Lab 5: Implementing IO redirection (Week of 12.2.18)

In this lab you will be creating your own mini-shell, that is, your program will read in command lines, distinguish between program name, command-line arguments, < and > for redirection, and launch new processes matching the command. It will require you to use fork and exec (hopefully that's obvious) and it will also require you to change standard input or standard output when you receive commands like "mmame < foo.txt" or “mmame > foo.txt”. This we'll do at the file descriptor level. So ask yourself: Which file descriptor is "standard in", “standard out”? How do I change that file descriptor so that it refers to a connection to some other file?

You do NOT need to deal with background jobs: you can assume the '&' character will never appear on your minishell command line. Of course, a shell like bash does have to deal with these things, but for this lab, we're interested in you showing how the shell prepares for and does an exec, not in you showing how to parse the command line.

Getting checked off: Parts 1-5 are due in-lab. Show your program solutions to a TA.

Background

The tar archive contains code for a utility called mmame (min-max-average-median). This utility will read the input from stdin, i.e. reading until end-of-file (^D, CONTROL-D) is encountered, then printing the results:

```
$ mmame                                 mmame reads from stdin
1 2 3                                   the user enters data
^D                                      the user indicates end-of-input
1.000 3.000 2.000 2.000                 results are printed
$                                       back to the shell prompt
```

The mmame utility will also read from a file named on the command line, but after the file has been processed, the program continues reading from standard input! Note that results are not printed until the user enters ^D. This allows the user to continue entering data from stdin after mmame first reads data from the named file:

```
$ mmame test_data.dat                   mmame reads data from test_data.dat
1 2 3                                   the user continues to enter values
^D                                      the user then indicates end-of-input
1.000 32767.000 16601.027 16617.000     results are printed
$                                       back to the shell prompt
```

mmame behaves differently when the shell redirects stdin to read from a file, as follows:

```
$ mmame < test_data.dat                 mmame reads from stdin (redirected from test_data.dat)
55.000 32767.000 16650.824 16633.000    results are printed
$                                       back to the shell prompt
```

Notice that ^D is not entered by the user when IO redirection is handled by the shell.

The command make mmame will compile-and-link the program. Do that, and experiment with it a little bit.

Part 1: Minimal minishell

For Part 1 of the project you are to create a program named minishell. You are given skeleton code for minishell in a file named minishell.c. This is your starting point. If you give the make command, it'll be compiled-and-linked ... although the version your given doesn't do anything. Read the source code and follow the numbered comments! minishell works as follows:

1. The minishell prompt is "IT215> "
2. minishell reads "minishell commands" from stdin, but it is a very simple shell: it "understands" only how to run the program named on its command line. We're testing the minishell with the mmame program, but you may NOT assume the user won't try to run a program other than mmame.
3. Your minishell should read user input, then fork/exec as appropriate to execute mmame.

Here are some examples of the minishell in action for a correct Part 1 solution:

```
$ minishell                     Start the minishell
IT215> mmame                    IT215 is the minishell prompt. Here we're running mmame program
30 25 60 30                     User enters data (mmame reading from stdin)
^D                              User indicates end-of-input with the ^D character
25.000  60.000  36.250  30.000  This is mmame output
IT215>                          mmame terminated: back to the minishell prompt

$ minishell
IT215>                  User enters nothing on the command line: just presses the ENTER key
IT215>                  (returns the user to the minishell prompt)

$ minishell
IT215> grok                                     User enters a command that can't be executed
IT215> minishell: grok: command not found       minishell prints an error message
IT215>                                          ...and returns the user to the minishell prompt
```

Part 2: Reading direct from file

Add to a correctly functioning Part 1 the ability to execute mmame reading directly from a file. Do not attempt Part 2 unless Part 1 works completely and correctly. Here are two example of the minishell in action for a correct Part 2:

```
$ minishell
IT215> mmame test_data.dat
^D
55.000 32767.000 16650.824 16633.000
IT215> mmame test_data.dat
1 2 3
^D
1.000 32767.000 16601.027 16617.000
IT215>
```

Part 3: Input Redirection

Add to a correctly functioning Part 2 the ability to execute mmame using input redirection. Do not attempt Part 3 unless Part 2 works completely and correctly. Here's an example of the minishell in action for a correct Part 3:

```
IT215> mmame < test_data.dat
55.000 32767.000 16650.824 16633.000
```

You may assume that only well-formed commands are entered on your minishell command line. A well-formed command looks like this:

mmame < filename

You DO NOT have to worry about handling lines like these:

mmame foo.dat <	(< In The Wrong Location)
mmame <foo.dat	(Missing Space After <)
mmame <	(Missing Filename After <)
< mmame foo.dat	(mmame not first)

Part 4: Output Redirection

Add to a correctly functioning Part 2 the ability to execute mmame using output redirection. Do not attempt Part 4 unless Part 2 works completely and correctly. Here's an example of the minishell in action for a correct Part 4:

```
IT215> mmame > foo.dat
1 2 3
^D
IT215> ^D
$
```

You may assume that only well-formed commands are entered on your minishell command line. A well-formed command looks like this:

mmame > filename

You DO NOT have to worry about handling lines like these:

mmame filename >	(> In The Wrong Location)
mmame >filename	(Missing Space After >)
mmame >	(Missing Filename After >)
> mmame filename	(mmame not first)

Part 5: Questions

Answer the following questions regarding the skeleton code we gave you. Put your answers in a file named answer.txt:

1) Do a man on fgets and answer the following questions:
   a) Briefly explain what fgets does.
   b) Why is it recommended that fgets is used instead of gets?
   c) Will '\n' appear in the string fgets returns?
   d) What does fgets return if end-of-file is reached and no bytes have been read?

2) Explain what the function strrchr does.

3) Explain the difference between sscanf and fscanf?

4) Explain how sscanf and strstr are used together to extract tokens (or words) from the command line string.

5) Do a man on make and answer the following questions:

   a) What is the name of the default file read by make command?
   b) What must be the first character on every command line in a makefile?
   c) Give a brief description of what a target is with respect to a makefile rule.
   d) Give an example of a pseudo-target that might appear in a makefile.