```
Summary file for decaf-Muzual
name: Miles Schofield
Student ID: 14004899
Lab Time: Thursday 14:00
********************
   PARSER TESTING
*** Legal tests ***
15/15 correct parser test cases
*** Legal tests with debug ***
*** Illegal tests ***
illegal-01 generated expected error
illegal-02 generated expected error
illegal-03 generated error:
line 2:8 missing INT_LITERAL at ']'
illegal-04 generated error:
line 3:10 mismatched input '=' expecting {';', ','}
illegal-05 generated expected error
illegal-06 generated expected error
illegal-07 generated expected error
illegal-08 generated error:
line 4:14 mismatched input '=' expecting {';', '+', '-', '%', '/', '*', '>', '<'
, '>=', '<=', '==', '!=', '&&', '||'}
illegal-09 generated expected error
illegal-10 generated expected error
illegal-11 generated expected error
illegal-12 generated error:
line 7:10 extraneous input 'b' expecting {';', ','}
illegal-13 generated error:
line 2:2 mismatched input 'main' expecting {'boolean', 'void', 'int', '}'}
illegal-14 generated error:
line 2:11 extraneous input 'a' expecting {'boolean', 'int', ')'}
illegal-15 generated error:
line 3:8 extraneous input 'int' expecting {'callout', '(', ')', '-', '!', ID, CH
AR_LITERAL, BOOL_LITERAL, INT_LITERAL}
illegal-16 generated error:
line 3:8 mismatched input ')' expecting {'callout', '(', '-', '!', ID, CHAR_LITE
RAL, BOOL_LITERAL, INT_LITERAL}
illegal-17 generated error:
line 3:4 mismatched input '0xcafe' expecting {'boolean', 'break', 'callout', 'co
ntinue', 'for', 'if', 'return', 'int', '{', '}', ID}
line 5:0 extraneous input '}' expecting <EOF>
illegal-18 generated error:
line 3:12 mismatched input '5' expecting STRING_LITERAL
illegal-19 generated error:
line 5:11 no viable alternative at input 'forpari'
line 5:16 mismatched input ',' expecting {';', '+', '-', '%', '/', '*', '>', '<'
, '>=', '<=', '==', '!=', '&&', '||'}
illegal-20 generated error:
line 5:11 no viable alternative at input 'forpari'
line 5:16 mismatched input ',' expecting {';', '+', '-', '%', '/', '*', '>', '<'
, '>=', '<=', '==', '!=', '&&', '||'}
   20 /20 correctly detected parser errors
```

```
/*
 * A skeleton for your parser, provided by Emma Norling.
 * Extended by Miles Schofield
 *
 *
 * The parser holds the grammar rules for the compiler utilising the tokens
 * created by the Lexer component
 *
 */
parser grammar DecafParser;
options { tokenVocab = DecafLexer; }

// Rule to show that a program is defined by class, ID, then any amount of field
 or method declarations
// within curly braces. The program is ended by the END OF FILE which is automat
ically provided by
// ANTLR and does not exist in the Lexer.

program: CLASS ID LCURLY (field_decl)* (method_decl)* RCURLY EOF;

field_decl: type field_name(COMMA field_name)* SEMICOLON ;
field_name: (ID|ID LSQRBRK INT_LITERAL RSQRBRK);

method_decl: (type | VOID) ID LPAREN ((type ID)(COMMA type ID)*)? RPAREN  block;

block: LCURLY var_decl* statement*  RCURLY;

var_decl: type ID(COMMA ID)* SEMICOLON;

type: (INT | BOOLEAN);

statement: location assign_op expr SEMICOLON
                | method_call SEMICOLON
                | IF LPAREN expr RPAREN block (ELSE block)?
                | FOR ID ASSIGNOPERATOR expr COMMA expr block
                | RETURN (expr)? SEMICOLON
                | BREAK SEMICOLON
                | CONTINUE SEMICOLON
                | block;


assign_op: ASSIGNOPERATOR
                | ARITHPLUS ASSIGNOPERATOR
                | ARITHMINUS ASSIGNOPERATOR;

method_call: method_name LPAREN (expr(COMMA expr)*)? RPAREN
                | CALLOUT LPAREN STRING_LITERAL (COMMA callout_arg(COMMA callout
_arg)*)? RPAREN;

method_name: ID;
location: ID
                | ID LSQRBRK expr RSQRBRK;

expr: location
                | method_call
                | literal
                | expr bin_op expr
                | ARITHMINUS expr
                | EXCLMRK expr
                | LPAREN expr RPAREN;

callout_arg: (expr | STRING_LITERAL);
```

