

#*

This script translates 'sed' (the unix stream editor) scripts into java source code.

STATUS

Syntax is parsing well, lots of functionality but still missing a
l branching commands, which may be impossible or too difficult to bother with in java.

Also <nth> after s/// for nth occurrence substitution.
Syntax like "s#a#A#;" for substitutions is not supported yet.
The 0~8 gnu sed syntax (every 8th line) is supported . The syntax
a\ c\
i\ is now being parsed with a special .restart trick.

NOTES

Writing a translator for c++ might be more authentic because c++ supports the 'goto' statement, which seems to be needed for the branching commands in (gnu) sed.

It is possible to parse a\ etc by issuing a
restart after every char and checking for \n without \\n.
This is an interesting parsing technique used here for the 1st time.
The workspace is not cleared and .restart is issued to get the text
for a\ one character at a time. See the block B"a\\",B"c\\",B"i\\"
{...}

DONE

changed gnu regex to java - eg \1 \2 -> \$1 \$2 in replacement
but only 9 backrefs at the moment
also, \(\) group -> ()
wrote s///w filename; syntax, write to file if sub has occurred
made a\ i\ and c\ work properly
support /rx/I (insensitive address) and /rx/M multiline
(But what is the meaning of "M" really?)
wrote s///4; only sub 4th occurrence
or "s///g4;" sub 4th occurrence and all occurrences after that.

TODO

Allow a different delimiter for s/// eg "s#a#b#gi" or even
s{a}{b}gi etc I could use the same technique that was used for

a\ c\ i\ but it would be much easier to use the new until; syntax.

make all branching commands work eg : b t T etc, this could be almost impossible in a language that doesnt have goto ...? But c++ does have goto.

BUGS

Some regex patterns maybe java's not gnu seds.

NOTES

Adapting this script for an interpreted language would allow sed scripts to be executed directly within the target language. But not for java which has to be compiled.

string.matches and Pattern.matches matches the whole input string! So I need to add .* to the front and back of regular expressions.

The script uses a similar strategy as tr/translate.java.pss Each machine

command is a method, except trivial commands, for which 'in-line' code can be generated.

The file 'sedlline.txt' can be used to test this script.

This script is recognising a very large subset of gnu sed commands at the moment. Also, it does not parse the regular expressions.

Currently there is a difficulty for the pep machine in dealing with the sed syntax 's#a#A#p'. That is, where alternative delimiters are used for substitutions. This could be solved with a

new 'until' command that looks to the tapecell for the stop condition
(text).

Initially, I will only allow standard s/a/A/p syntax.

HISTORY

26 July 2022

Added the "s/a/A/3;" syntax for doing only the nth substitution. Also "s/a/A/3g;" to sub all occurrences at 3rd and after.

23 july 2022

Added a\ i\ c\ syntax. There may be slight differences with how whitespace is handled in gnu sed with a\ etc.

15 july 2022

Added s///w<filename> syntax.

5 july 2022

Added 0~8 step-range syntax. Also, added ranges with single commands which had been forgotten.

4 july 2022

Adding a/c/i commands but not a\ syntax yet.

converting to using a run() method for translating the script to java source code. However, this is not so useful for a 'compiled

language such as java. For a script language it would be very useful, because the translated sed script could be executed immediately with piped input. But the sed script would have to be

read from file, or a string (not <stdin).

3 july 2022

Implemented lots of commands, but all branching commands are not implemented and may not be. Also a/c/i are not either but can be.

Did y/// command and read file command. There are a lot of details

to cover with this conversion, including seeing how gnu sed behaves. eg s/a/a/p does print the line twice.

1 july 2022

A lot of progress, need to do the rRwW commands. A big challenge are the branching commands tTbB etc because java has no goto. Also need to do the text insert command a/i/c.

Worked on ranges, which seem to be working now.

The grammar is now a little different to eg/sed.parse.pss because of the necessity of generating java code.

30 june 2022

Started to adapt from sed.parse.pss

Code is compiling with very simple commands. Use the pep.sedjas and pep.sedjaf helper functions to test this.

*#

read;

make char number relative to line, for error messages
[\n] { nochars; }

newlines can separate commands in (gnu) sed so we will
just add a dummy ';' here. Also, no trailing ; is required

```

[\n] {
    put; clear; add "/*"; push; .reparse
}

# ignore extraneous white-space?
[:space:] { clear; (eof) { .reparse } .restart }

# comments, convert to java comments
"#" {
    clear; add "/* "; until "\n";
    E"\n" { clip; } add " */\n"; put; clear;
    # uncomment line below to include comments in output
    # add "comment*"; push; .reparse
}

# literal tokens '{' and '}' are used to group commands in
# sed, ';' is used to separate commands and ',' to separate line
# ranges. ! is the postfix negation operator for ranges
"~", ",", "{", "}", ";", "!" { put; add "/*"; push; .reparse }

# various actions: print, delete, swap
"=", "p", "P", "l", "d", "D", "F", "g", "G", "h", "H",
"n", "N", "x", "z" {
    "=" {
        clear;
        # print line-number + newline
        add "this.output.write(String.valueOf(this.linesRead)+'\\n');\n";
        add "this.output.flush(); /* '=' */";
    }
    "d" {
        clear;
        # 'd' delete pattern-space, restart
        # the if true trick is necessary to avoid 'unreachable stateme
        # java compile errors (when multiple 'd' commands are given)
        add "if (true) { this.patternSpace.setLength(0); continue; } /
    * 'd' */";
    }
    "D" {
        clear;
        # add "/* 'D' delete pattern-space to 1st \\n, restart */";
        add "if (this.patternSpace.indexOf(\"\\n\") > -1) {\n";
        add "    this.patternSpace.delete(0, this.patternSpace.indexOf(\\n\"));\n";
        add "    this.readNext = false; if (true) continue; \n";
        add "} else { this.patternSpace.setLength(0); continue; } /* '
d' */";
    }
}

