```
Summary file for decaf-Muzual
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 CORRECT
legal-07 CORRECT
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 CORRECT
legal-16 CORRECT
legal-17 produced incorrect output
legal-18 CORRECT
legal-19 produced incorrect output
legal-20 CORRECT
legal-21 produced incorrect output
legal-22 produced incorrect output
legal-23 produced incorrect output
legal-24 produced incorrect output
legal-25 CORRECT
*** Illegal tests ***
illegal-01 CORRECT
illegal-02 CORRECT
illegal-03 CORRECT
illegal-04 CORRECT
illegal-05 CORRECT
illegal-06 CORRECT
illegal-07 CORRECT
illegal-08 CORRECT
illegal-09 CORRECT
illegal-10 CORRECT
```

update ABOUT file

```
/*
 * Skeleton code for your Lexer, provided by Emma Norling
 *
 * Please note that this code is far from complete.
 * It needs to be extended and the documentation updated to reflect your changes
.
 *
 */
lexer grammar DecafLexer;

// These rules match
// all of the reserved words for Decaf (case sensitive)
CLASS : ’class’;
BOOLEAN : ’boolean’;
BREAK : ’break’;
CALLOUT : ’callout’;
CONTINUE : ’continue’;
ELSE : ’else’;
FALSE : ’false’;
FOR : ’for’;
IF : ’if’;
RETURN : ’return’;
TRUE : ’true’;
VOID : ’void’;
INT : ’int’;

// these rules capture numerical operators
// not yet imp

// These two rules deal with characters that have special meaning in Decaf - aga
in, what others?
LCURLY : ’{’;
RCURLY : ’}’;
```

} lots more needed.

```
// This says an identifier is a sequence of one or more alphabetic characters
// or beginning with an underscore. can also contain digits.
// Decaf is a little more sophisticated than this.
ID :
  (’a’..’z’ | ’A’..’Z’ | ’_’ ) (’a’..’z’ | ’A’..’Z’ | ’_’ | ’0’ .. ’9’)*;

// This rule simply ignores (skips) any space, tab or newline characters
WS_ : (’ ’ | ’\t’ | ’\n’ | SL_COMMENT | ’\f’ )+ -> skip;

// And this rule ignores comments (everything from a ’//’ to the end of the line
)
SL_COMMENT : ’//’ (~’\n’)* ’\n’ -> skip;

// These two rules completely describe characters and strings, and make use of t
he ESC and NOTESC fragments described below
// This rule says a character is contained within single quotes, and is a single
 instance of either an ESC, or any
// character other than a single quote, a single backslash, a single double quot
e, plus the 2-character sequences
// of \", \’, \\, \t and \n
// Character literals are composed of a <char> in single quotes
CHAR : ’\’’ (ESC|NOTESC) ’\’’;

// This rule says a string is contained within double quotes, and is one or more
 instances of either an ESC, a NOTESC
// character or any other than a double quote.
// String Literals are composed of <char>s enclosed in double quotes
STRING : ’"’ (ESC|NOTESC)* ’"’;
```

```
// this rule says an integer is either one or no negative signs followed by
or more integer
INTLIT : ’-’?[0-9]+;
```

| HEX ;

```
// this rule says a hex number is an integer from 0-9 followed by either ca
// of a-f
```

fragment

```
HEX : ’0’’x’([0-9]|[a-f]|[A-F])+;

// A rule that is marked as a fragment will NOT have a token created for it
anything matching the rules above
// will create a token in the output, but something matching the ESC rule be
will only be used locally in the scope
// of this file. Any rule that should not generate an output token should be
ceded by the fragment keyword.

// ESC matches either a pair of characters representing a newline (’\’ and
or a pair of characters representing
// a double quote (’\’ and ’"’). HINT: there are many other characters that
ld be escaped - think of how you need
// to write them in strings in languages like Java.
fragment
ESC :  ’\\’ (’"’|’n’|’t’|’\’’|’\\’);

// NOTESC matches single quotes, double quotes, backslash, double backslash
s well
// the escape character for two character sequences such as newline, new ta
 comment
fragment
NOTESC : ~(’"’|’\n’|’\t’|’\’’|’\\’);
```

Make sure you put the correct labels in Main.java (otherwise automated testing fails).