COMPILERS QUIZ 3: CONDITIONALS AND LOOPS IN FORTH

In this part, you are expected to complete all features from quiz 2 (both take-home and in-class) and implement the following features on top of it.

1 String comparison

Add s= and s!= for string comparison. lex<, lex>, lex<=, and lex>= for lexicographic string comparison.

2 Booleans

Add b=, b!= for comparing Booleans.

3 Some list manipulation

len
    [x1 xn] S to n S
listn
    n xn x1 S to [x1 xn] S.
list
    Turns all values in stack into a list.

4 Stored procedures

We introduce programs as values.

We add two special words { and }. The general form is:

```
{ <program> }
```

where <program> is an arbitrary program. This is treated as a single value that is stored on the stack.

The program

```
{ "Hello" print }
```

has no effect.

Add a run command to run the stored procedure on top of the stack.

```
{ "Hello" print } run
```

prints Hello to screen.

Using this new type of value, you can implement the following program:

```
get 0 >
{ "Positive" print }
{ "Not positive" print }
```

```
  if
```

which will check whether the user entered number is positive or not and print the correct message.

Here,

```
  print prints the top value in stack in a human-readable way.
  if expects stack to be ELSE THEN b S where ELSE and THEN are stored
  procedures and b is a Boolean value. If b is true, THEN is executed,
  else ELSE is executed. Notice that before THEN or ELSE is executed,
  stack should just be S.
```

Implement some loop constructs:

repeat

> Stack should be P N S where N is a natural number and P is a stored procedure. Execute P, N times. Remove P and N from the stack at the beginning of execution.

whlie

> Stack should be B C S where B and C are stored procedures. C should produce a Boolean, if that is true, then B is executed after removing true from the stack, if that is false, terminate the loop and remove false from the stack. At the beginning, both B and C are removed from the stack so that C's execution does not see B and C on the stack.

```
  10 { "." print } repeat
```

prints 10 dots to the screen.

```
  10 { dup 0 > } { dup print dec } while
```

will print a countdown from 10 to 1. dec decrements the number by 1.