```

```

    }
    "F" {
        # F: print input filename + newline
        # maybe unsupported in java
        clear; add 'this.output.write("<unknown-file>\n"); /* F */';
    }
    "g" {
        # g: replace patt-space with hold-space
        clear;
        add "this.patternSpace.setLength(0); \n";
        add "this.patternSpace.append(this.holdSpace); /* 'g' */";
    }
    "G" {
        # G: append hold-space to patt-space + \n
        clear;
        add "this.patternSpace.append(\"\\n\" + this.holdSpace); /* '
G' */";
    }
    "h" {
        # h: replace hold-space with patt-space
        clear;
        add "this.holdSpace.setLength(0); \n";
        add "this.holdSpace.append(this.patternSpace); /* 'h' */";
    }
    "H" {
        # H: append patt-space to hold-space + newline
        clear;
        add "this.holdSpace.append(\"\\n\" + this.patternSpace); /* '
H' */";
    }
    "l" {
        # print pattern-space unambiguously, synonym for p ?
        clear;
        add "this.output.write(this.patternSpace.toString()+'\n'); /*
'l' */";
    }
    "n" {
        # n: print patt-space, get next line into patt-space
        clear;
        add "if (this.autoPrint) { \n";
        add "    this.output.write(this.patternSpace.toString()+'\n');\n
n}\n";
        add "this.patternSpace.setLength(0);\n";
        add "this.readLine(); /* 'n' */";
    }
    "N" {
        # N: append next line to patt-space + newline
        clear;
        add "this.patternSpace.append('\n'); ";
        add "this.readLine(); /* 'N' */";
    }

```

```

    }
    "p" {
        clear;
        add "this.output.write(this.patternSpace.toString()+'\\n'); /*
'p' */";
    }
    "P" {
        # P: print pattern-space up to 1st newline"
        clear;
        add 'if (this.patternSpace.indexOf("\\n") > -1) {\\n';
        add '    this.output.write(\\n';
        add '        this.patternSpace.substring(0, \\n';
        add '        this.patternSpace.indexOf("\\n")+\\'\\n\\');\\n';
        add "} else { this.output.write(this.patternSpace.toString()+'
\\n'); }";
    }
    "x" {
        # x: # swap pattern-space with hold-space
        clear; add "this.swap(); /* x */";
    }
    "z" {
        # z: delete pattern-space, NO restart
        clear; add "this.patternSpace.setLenth(0); /* z */";
    }
    put; clear; add "action*"; push; .reparse
}

# M and I are modifiers to selectors (multiline and case insensiti
ve)
# eg /apple/Ip; or /A/M,/b/I{p;p}
"M","I" {
    "I" { clear; add "(?i)"; put; }
    "M" { clear; add "(?m)"; put; }
    clear; add "mod*"; push; .reparse
}

# patterns - only execute commands if lines match
# line numbers are also selectors
[0-9] {
    while [0-9]; put; clear; add "number*"; push; .reparse
}
# $ is the last line of the file
"$" {
    put; clear; add "number*"; push; .reparse
}
# patterns - only execute commands if lines match
"/" {
    # save line/char number for error message
    clear; add "near line/char "; lines; add ":"; chars;
    put; clear;

```

```

until "/";
!E"/" {
    clear; add "Missing '/' to terminate ";
    get; add "?\n"; print; quit;
}
clip;
# java .matches method matches whole string not substring
# so we need to add .* at beginning and end, but not if regex
# begins with ^ or ends with $. complicated hey
!E"$" { add ".*$"; }
!B"^" { put; clear; add "^.*"; get; }
put; clear;
# add any delimiter for pattern here, or none
add "'"; get; add "'"; put;
clear;
add "pattern*"; push; .reparse
}

# read transliteration commands
"y" {
    # save line/char number for error message
    clear; add "near line "; lines; add ", char "; chars;
    put; clear;
    # allow spaces between 'y' and '/' although gnu set doesn't
    until "/";
    !E"/",![ /] {
        clear;
        add "Missing '/' after 'y' transliterate command\n";
        add "Or trailing characters "; get; add "\n";
        print; quit;
    }
    # save line/char number for error message
    clear; add "near line "; lines; add ", char "; chars;
    put; clear;
    until "/";
    !E"/" {
        clear;
        add "Missing 2nd '/' after 'y' transliterate command "; get;
        add "\n"; print; quit;
    }
    "/" {
        clear;
        add "Sed syntax error? \n";
        add "    Empty regex after 'y' transliterate command "; get;
        add "\n"; print; quit;
    }
    # replace pattern found
    clip; put;
    clear;

```

```

add 'this.transliterate("' ; get; add '"', "' ; put;
clear;
# save line/char number for error message
add "near line "; lines; add ", char "; chars;
++; put; --; clear;
until "/" ;
!E"/" {
    clear;
    add "Missing 3rd '/' after 'y' transliterate command "; get;
    add "\n"; print; quit;
}
clip; swap; get;
add '"'); /* y */ ' ;
# y/// does not have modifiers (unlike s///)
put; clear;
add "action*"; push; .reparse
}

# this is an artificial block, created by the code below
# which reads multiline append/changes/inserts one char at a time
B"a\\",B"c\\",B"i\\" {
    # print; print; print;
    E"\\\n" {
        # turn multiline into java single line with \n
        # \ means continue text on next line.
        clip; clip; add "\\n";
        .restart
    }
    # end of stream means we are finished, so add a dummy
    # \n
    (eof) { add "\n"; }
    E"\n".!E"\\\n" {
        # finished! the !E"\\\n" above is unnecessary (already checked
)

        # but I will leave for clarity
        clip;
        replace "\n" "\\n";
        B"a\\" {
            clop; clop; put; clear;
            add "this.patternSpace.append('\n'+\""; get; add '");';
        }
        B"c\\" {
            clop; clop; put; clear;
            add "this.patternSpace.setLength(0);\n";
            add "this.patternSpace.append(\""; get; add '");';
        }
        B"i\\" {
            clop; clop; put; clear;
            add "this.patternSpace.insert(0, \""; get; add '+'+'\\n\\');'
;

```



```

    }
    put; clear; add "action*;*"; push; push; .reparse
}
.restart
}
# the add/change/insert commands: have 2 forms
#   a text or a\ <multiline text>
"a","c","i" {
    # ignore intervening space if any
    put; clear;
    while [ \t\f]; clear;
    (eof) {
        clear;
        add "Sed syntax error? (near line:char ";
        lines; add ":"; chars; add ")\n";
        add "  No argument for '"; get; add "' command.\n";
        print; quit;
    }

# also handle the a\ multiline form here
# The following are ok: 'a\ text' 'a  \ text '
#   'a \ text \
#       text'
# So a\ can be terminated by eof, or \n without \

# strategy: read one char, check for \, if so restart
# and write a block "a\\","i\\" etc, and read one char
# at a time until ends with \n but not \\n

# if the first not whitespace char is "\" then we need to read
# the inputstream until it ends with \n but not \\n. This
# is the a\ i\ c\ syntax This is tricky with pep at the moment.
# allowing logic syntax for 'until' would solve this. eg
#   until "\n".!"\\n";
# or allow 2 args to until;

read;
B"\\ " {
    swap; get;
    #print; print; print;
    # now should be a\\ or c\\ or i\\
    # this will be handled by the block above.
    .restart
}

(eof)."\n" {
    clear;
    add "[Sed syntax error?] (near line:char ";
    lines; add ":"; chars; add ")\n";
    add "  No argument for '"; get; add "' command.\n";

```

```

    print; quit;
}

until "\n";
(eof) {
    E"\n" { clip; }
    "" {
        clear;
        add "{Sed syntax error?] (near line:char ";
        lines; add ":"; chars; add ")\n";
        add "    No argument for '"; get; add "'" command.\n";
        print; quit;
    }
}

replace "\n" "\\n";
swap;
"a" {
    clear;
    add "this.patternSpace.append('\n'+\""; get; add ');';
}
"c" {
    clear;
    add "this.patternSpace.setLength(0);\n";
    add "this.patternSpace.append(\""; get; add ');';
}
"i" {
    clear;
    add "this.patternSpace.insert(0, \""; get; add '"+'\n\');';
}
# should work, because 'this' starts with 't' not a/c/i
put; clear;
add "action*;*"; push; push; .reparse
}
# various commands that have an option word parameter
# e has two variants
# "e" { replace "e" "e; # exec patt-space command and replace";
}
"b","e","q","Q","t","T" {
    # ignore intervening space if any
    put; clear;
    while [ ]; clear;
    # A bit more permissive than gnu-sed which doesn't allow
    # read to end in ';'.
    whilenot [ ;]];
    # word parameters are optional to these commands
    # just add a space to separate command from parameter
    !"" { swap; add " "; swap; }
    swap; get;
    # hard to implement because java has no goto ?

```

```

# or try to use labelled loops??
B"b" {
    clear;
    # todo: 'b' branch to <label> or start";
    add 'this.unsupported("b -branch ");\n';
    put; clear;
}
B"e " {
    clear;
    # 'e <cmd>' exec <cmd> and insert into outputfk
    add 'System.out.print(this.execute("'"; get; add '")); /* "e <
cmd>" */';
    put; clear;
}
"e" {
    clear;
    add "temp = this.patternSpace.toString();\n";
    add "this.patternSpace.setLength(0); /* 'e' */\n";
    add 'this.patternSpace.append(this.execute(temp)); ';
    put; clear;
}
"q" {
    # q; print + quit
    clear;
    add "this.output.write(this.patternSpace.toString()+'\\n');\n"
;
    add "System.exit(0);";
    put; clear;
}
B"q " {
    # q; print + quit with exit code
    clon; clon; put; clear;
    add "this.output.write(this.patternSpace.toString()+'\\n');\n"
;
    add "System.exit("; get; add ");";
    put; clear;
}
"Q" {
    # Q; quit, dont print
    clear; add "System.exit(0);";
    put; clear;
}
B"Q " {
    # Q; quit with exit code, dont print
    clon; clon; put; clear;
    add "System.exit("; get; add ");";
    put; clear;
}
B"t" {
    clear;

```

```

# 't' command not implemented yet! \n";
# (branch to <label> if substitution made or start)";
add 'this.unsupported("t - branch ");\n';
put; clear;
}
B"T" {
  clear;
  # 'T' command not implemented yet! \n";
  # (branch to <label> if NO substitution made or start)";
  add 'this.unsupported("T - branch ");\n';
  put; clear;
}
add "action*"; push; .reparse
}

# read 'read <filename>' and write commands
":", "r", "R", "w", "W" {
  # ignore intervening space if any
  put; clear;
  while [ ]; clear;
  # A bit more permissive than gnu-sed which doesn't allow
  # read to end in ';' . i.e. filename can't contain ; or } in
  # this version.
  whilenot [ ;]];
  "" {
    clear;
    add "Sed syntax error? (at line:char ";
    lines; add ":"; chars; add ")\n";
    add "  no filename for read 'r' command. \n";
    print; quit;
  }
  swap; add " "; get;
  B": " {
    clear;
    # todo?: ':' branch to <label>\n";
    # might be hard without 'goto' !";
    add 'this.unsupported(":" - branchlabel ");\n';
    put; clear;
  }
  B"r " {
    clear;
    # r' read file into patt-space
    add '/* "r" */\n';
    add 'Path path = Path.of("'; get; add '");\n';
    add 'File f = new File("'; get; add '"); \n';
    add 'if (f.isFile()) { \n';
    add '  this.patternSpace.append(Files.readString(path));\n';
    add '}";
    put; clear;
  }
}

```

```

B"R " {
    clear;
    # 'R' insert file into output before next line";
    # bug! inserts file immediately into output.
    add '/* "R" */\n';
    add 'Path path = Path.of(""); get; add ");\n';
    add 'File f = new File(""); get; add ");\n';
    add 'if (f.isFile()) { \n';
    add "    this.output.write(Files.readString(path)+'\\n');\n";
    #add '    System.out.println(Files.readString(path));\n';
    add "}";
    put; clear;
}
B"w " {
    clear;
    # 'w' write patt-space to file";
    add 'this.writeToFile(""); get; add ");';
    put; clear;
}
# mm.writeToFile(name)
B"W " {
    clear;
    # 'W' write 1st line of patt-space to file";
    add 'this.writeFirstToFile(""); get; add ");';
    put; clear;
}
add "action*"; push; .reparse
}

# read substitution commands
"s" {
    # save line/char number for error message
    clear; add "near line/char "; lines; add ":"; chars;
    put; clear;
    # allow spaces between 's' and '/' ???
    until "/";
    !E"/",![ /] {
        clear;
        add "Missing '/' after 's' substitute command\n";
        add "Or trailing characters "; get; add "\n";
        print; quit;
    }
    # save line/char number for error message
    clear; add "near line "; lines; add ", char "; chars;
    put; clear;
    until "/";
    !E"/" {
        clear;
        add "Sed syntax error? \n";
        add "    Missing 2nd '/' after 's' substitute command "; get;

```

```

    add "\n"; print; quit;
}
"/" {
    clear;
    add "Sed syntax error? \n";
    add "    Empty regex after 's' substitute command "; get;
    add "\n"; print; quit;
}
# replace pattern found
# legal escape chars in java are \t \b \n \r \f \' \" \\
# anything else will crash the compiler and needs to be elimina
ted
# but may have to live with this
clip;
replace "\\(" "("; replace "\\)" ")" ; replace "\\'" "'";
put;
clear;
add 'this.substitute("' ; get; add '"', "'"; put;
clear;
# save line/char number for error message
add "near line/char "; lines; add ":"; chars;
++; put; --; clear;
until "/";
!E"/" {
    clear;
    add "Missing 3rd '/' after 's' substitute command "; get;
    add "\n"; print; quit;
}
clip;
# this is a hack
replace "\\1" "$1"; replace "\\2" "$2"; replace "\\3" "$3";
replace "\\4" "$4"; replace "\\5" "$5"; replace "\\6" "$6";
replace "\\7" "$7"; replace "\\8" "$8"; replace "\\9" "$9";
swap; get; add '"', "'";
# also need to read s/// modifiers, eg e/w/m/g/i/p/[0-9] etc
# in gnu sed 'w filename' reads filename to end of line, so
# no other commands on that line.
while [emgip0123456789]; add '"', "'";
# now read filename given to 'w' switch (if any)
while [w];
E"w" {
    # gnu-sed allows ';' in filename, but I wont (for now)
    # will need to use substitute method to trim whitespace from
    # the filename and remove leading 'w'
    whilenot [;\n];
}
add '); /* s */ '; put; clear;
add "action*"; push; .reparse
}

```

```

    "b", "T", "t" {
        # branch
        put; clear;
        add "Unimplemented command (near line:char ";
        lines; add ":"; chars; add ")\n";
        add "  The script does not recognise '"; get; add "' yet.\n"; p
rint; quit;
    }

    !" {
        put; clear;
        add "Sed syntax error? (near line:char "; lines; add ":"; chars
; add ")\n";
        add "  unrecognised command '"; get; add "'\n"; print; quit;
    }

# where token reduction begins
parse>

# To visualise token reduction uncomment this below:
add "// "; lines; add ":"; chars; add " "; print; clear;
add "\n"; unstack; print; clip; stack;

# commands do not have to be terminated with ';' at the end of a se
d script.
 eof) {
    pop; "action*" { add ";*"; push; push; .reparse }
    push;
}

pop; pop; pop; pop; pop; pop;
# -----
# 6 token reductions
# these must be done first, to take precedence over
# eg pattern/{/commandset/}
"pattern*,*pattern*{*commandset}*\"",
"pattern*,*number*{*commandset}*\"",
"number*,*number*{*commandset}*\"",
"number*~*number*{*commandset}*\"",
"number*,*pattern*{*commandset}*\" {
    # also, need to indent the command set.
    ++; ++; ++; ++; swap;
    replace "\n" "\n ";
    # use a brace token as temporary storage, so that we can
    # indent the 1st line of the commandset
    # should add 2 spaces but 1st line is getting an extra one.
    # somewhere...
    --; put; clear; add " "; get; ++; swap;
    --; --; --; --;

```

```

# using an array of boolean states to remember if a
# pattern has been 'seen'
B"pattern*,*pattern*" {
    clear;
    add "if (this.line.toString().matches("; get;
    add ") && (this.states["; count; add "] == false))\n {";
    add " this.states["; count; add "] = true; }\n";

    add "if (this.states["; count; add "] == true) {\n";
    # get commandset at tape+4
    ++; ++; ++; ++; get; --; --; --; --;
    add "\n}\n";

    # comes after so last line is matched
    add "if (this.line.toString().matches("; ++; ++; get; --; --;
    add ") && (this.states["; count; add "] == true))\n {";
    add " this.states["; count; add "] = false; }\n";
    put;
    a+;
}

B"pattern*,*number*" {
    clear;
    add "if (this.line.toString()..matches("; get;
    add ") && (this.states["; count; add "] == false))\n";
    add " { this.states["; count; add "] = true; }\n";

    add "if (this.states["; count; add "] == true) {\n ";
    # get commandset at tape+4
    ++; ++; ++; ++; get; --; --; --; --;
    add "\n}\n";

    # put here to match last line in range
    add "if ((this.linesRead > "; ++; ++; get; --; --;
    add ") && (this.states["; count; add "] == true))\n";
    add " { this.states["; count; add "] = false; }\n";

    put;
    a+;
}

B"number*,*pattern*" {
    clear;
    # but this logic doesn't include last line
    add "if ((this.linesRead == "; get;
    add ") && (this.states["; count; add "] == false))\n";
    add " { this.states["; count; add "] = true; }\n";

    add "if (this.states["; count; add "] == true) {\n ";

```



```

# get commandset at tape+4
++; ++; ++; ++; get; --; --; --; --;
add "\n}\n";

# after to match last line in range
add "if (this.line.toString().matches("; ++; ++; get; --; --;
add ") && (this.states["; count; add "] == true))\n";
add "  { this.states["; count; add "] = false; }\n";
put;
a+;
}
B"number*,*number*" {
  clear;
  add "if ((this.linesRead >= "; get;
  add ") && (this.linesRead <= "; ++; ++; get; --; --;
  add ")) {\n";
  # get commandset at tape+4
  ++; ++; ++; ++; get; --; --; --; --;
  add "\n}"; put;
  #a+;
}
B"number*~*number*" {
  # 0~8 step syntax
  clear;
  add "if ((this.linesRead % "; ++; ++; get; --; --;
  add ") == "; get; add ") {\n";
  # get commandset at tape+4
  ++; ++; ++; ++; get; --; --; --; --;
  add "\n}"; put;
}
clear; add "command*"; push; .reparse
}

push; push; push; push; push; push;

# -----
# priority 4 token reductions
pop; pop; pop; pop;
# these must be done first, to take precedence over
# eg pattern/command/} I forgot about this pattern.
# a cool trick! just convert this to {*commandset}* and
# reparse, so we dont have to rewrite all that code
"pattern*,*pattern*command*",
"pattern*,*number*command*",
"number*,*number*command*",
"number*~*number*command*",
"number*,*pattern*command*" {
  # preserve 1st 3 tokens
  push; push; push; clear; get; ++; put; --;
  clear; put;

