CS442/542
Shell Project, part I
Due: Friday, September 24

version 1.0 - Tue Sep 9, 2021

Program

Your next project will tie together many of the Operating System/UNIX concepts that we have studied so far this quarter. For the project, you will eventually write a UNIX program called the Betterthan Average Shell, *bash*, which is a simple UNIX command interpreter. To simplify the design of your shell, this first half of the project only deals with the input parsing aspects of the project. This semester, we're going to use the standard Unix tools Flex and Bison to build the input parser for the shell. Flex generates a *lexical analyzer* that breaks an input stream into *tokens.* Bison understands language grammars. Once you've specified the grammar for your `bash`, Bison will do most of the parsing for you. We'll talk about both of these programs extensively in class.

Shell Grammar

To use Lex and Bison, you need to understand the *grammar* that they need to read. For your shell, the command input must be in the following format:

```
cmd (arg)* (<infile)?  (2>(>)?errfile)?)* (| cmd (arg)* (2>(>)?errfile)?)* (>(>)?  outfile)?
```

Input Tokens

Your shell understands the following input tokens:

|
: The PIPE symbol, forward stdout to next cmd

<
: stdin redirection

>
: stdout redirection (create)

>>
: stdout redirection (append)

2>
: stderr redirection (create)

2>>
: stderr redirection (append)

WORD
: Any sequence of characters not already defined as tokens (separated by white space). Alternately, a WORD can be any sequence of characters between double quotes

Example

In the github repository for this assignment, under "examples", there's an example lex/yacc/C program called `arith` that understands a grammar very similar to what you'll need for your shell project. We'll cover that example in great detail in class.

Headstart

In the github repository for this assignment, under "headstart", I've placed a skeleton `bash` "headstart" version with a Makefile stub header files, and etc. You should start from that. It should compile and "run" a little, but you'll need to finish a lot of it before it will do much.

Requirements

Your program must work for *any* reasonable number of arguments and must give meaningful error messages. Your shell should terminate neatly and quietly when it reaches the end of file. You **must** use Flex for lexical analysis and **Bison** for parsing.

Turnin

You must use the class github archive to submit all of your source files, along with a Makefile that generates a program called `bash` that can read input from standard input.

Example Input and Output

For example, if the input to your program looks like this:

```
echo Hello World
/bin/cat < /etc/motd
/bin/cat > outfile
echo "this is a test" | cat
gcc -c -Wall file.c 2>> ERRS
cat < infile | tr "a-z" "A-Z" | cat 2>caterrs | wc -l > out
```

then the output to your program should look something like this:

```
========== line 1 =================
Command name: 'echo'
    argv[0]: 'echo'
    argv[1]: 'Hello'
    argv[2]: 'World'
  stdin:  '<undirected>'
  stdout: '<undirected>'
  stderr: '<undirected>'
========== line 2 =================
Command name: '/bin/cat'
    argv[0]: '/bin/cat'
  stdin:  '/etc/motd'
  stdout: '<undirected>'
  stderr: '<undirected>'
========== line 3 =================
Command name: '/bin/cat'
    argv[0]: '/bin/cat'
  stdin:  '<undirected>'
  stdout: 'outfile'
  stderr: '<undirected>'
========== line 4 =================
Command name: 'echo'
    argv[0]: 'echo'
    argv[1]: 'this is a test'
  stdin:  '<undirected>'
  stdout: PIPE1
  stderr: '<undirected>'
Command name: 'cat'
    argv[0]: 'cat'
  stdin:  PIPE1
  stdout: '<undirected>'
  stderr: '<undirected>'
========== line 5 =================
Command name: 'gcc'
    argv[0]: 'gcc'
    argv[1]: '-c'
    argv[2]: '-Wall'
    argv[3]: 'file.c'
  stdin:  '<undirected>'
  stdout: '<undirected>'
  stderr: 'ERRS' (append)
========== line 6 =================
Command name: 'cat'
    argv[0]: 'cat'
  stdin:  'infile'
  stdout: PIPE1
  stderr: '<undirected>'
Command name: 'tr'
    argv[0]: 'tr'
    argv[1]: 'a-z'
    argv[2]: 'A-Z'
  stdin:  PIPE1
  stdout: PIPE2
  stderr: '<undirected>'
Command name: 'cat'
    argv[0]: 'cat'
  stdin:  PIPE2
  stdout: PIPE3
  stderr: 'caterrs'
Command name: 'wc'
    argv[0]: 'wc'
    argv[1]: '-l'
  stdin:  PIPE3
  stdout: 'out'
  stderr: '<undirected>'
```