Build:

```
gcc main.c -o BASIC
```

Usage:

```
./BASIC [filename]
```

Commands:

run: rerun the program
quit: exit the interpreter

Language Specification:

Variables:

One can specify a variable by assigning a value to an identifier (defined with the regex `/[a-zA-Z\$][a-zA-Z0-9\$]+/`). Variables ending with a $ are interpreted as string values. Any variable that doesn't end with a $ is considered an integer.

```
MYINT = 10
MYSTRING$ = "Hello"
```

Booleans:

Boolean values are not explicitly supported. Instead use the integer value 1 as true and 0 as false.

Line Numbers:

We do not consider physical line numbers as the line numbers for the program. Instead the user must manually specify the line number for each line. To do this every statement must start with an unsigned integer followed by a statement. See 'Example Programs' below for furthur details.

Statements:

- **GOTO [line number]**:

Set the program counter to the line specified by [line number]

- **GOSUB [line number]**:
  Set the program counter to the line specified by [line number] and push the current line number onto the subroutine stack. See 'RETURN' usage for furthur details.

- **INPUT [variable name]**:
  Prompt the user to input a value for [variable name]. Whether the user must input a string or an integer is determined by the variables name. If the variable name ends with the $ then the interpreter expects the user to input a string, otherwise an integer is expected.

- **PRINT [variable name]**:
  Write the value stored in [variable name] to the standard output.

- **RETURN**:
  Pop the last line number from the stack and set the program counter to it.

- **IF [variable name] THEN GOTO [line number]**:
  If [variable name] is true then set the program counter to [line number].

- **[variable name](1) = NOT [variable name](2)**:
  Set the value of [variable name](1) to false if [variable name](2) is true else set it to false.

- **[variable name] = [arg1] [op] [arg2]**:
  Where [arg1] and [arg2] can be a variable or integer. [op] is defined to be one of the operators from the set {+-*<>}. Apply the [op] to both [arg1] and [arg2] and store the resulting value in [variable name].

- **[variable name] = [arg1] = [arg2]**:
  Where [arg1] and [arg2] can be a variable, integer or string. If [arg1] is equal to [arg2] the store 1 in [variable name] else store 0 in [variable name].

- **[variable name] = [arg]**:
  Where [arg1] can be a variable, integer or string. Store the value of [arg] in [variable name].

Example Programs:

Factorial program:

Implements an iterative version of the factorial algorithm. The user is asked to input a number which the program then calculates the factorial value and prints it out.

```
10 INPUT N
20 RESULT = 1
30 DONE = N = 0
40 IF DONE THEN GOTO 80
```

```
50 RESULT = RESULT * N
60 N = N - 1
70 GOTO 30
80 PRINT RESULT
```

Guessing game:

Asks the user to input a value until the user guesses the correct value. If the user guesses wrong ask the user again. Line 3 uses the GOSUB statement to call a function-like set of statements.

```
1 WRONG$ = "WRONG"
2 CORRECT$ = "CORRECT"
3 GOSUB 6
4 IF EQ THEN GOTO 13
5 GOTO 3
6 INPUT answer
7 EQ = answer = 7
8 IF EQ THEN GOTO 11
9 PRINT WRONG$
10 RETURN
11 PRINT CORRECT$
12 RETURN
13 END
```