```

```

    add "{*commandset}*"; push; push; push; .reparse
}
push; push; push; push;

#-----
# 2 tokens:
pop; pop;
# modifiers only come after /.../
E"mod*".!B"pattern*" {
    clear;
    add "[Sed syntax error?] near line:char "; lines; add ","; chars;
add "\n";
    add "  Modifiers (I,M) can only come after line pattern selectors
\n";
    print; quit;
}

"pattern*mod*" {
    # remove quote from start of regex
    clear; get; clop; put; clear;
    # add (?i) or (?m) at the beginning of the java pattern
    add "'"; ++; get; --; get; put; clear;
    add "pattern*"; push; .reparse
}

#-----
# 3 tokens:
#   we have to do this first before the action*;* rule
#   is reduced.
pop;
# change to the equivalent eg: range*{*commandset}*
# This avoids have to rewrite all the java code construction
"range*action*;*", "number*action*;*",
"pattern*action*;*" {
    # preserve range/number/pattern parse token
    push; clear;
    # transfer action/command code to the correct tapecell
    get; ++; put; --;
    clear; add "{*commandset}*"; push; push; push; .reparse
    # now we have on the stack, for example
    # range*{*commandset}* which is already handled, and the
    # code attributes should be in the right tape cells.
    # we could do: add "{*commandset}*" but it doesnt matter....
}

# gnu sed allows empty braces, so we will too.
# Another trick: push an empty commandset onto the stack
# after a brace - that gets rid of this rule and also
# the : command/command/ -> commandset/ rule
"range*{*}*","number*{*}*", "pattern*{*}*" {

```

```

    # preserve 1st 2 tokens
    push; push; clear;
    add " // warning: empty braces {} - does nothing!"; put;
    # add a 'dummy' commandset and reparse.
    clear; add "commandset*}*"; push; push; .reparse
}

push; push; push;

pop; pop;
#-----
# 2 token errors

"pattern*number*", "pattern*pattern*",
"number*number*", "number*pattern*",
"range*number*", "range*pattern*",
"pattern*;*", "number*;*", "range*;*" {
    clear;
    add "Sed syntax error? (near line:char "; lines; add ":"; chars;
add ")\n";
    add "    line selector/number/range with no action \n";
    add "    (missing ',' or misplaced ';' ?) \n";
    print; quit;
}

"action*action*", "action*command*",
"action*number*", "action*pattern*", "action*range*", "action*{*" {
    clear;
    add "Sed error (line "; lines; add ", chars "; chars; add "):\n";
    add "    Missing ';' after command?\n";
    print; quit;
}

",*}*"; ",*{*"; ",*;*"; ",*,*";
";*,*"; ";*{*"; "range*,*" {
    clip; clop; clop; put;
    clear;
    add "Sed error (line "; lines; add ", chars "; chars; add "):\n";
    add "    Unexpected character '"; get; add "' \n";
    print; quit;
}

#-----
# 2 token reductions

# ignore empty commands (and multiple \n)
"command*;*", "commandset*;*", ";*;*" { clip; clip; push; .reparse }

"action*;*" {

```

```

    clear; add "command*"; push; .reparse
}

# maybe need a new token type for clarity here
# eg: negated selector
"number*!*" {
    clear; get; ++; get; --; put;
    clear; add "number*"; push; .reparse
}
"pattern*!*" {
    clear; get; ++; get; --; put;
    clear; add "pattern*"; push; .reparse
}

"command*command*", "commandset*command*" {
    clear; get; ++; add "\n"; get; --; put;
    clear; add "commandset*"; push; .reparse
}

pop;
#-----
# 3 token errors
# eg: '/a/,/bb/p;' or '/[0-3]/,20p;' etc

#-----
# 3 token reductions

# commands dont need a ';' before a closing brace in gnu sed
# so transmogrify
"command*command*}*","command*action*}*","
"commandset*action*}*","commandset*command*}* " {
    clear; get; ++; add "\n"; get; --; put;
    clear; add "commandset*}*"; push; push; .reparse
}
"range*action*}*","number*action*}*","pattern*action*}* " {
    clear; get; add "{\n "; ++; get; add "\n"; --; put;
    clear; add "command*}*"; push; push; .reparse
}
"*action*}* " {
    # make commandset not command for grammar simplicity
    clear; add "{*commandset*}*"; push; push; push; .reparse
}

# a single command in braces can be just treated like a
# set of commands in braces, so lets change to make other
# grammar rules simpler
"*command*}* " {
    # make commandset not command for grammar simplicity
    clear; add "{*commandset*}*"; push; push; push; .reparse
}

```

```

pop;
#-----
# 4 token errors

#-----
# 4 token reductions
"pattern*{*commandset*}*","number*{*commandset*}*" {
    # indent brace commands in tapecell+2
    ++; ++; swap; replace "\n" "\n ";
    # indent 1st line using { token as temporary storage
    --; put; clear; add " "; get; ++; swap;
    --; --;

    "pattern*{*commandset*}*" {
        clear;
        add "if (this.line.toString().matches("; get;
        add ")) {\n"; ++; ++; get; --; --; add "\n}"; put;
    }
    "number*{*commandset*}*" {
        clear;
        add "if (this.linesRead == "; get;
        add ") {\n"; ++; ++; get; --; --; add "\n}"; put;
    }
    clear; add "command*"; push; .reparse
}

pop; pop;
# -----
# 6 token reductions
# none because we have to do them first.

push; push; push; push; push; push;

(eof) {
    # check for valid sed script
    add "/* The token parse-stack was: "; print; clear;
    unstack; add " */\n"; print; clip; clip; clip; clip;
    !"commandset*".!"command*" {
        clear;
        add "# [error] Sed syntax error? \n";
        add "# ----- \n";
        add "# Also, uncomment lines after parse> label in script\n";
        add "# to see how the sed script is being parsed. \n";
        print; quit;
    }
    "commandset*","command*" {
        clear;
        # indent the generated code

