Computer Science 420: TruPL - The **Tru**man **P**rogramming **L**anguage, Lexical Conventions

This list enumerates the legal lexemes of TruPL. The input alphabet consists of all lowercase alphabetic ASCII characters, the digits '0' through '9', and the following non-alphanumeric characters:

```
; : ( ) , = < > + - * / # space tab newline
```

All other characters are illegal in TruPL lexemes.

We use the following shorthand notation when describing the lexemes: `[a..z]` is a regular expression that matches any single lowercase ASCII character. The regular expression `[0..9]` matches any digit.

Lexeme	**Regular Expression for Lexeme**	**Attribute**
keyword	*program*	`kw_program`
	procedure	`kw_procedure`
	int	`kw_int`
	bool	`kw_bool`
	begin	`kw_begin`
	end	`kw_end`
	if	`kw_if`
	then	`kw_then`
	else	`kw_else`
	while	`kw_while`
	loop	`kw_loop`
	print	`kw_print`
	not	`kw_not`
punctuation	`;`	`punc_semi`
	`:`	`punc_colon`
	`,`	`punc_comma`
	`:=`	`punc_assign`
	`(`	`punc_open`
	`)`	`punc_close`
relational operator	`=`	`relop_eq`
	`<>`	`relop_neq`
	`>`	`relop_gt`

	`>=`
	`<`
	`<=`
		`relop_ge`
		`relop_lt`
		`relop_le`

additive operator	`+`
	`-`
	`or`
		`addop_add`
		`addop_sub`
		`addop_or`

multiplicative operator	`*`
	`/`
	`and`
		`mulop_mul`
		`mulop_div`
		`mulop_and`

identifier	`[a..z]([a..z]|[0..9])`*	the identifier name
num	`[0..9][0..9]`*	the number

Assumptions:

- Comments begin with a '`#`' and continue until the end of the current line. Any ASCII character may appear in a comment. Your lexical analyzer should skip over all comments.

- Whitespace consists of one or more repetitions of the space character, the tab character ('`\t`') or the newline character ('`\n`'). Whitespace may not appear inside any token.

- If a keyword, identifier, `and`, or `or` is followed by another keyword, identifier, num, `and`, or `or`, then the tokens will be separated by whitespace. Thus, "`andnota23`" should be recognized as a single token of type `identifier`, but "`and nota23`" should be recognized as two tokens, and "`1and not a23`" should be recognized as four.

- If two relational operators appear next to each other, the two tokens will be separated by whitespace. Thus, "`>=`" should be recognized as a single token, but "`> =`" should be recognized as two tokens.

- If the `:` token is followed by the `=` token, the two tokens will be separated by whitespace. ":=" should be recognized as the `punctuation` token.