```
#!/usr/bin/perl -w
use strict;

# Michael E Sparks, 24 Dec 2020

# Testing framework to verify our Scheme-based HP-12C emulator/
# Forth interpreter behaves as expected.

# SAMPLE USAGE:
# $ ./interpreter_test.pl 2 3 1 + 2 + \* 3 + 5 / 42 + 3 -
# 42
# $ ./interpreter_test.pl 2 3 4 5 \* + 3
# 2 23 3

# assume forthLite.scm is in current directory.
# if available on $PATH, just set $MYPATH="";
my $MYPATH=`pwd`;
chomp($MYPATH);
$MYPATH .= '/';

# slurp in Forth code (in reverse Polish notation) from CLI
my $rpn_code=join(" ",@ARGV);

# instruct gforth interpreter to display stack, print newline & exit
$rpn_code.=" .s cr bye";

# invoke it
my $res1=`gforth -e "$rpn_code"`;

# strip off ornamentation from gforth result/ expected value
chomp($res1); # clip off '\n', if it exists
$res1=~s/^\S+\s+//; # clip off first token
$res1=~s/\s*$//; # clip off any residual whitespace

# our guile script doesn't interpret gforth directives
$rpn_code=~m/^(.*?) \.s cr bye$/;
$rpn_code=$1;
$rpn_code=~s/\*/\\\*/g; # must escape '*' at shell, else globbing

# invoke the guile script
my $res2=`${MYPATH}forthLite.scm $rpn_code`;

# strip off ornamentation from scheme result/ observed value
chomp($res2);
$res2=~tr/()//d;

# note that Forth results need not be scalars!
if($res1 eq $res2) {
  print $res1,"\n";
  exit 0;
}
else {
  print STDERR "\nDiscrepancy detected:
\t  gforth gave \"$res1\"
\tforthLite gave \"$res2\"\n\n";
  exit 1;
}
```

```
#!/usr/bin/guile -s
!#

;; Michael Sparks, 24 Dec 2020

;; Implement a subset of Forth/ HP-12C functionality in
;; Scheme, handling only the words +, -, * and /.

;; Error checking is not performed, and it is assumed
;; only well-formed Forth code is passed to the interpreter.
;; The stack is returned to the shell as a list of strings.

(define-macro (apply-word op stack)
  `(number->string
    (let ((opc (car (string->list ,op)))
          (arg1 (string->number (cadr ,stack)))
          (arg2 (string->number (car ,stack))))
      (cond ((eqv? opc #\+) (+ arg1 arg2))
            ((eqv? opc #\-) (- arg1 arg2))
            ((eqv? opc #\*) (* arg1 arg2))
            ((eqv? opc #\/) (/ arg1 arg2))
            (else #f)))))

(define (parse-forth stack code)
  (cond ((null? code) (begin (write (reverse stack))
                             (newline)))
        ((let ((op (car code)))
           (or (string=? "+" op) (string=? "-" op)
               (string=? "*" op) (string=? "/" op)))
         (parse-forth
          (cons (apply-word (car code) stack)
                (cddr stack))
          (cdr code)))
        (else (parse-forth (cons (car code) stack)
                           (cdr code)))))

(parse-forth '() (cdr (command-line)))
```