```

```

add "\n"; get; replace "\n" "\n"; put; clear;
# create the java preamble, with a 'sedmachine' having a
# holdspace and pattern space
add '

```

```

/* [ok] Sed syntax appears ok */
/* ----- */
/* Java code generated by "sed.tojava.pss" */
import java.io.*;
import java.nio.file.*;
import java.nio.charset.*;
import java.util.regex.*;
import java.util.*; // contains stack

public class javased {
    public StringBuffer patternSpace;
    public StringBuffer holdSpace;
    public StringBuffer line;           /* current line unmodified */
    public int linesRead;
    private boolean[] states;          /* pattern-seen state */
    private Scanner input;             /* what script will read */
    public Writer output;              /* where script will send output */
    private boolean eof;               /* end of file reached? */
    private boolean hasSubstituted;    /* a sub on this cycle? */
    private boolean lastLine;          /* last line of input (for $) */

    private boolean readNext;          /* read next line or not */
    private boolean autoPrint;         /* autoprint pattern space? */

    /** convenience: read stdin, write to stdout */
    public javased() {
        this(
            new Scanner(System.in),
            new BufferedWriter(new OutputStreamWriter(System.out)));
    }

    /** convenience: read and write to strings */
    public javased(String in, StringWriter out) {
        this(new Scanner(in), out);
    }

    /** make a new machine with a character stream reader */
    public javased(Scanner scanner, Writer writer) {
        this.patternSpace = new StringBuffer("");
        this.holdSpace = new StringBuffer("");
        this.line = new StringBuffer("");
        this.linesRead = 0;
        this.input = scanner;
        this.output = writer;
    }

```

```

    this.eof = false;
    this.hasSubstituted = false;
    this.readNext = true;
    this.autoPrint = true;
    // assume that a sed script has no more than 1K range tests! */
    this.states = new boolean[1000];
    for (int ii = 0; ii < 1000; ii++) { this.states[ii] = false; }
}

/** read one line from the input stream and update the machine. */
/
public void readLine() {
    int iChar;
    if (this.eof) { System.exit(0); }
    // increment lines
    this.linesRead++;
    if (this.input.hasNext()) {
        this.line.setLength(0);
        this.line.append(this.input.nextLine());
        this.patternSpace.append(this.line);
    }
    if (!this.input.hasNext()) { this.eof = true; }
}

/** command "x": swap the pattern-space with the hold-space */
public void swap() {
    String s = new String(this.patternSpace);
    this.patternSpace.setLength(0);
    this.patternSpace.append(this.holdSpace.toString());
    this.holdSpace.setLength(0);
    this.holdSpace.append(s);
}

/** command "y/abc/xyz/": transliterate */
public void transliterate(String target, String replacement) {
    // javacode for translit
    //String target      = "ab";
    //String replacement = "**";
    //char[] array = "abcde".toString().toCharArray();
    int ii = 0;
    char[] array = this.patternSpace.toString().toCharArray();
    for (ii = 0; ii < array.length; ii++) {
        int index = target.indexOf(array[ii]);
        if (index != -1) {
            array[ii] = replacement.charAt(index);
        }
    }
    this.patternSpace.setLength(0);
    this.patternSpace.append(array);
}

```

```

/** command "s///x": make substitutions on the pattern-space */
public void substitute(
    String first, String second, String flags, String writeText) {
    // flags can be gip etc
    // gnu sed modifiers M,<num>,e,w filename

    Process p;
    BufferedReader stdin;
    BufferedReader stderr;
    String ss = null;
    String temp = new String("");
    String old = new String(this.patternSpace);
    String opsys = System.getProperty("os.name").toLowerCase();

    // here replace \1 \2 etc (gnu replace group syntax) with
    // $1 $2 etc (java syntax)
    //second = second.replaceAll("\\\\\\"([0-9])", "X$1");
    //System.out.println("sec = " + second);
    // also \(\) gnu group syntax becomes () java group syntax
    // but this is already dealt with, in the parser

    // case insensitive: add "(?i)" at beginning
    if ((flags.indexOf('\''i\'') > -1) ||
        (flags.indexOf('\''I\'') > -1)) { first = "(?i)" + first; }

    // multiline matching, check!!
    if ((flags.indexOf('\''m\'') > -1) ||
        (flags.indexOf('\''M\'') > -1)) { first = "(?m)" + first; }

    // <num>- replace only nth match
    // todo

    // gnu sed considers a substitute has taken place even if the
    // pattern space is unchanged! i.e. if matches first pattern.
    if (this.patternSpace.toString().matches(".*" + first + ".*"))
    {
        this.hasSubstituted = true;
    }

    // syntax "s/a/A/3;" replace nth (3rd) occurrence of match
    if (flags.matches(".*[0-9]+.*")) {
        String[] parts = flags.replaceAll("[^0-9]+", " ").trim().split(" ");
        int nn = Integer.parseInt(parts[0]);
        //System.out.println("nn=" + nn);
        int ii = 0;
        int index = -1;
        Pattern pp = Pattern.compile(first);
        Matcher m = pp.matcher(this.patternSpace.toString());

