CMSC 430 Project 2

The second project involves modifying the syntactic analyzer for the attached compiler by adding to the existing grammar. The full grammar of the language is shown below. The highlighted portions of the grammar show what you must either modify or add to the existing grammar.

```
function:
      function_header {variable} body

    function_header:
      FUNCTION IDENTIFIER [parameters] RETURNS type ;

    variable:
      IDENTIFIER : type IS statement ; |
      IDENTIFIER : LIST OF type IS list ;

    list:
        ( expression {, expression} )

    parameters:
      parameter {, parameter}

    parameter:
      IDENTIFIER : type

    type:
      INTEGER | REAL | CHARACTER

    body:
      BEGIN statement END ;

    statement:
      expression ; |
      WHEN condition , expression : expression ; |
      SWITCH expression IS {case} OTHERS ARROW statement ENDSWITCH ; |
      IF condition THEN statement {ELSIF condition THEN statement}
          ELSE statement ENDIF ; |
      FOLD direction operator list_choice ENDFOLD ;

    case:
      CASE INT_LITERAL ARROW statement

    direction:
        LEFT | RIGHT

    operator:
      ADDOP | MULOP

    list_choice:
        list |
        IDENTIFIER

    condition:
        expression RELOP expression |
```

```
        condition logical_operator condition |
        ( condition ) |
        NOTOP condition

logical_operator:
    ANDOP | OROP
expression:
  ( expression ) |
  expression arithmetic_operator expression |
  NEGOP expression |
  INT_LITERAL | REAL_LITERAL | CHAR_LITERAL |
  IDENTIFIER ( expression ) |
  IDENTIFIER

arithmetic_operator: ADDOP | MULOP | MODOP | EXPOP
```

In the above grammar, the red symbols are nonterminals, the blue symbols are terminals and the black punctuation are EBNF metasymbols. The braces denote repetition 0 or more times and the brackets denote optional.

You must rewrite the grammar to eliminate the EBNF brace and bracket metasymbols and to incorporate the significance of parentheses, operator precedence and associativity for all operators. The precedence and associativity rules are as follows:

- Among binary arithmetic operators the exponentiation operator has highest precedence followed by the multiplying operators and then the adding operators. But the unary negation operator ~ has higher precedence that all of the binary operators.
- All relational operators have the same precedence.
- Among the binary logical operators, `&` has higher precedence than `|`. But the unary logical operator `!` has higher precedence than either of the binary logical operators.
- All binary operators except the exponentiation operator are left associative.

The directives to specify precedence and associativity, such as `%prec` and `%left`, may not be used.

Your parser should be able to correctly parse any syntactically correct program without any problem.

You must modify the syntactic analyzer to detect and recover from additional syntax errors using the semicolon as the synchronization token. To accomplish detecting additional errors an error production must be added to the function body, to the variable declaration and to the when clause.

Your bison input file should not produce any shift/reduce or reduce/reduce errors. