Floyd Lexical Components

The following is a complete list of lexical components for Floyd:

- **Newline.** Floyd is not a free-form language. Many Floyd constructs are terminated by a newline sequence (UNICODE 10, or UNICODE 13 followed by UNICODE 10), so the scanner must report the newline symbol as a token. A newline may only appear where indicated in the grammar, unless it is immediately preceded by the line continuation symbol, '_', in which case the scanner should silently consume both the _ and the newline without reporting either to the parser.

- **Comment.** A comment begins with a tilde ~ and continues to the end of the line. The scanner must not report comments to the parser, but must report the terminating newline.

- **White space.** Any sequence of one or more space characters or tabs is white space. The scanner should ignore them.

- **Keyword.** All of the following are keywords:

  boolean begin class else end false from if inherits int is loop me new ~~not~~ null string then true while

  Each keyword should be its own token (don't have a token type "keyword" that lumps all keywords together).

- **Identifier (name).** An identifier is a sequence of letters, digits, and underscores (_) starting with an underscore or letter, other than a keyword.

  Note that Floyd is *case sensitive* with respect to keywords and identifiers: `begin` is a keyword, but BEGIN is an identifier.

- **Integer literal.** An integer literal is any nonempty sequence of decimal digits, with an optional prefix - to indicate a negative quantity.

  Examples: 35, 0, -15

- **String literal.** Strings begin and end with the double quote character. A string consists of any number of printable ASCII characters except a double quote or

backslash. In addition, a string may contain an escape sequence beginning with a backslash. There are two kinds of escape sequences: "named" and "octal". An octal escape is a backslash followed by three octal digits (0-7) and represents the corresponding 8-bit byte. The named escape sequences are in the table below. Here's a valid string: "Hi, \"Tom\", \nHow are \333things\222 today?"

Escape	Name	Octal
\t	tab	011
\n	newline	012
\f	form feed	014
\r	carriage return	015
\"	double quote	042
\\	backslash	134

- **Predefined operator.** Certain symbols stand for predefined operations. The list of operators follows. Note that and, or, and not are listed here for completeness, but should be categorized as keywords, not operators.

Operator	Description
&	String concatenation
+	integer and real addition; unary positive
-	integer and real subtraction; unary negative
*	integer and real multiplication
/	integer and real division
and / or / not	logical and / or / not operators
> / >= / =	greater than, greater than or equal, equal operators

- **Miscellaneous.** Each token listed here should be categorized separately.

Token	Description	Token	Description
:=	assignment op	,	comma
(	open parenthesis	;	semicolon
)	close parenthesis	:	colon
[	open bracket	.	period
]	close bracket		

Sample Input

The following is a sample Floyd program *test.floyd*:

```
 1 ~ This is a comment
 2
 3 class Main is
 4   x: int         ~ x coord
 5   name: string   ~ identifier
 6
 7   start() is
 8     i: int
 9     begin
10       print("Hey,\"Sue!\"" & name)
11     end
12 % ~ The % is a bad token
13 end "Unterminated
14 "Hey\q"
```

Expected Output

```
demo.floyd:1,20:cr
demo.floyd:2,1:cr
demo.floyd:3,1:keyword:class
demo.floyd:3,7:identifier:Main
demo.floyd:3,12:keyword:is
demo.floyd:3,14:cr
demo.floyd:4,3:identifier:x
demo.floyd:4,4:':'
demo.floyd:4,6:keyword:int
demo.floyd:4,25:cr
demo.floyd:5,3:identifier:name
demo.floyd:5,7:':'
demo.floyd:5,9:keyword:string
demo.floyd:5,28:cr
demo.floyd:6,1:cr
demo.floyd:7,3:identifier:start
demo.floyd:7,8:'('
demo.floyd:7,9:')'
demo.floyd:7,11:keyword:is
demo.floyd:7,13:cr
demo.floyd:8,5:identifier:i
demo.floyd:8,6:':'
demo.floyd:8,8:keyword:int
demo.floyd:8,11:cr
demo.floyd:9,5:keyword:begin
demo.floyd:9,10:cr
demo.floyd:10,7:identifier:print
demo.floyd:10,12:'('
demo.floyd:10,13:string lit:"Hey,\"Sue!\""
demo.floyd:10,28:operator:'&'
demo.floyd:10,30:identifier:name
demo.floyd:10,34:')'
demo.floyd:10,35:cr
demo.floyd:11,5:keyword:end
demo.floyd:11,8:cr
demo.floyd:12,1:Unrecognized char: %
demo.floyd:12,25:cr
demo.floyd:13,1:keyword:end
demo.floyd:13,5:Unterminated string:"Unterminated
demo.floyd:13,18:cr
demo.floyd:14,1:Illegal string:"Hey\q"
demo.floyd:14,8:cr
```