```
Summary file for decaf-daniel-byers
Name: put your name here
Student number: put your student number here
Lab class: put the day and time of your Software Architectures lab class here
********************
   LEXER TESTING
*** Legal tests ***
legal-01 produced incorrect output
legal-02 produced incorrect output
legal-03 produced incorrect output
legal-04 produced incorrect output
legal-05 produced incorrect output
legal-06 produced incorrect output
legal-07 produced incorrect output
legal-08 produced incorrect output
legal-09 produced incorrect output
legal-10 produced incorrect output
legal-11 produced incorrect output
legal-12 produced incorrect output
legal-13 produced incorrect output
legal-14 produced incorrect output
legal-15 produced incorrect output
legal-16 produced incorrect output
legal-17 produced incorrect output
legal-18 produced incorrect output
legal-19 produced incorrect output
legal-20 produced incorrect output
legal-21 produced incorrect output
legal-22 produced incorrect output
legal-23 produced incorrect output
legal-24 produced incorrect output
legal-25 produced incorrect output
*** Illegal tests ***
illegal-01 - error not detected correctly
illegal-02 CORRECT
illegal-03 CORRECT
illegal-04 CORRECT
illegal-05 CORRECT
illegal-06 CORRECT
illegal-07 CORRECT
illegal-08 - error not detected correctly
illegal-09 CORRECT
illegal-10 - error not detected correctly
```

update ABOUT file.

Read the output specifications for stage 1 and make sure you modify Main.java to give the correct behaviour.

```
/**
 * @author Daniel Byers | 13121312
 *
 * This code builds on examples provided by the following book:
 * Parr, Terence (2012). The Definitive ANTLR 4 Reference. USA: The Pragmatic Bo
okshelf. 322.
 */

lexer grammar DecafLexer;

// Keywords:
CLASS     : 'class';
BOOLEAN   : 'boolean';
FOR       : 'for';
BREAK     : 'break';
IF        : 'if';
ELSE      : 'else';
CALLOUT   : 'callout';
INT       : 'int';
RETURN    : 'return';
CONTINUE  : 'continue';
VOID      : 'void';

// Special terminals
LCURLY : '{';
RCURLY : '}';
LBRACE : '[';
RBRACE : ']';
LPAREN : '(';
RPAREN : ')';
COMMA  : ',';
EOL    : ';';
PERIOD : '.';

// Operators
ADDITION    : '+';
MINUS       : '-';
DIVISION    : '/';
MULTIPLY    : '*';
MODULO      : '%';

ASSIGNMENT  : '=';
ASSIGNMENTP : '+=';
ASSIGNMENTS : '-=';

EQUAL       : '==';
NOTEQUAL    : '!=';
LESSTHAN    : '<';
GREATERTHAN : '>';
LSSTHNEQTO  : '<=';
GRTTHNEQTO  : '>=';

AND         : '&&';
OR          : '||';
NOT         : 'not' | '!';


// Any number in the range zero to nine.
INTLITERAL : (DECLITERAL | HEXLITERAL);

// Any instance of the boolean expressions of truth and non-truth.
BOOLEANLITERAL : ('true' | 'false');
```

has no meaning in Decaf so should generate an error.


```
// One character enclosed within single quotes.
CHARLITERAL : '\'' CHAR '\'';

// Any number of characters enclosed within double quotes.
STRINGLITERAL : '"' CHAR* '"';

// This rule simply ignores any space, newline, tab, linefeed or empty quot
WS_ : (' ' | '\n' | '\t' | '\f' | '\'\'' ) -> skip;

// This rule ignores comments ('//' to the end of the line).
SL_COMMENT : '//' (~'\n')* '\n' -> skip;

// A lower or uppercase letter or underscore, followed by none or more
// alphanumeric characters or underscore.
IDENTIFIER : APLHA ALPHANUM*;

fragment
LITERAL: (INTLITERAL | CHARLITERAL | BOOLEANLITERAL | STRINGLITERAL);

fragment
BIN_OP: (ARITH_OP | REL_OP | EQ_OP | COND_OP);

fragment
ARITH_OP: (ADDITION | MINUS | MULTIPLY | DIVISION | MODULO);

fragment
REL_OP: (LESSTHAN | GREATERTHAN | LSSTHNEQTO | GRTTHNEQTO);

fragment
EQ_OP: (EQUAL | NOTEQUAL);

fragment
COND_OP: (AND | OR);

// Fragments to hold certain sets of characters.
fragment
ESC :  '\\' ('n' | 't' | '\\' | '"' | '\'');

fragment
ALPHANUM : (APLHA | DIGIT);

fragment
APLHA : [a-zA-Z_];

fragment
DIGIT : [0-9];

fragment
HEXDIGIT: (DIGIT | [a-fA-F]);

fragment
DECLITERAL : DIGIT DIGIT*;

fragment
HEXLITERAL : '0x' HEXDIGIT HEXDIGIT*;

//fragment
NONWORD: [\u0020-\u0021\u0023-\u0026\u0028-\u002F\u003A-\u0040\u005B\u005D\u
\u0060\u007B-\u007E];

fragment
```

think you may have single quote in this set (you should not.)

```
CHAR : (ESC | ALPHANUM | NONWORD);
```