The C-- Programming Language

Tokens

The tokens of the C-- language are defined as follows:

- Reserved words.

-
  ```
   int   bool   void    true  false  if
   else  while  return  cin   cout
  ```
- Identifiers. Any sequence of one or more letters, digits, or underscores, starting with a letter or underscore, that is not a reserved word.

- Integer literals. Any sequence of one or more digits.

- String literals. A sequence of zero or more *string characters* surrounded by double quotes. A "string character" is either a single character other than a newline, double-quote, or backslash, or an *escape sequence* consisting of a backslash followed by a single quote, a double quote, the letter t, the letter n, or another backslash.

  Examples of legal string literals:

  ```
      ""
      "&!#"
      "use \n to denote a newline character"
      "include a quote like this \" and a backslash like this \\"
  ```

  Examples of things that are not legal string literals:

  ```
      "unterminated
      "also unterminated \"
      "backslash followed by space: \ is not allowed"
      "bad escaped character: \a AND not terminated
  ```

- Any of the following one- or two-character symbols

-
  ```
   {     }     (     )     [     ]     ,     =     ;
  ```

-
  ```
   +     -     *     /     !     &&    ||    ==    !=
   <     >     <=    >=    <<    >>
  ```

Comments

Text starting with a double slash (//) or a sharp sign (#) up to the end of the line is a comment (except of course if those characters are inside a string literal). For example:

```
    // this is a comment
    # and so is this
```

The scanner should recognize and ignore comments (but there is no COMMENT token).

Whitespace

Spaces, tabs, and newline characters are whitespace. Whitespace separates tokens, but should otherwise be ignored (except inside a string literal).

Invalid Characters

Any character that is not whitespace and is not part of a token or comment is invalid.

Length

You may *not* assume any limits on the lengths of identifiers, string literals, integer literals, comments, etc.

The C-- grammar

```
program
    : program varDecl
    | program fnDecl
    | /* empty */
    ;

varDecl
    : type id ';'
    | type id '[' INTLITERAL ']' ';'
    ;

type
    : INT
    | BOOL
    | VOID
    ;

fnDecl
    : type id parameters block
    ;

parameters
    : '(' ')'
    | '(' formalsList ')'
    ;

formalsList
    : formalDecl
    | formalsList ',' formalDecl
    ;

formalDecl
    : type id
    ;

block
    : '{' declList stmtList '}'
    ;

declList
    : declList varDecl
    | /* empty */
    ;

stmtList
    : stmtList stmt
```

```
        | /* empty */
        ;

stmt
        : CIN READ id ';'
        | CIN READ id '[' exp ']' ';'
        | COUT WRITE exp ';'
        | subscriptExpr '=' exp ';'
        | id '=' exp ';'
        | IF '(' exp ')' block
        | IF '(' exp ')' block ELSE block
        | WHILE '(' exp ')' block
        | RETURN exp ';'
        | RETURN ';'
        | fnCallStmt ';'
        ;

exp
        : exp '+' exp
        | exp '-' exp
        | exp '*' exp
        | exp '/' exp
        | '!' exp
        | exp ANDAND exp
        | exp OROR exp
        | exp EQEQ exp
        | exp NOTEQ exp
        | exp '<' exp
        | exp '>' exp
        | exp LESSEQ exp
        | exp GREATEREQ exp
        | '-' atom
        | atom
        ;

atom
        : INTLITERAL
        | STRINGLITERAL
        | TRUE
        | FALSE
        | '(' exp ')'
        | fnCallExpr
        | subscriptExpr
        | id
        ;

fnCallExpr
        :  id '(' ')'
        | id '(' actualList ')'
        ;

fnCallStmt
        :  id '(' ')'
        | id '(' actualList ')'
        ;

actualList
```

```
    : exp
    | actualList ',' exp
    ;

subscriptExpr
    : id '[' exp ']'
    ;

id
    : ID
    ;
```

The operators in C-- have precedence, from lowest to highest as follows:

```
    ||
    &&
    <, <=, >, >=, ==, !=
    +, -
    *, /
    !, unary -
```

On each line, the operators have equal precedence. All operators are left-associative except for the comparison oprators (<=, ==, etc.) which have no associativity. For example, the expression `a <= b == c` is illegal.