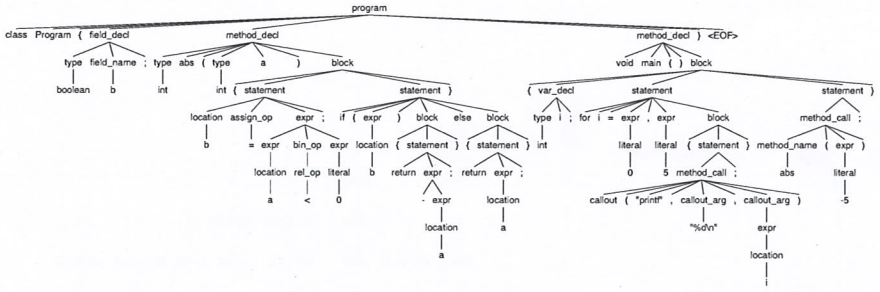


```
bin_op: (arith_op | rel_op | eq_op | cond_op);

arith_op: (ARITHPLUS | ARITHMINUS | ARITHMULT | ARITHDIV | ARITHMOD);

rel_op: (GRTTHAN | LESTHAN | GRTEQUAL | LESEQUAL);

eq_op: (EQUALOP | NOTEQUAL);

cond_op: (AND | OR);

literal: (INT_LITERAL | CHAR_LITERAL | BOOL_LITERAL);
```

Good work. Need to do something about the order of operations.

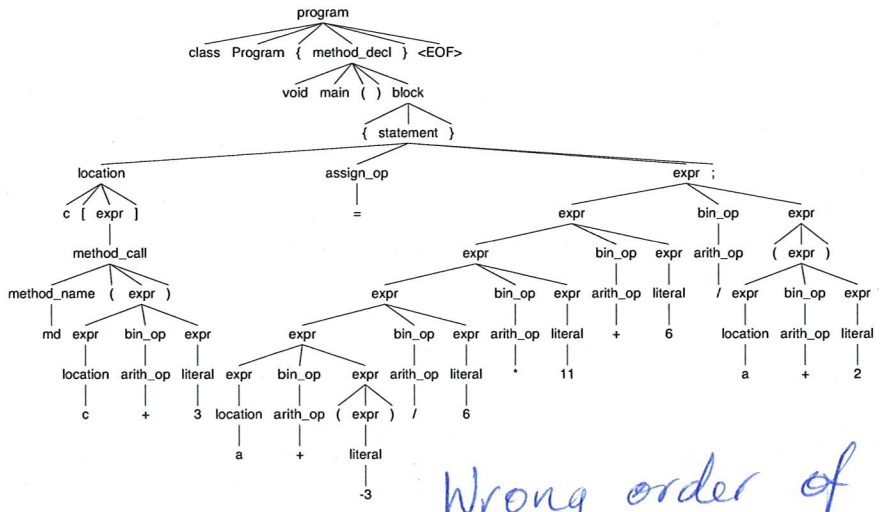
01

```
class Program {
  boolean b;

  int abs(int a) {
    b = a < 0;

    if (b) {
      return -a;
    }
    else {
      return a;
    }
  }

  void main() {
   int i;

   for i = 0, 5 {
      callout("printf", "%d\n", i);
   }
        abs(-5);
  }
}
```



02

```
class Program {
  void main() {
    c[md(c+3)] = a + (-3) / 6 * 11 + 6 / (a + 2);
  }
}
```



Wrong order of operations!