view
$ make fsm
$ make clean
$ ./fsm my_input_file_probably_not_this_name
```

This shouldn't yield any errors.

What to Submit

- **`Makefile`** — Your makefile, as described above
- All of your source code
- **`README.md`** (optional) — Anything you want to say, before we grade

- Do not submit a .doc file, a .zip file, or a .pdf file. These formats are not correct for this assignment and will not be accepted.
- Make sure you develop and test your program on tux. If it does not run on tux, it will not be considered correct.