```



```

temp = this.patternSpace.toString();
while(m.find()) {
    ii++;
    //System.out.println("ii=" + ii);
    if (ii >= nn) {
        index = m.start();
        temp = this.patternSpace.toString();
        this.patternSpace.setLength(0);
        this.patternSpace.append(temp.substring(0,index));
        this.patternSpace.append(
            temp.substring(index).replaceFirst(first, second));
        temp = this.patternSpace.toString();
        // trying to match gnu sed behavior where the "g" and "nt
h"
        // occurrence are combined (i.e. replace all matches from
        // the nth occurrence.
        if (flags.indexOf('\g\') == -1) { break; }
    }
}
else if (flags.indexOf('\g\') != -1) {
    // sed syntax "s/a/A/g;" g- global, replace all matches
    temp = this.patternSpace.toString().replaceAll(first, second)
;
} else {
    // sed syntax "s/a/A/;" replace only 1st match
    temp = this.patternSpace.toString().replaceFirst(first, secon
d);
}
this.patternSpace.setLength(0);
this.patternSpace.append(temp);
try {
    if (this.hasSubstituted) {
        // only print if substitution made, (but pattern-space may
be
        // unchanged, according to gnu sed).
        if (flags.indexOf('\p\') != -1) {
            this.output.write(this.patternSpace.toString()+"\\n\");
        }
        // execute pattern space, gnu ext
        if (flags.indexOf('\e\') != -1) {
            this.output.write(this.execute(this.patternSpace.toString
g())));
            //System.out.print(this.execute(this.patternSpace.toStrin
g())));
        }
        // write pattern space to file, gnu extension, if sub occur
red
        // The writeText parameter contains '\w\' switch plus possi
ble
        // whitespace.

```

```

        if (writeText.length() > 0) {
            writeText = writeText.substring(1).trim();
            this.writeToFile(writeText);
        }
    }
} catch (IOException e) {
    System.out.println(e.toString());
}

}

/** execute command/pattspace for s///e or e <arg> or "e" */
public String execute(String command) {
    Process p;
    BufferedReader stdin;
    BufferedReader stderr;
    String ss;
    StringBuffer output = new StringBuffer("");
    try {
        if (System.getProperty("os.name").toLowerCase().contains("win
")) {
            p = Runtime.getRuntime().exec(new String[]{"cmd.exe", "/c",
command});
        } else {
            p = Runtime.getRuntime().exec(new String[]{"bash", "-c", co
mmand});
        }
        stdin = new BufferedReader(new InputStreamReader(p.getInputSt
ream()));
        stderr = new BufferedReader(new InputStreamReader(p.getErrorS
tream()));
        while ((ss = stdin.readLine()) != null) { output.append(ss +
'\n'); }
        while ((ss = stderr.readLine()) != null) { output.append(ss +
'\n'); }
    } catch (IOException e) {
        System.out.println("sed exec \'e\' failed: " + e);
    }
    return output.toString();
}

/** command "W": save 1st line of pattern space to filename */
public void writeFirstToFile(String fileName) {
    try {
        File file = new File(fileName);
        Writer out = new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream(file), "UTF8"));
        // get first line of ps
        out.append(this.patternSpace.toString().split("\\R", 2)[0]);
        // yourString.split("\\R", 2);
    }
}

```

```

        out.flush(); out.close();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

/** command "w": save the pattern space to filename */
public void writeToFile(String fileName) {
    try {
        File file = new File(fileName);
        Writer out = new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream(file), "UTF8"));
        out.append(this.patternSpace.toString());
        out.flush(); out.close();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

/** handle an unsupported command (message + abort) */
public void unsupported(String name) {
    String ss =
        "The " + name + "command has not yet been implemented\\n" +
        "in this sed-to-java translator. Branching commands are hard i
n\\n" +
        "in a language that doesn't have \'goto\'. Your script will n
ot \\n" +
        "execute properly. Exiting now... \\n";
    System.out.println(ss); System.exit(0);
}

/** run the script with reader and writer. This allows the code t
o
    be used from within another java program, and not just as a
    stream filter. */
public void run() throws IOException {
    String temp = "";
    while (!this.eof) {
        this.hasSubstituted = false;
        this.patternSpace.setLength(0);
        // some sed commands restart without reading a line...
        // hence the use of a flag.
        if (this.readNext) { this.readLine(); }
        this.readNext = true;
    }
    ;
    get;
    add '
        if (this.autoPrint) {
            this.output.write(this.patternSpace.toString() + '\\n\\n');
            this.output.flush();
        }
    }
}

```

```

    }
}

/* run the script as a stream filter. remove this main method
   to embed the script in another java program */
public static void main(String[] args) throws Exception {

    // read and write to stdin/stdout
    javased mm = new javased();
    // new Scanner(System.in),
    // new BufferedWriter(new OutputStreamWriter(System.out));
    mm.run();

    // convert sedstring to java and write to string.
    // javased mm = new javased("/class/s/ass/ASS/g", new StringWri
ter());
    // then use mm.output.toString() to get the result (java source
code)
}
}
';
    print;
}
quit;
}

```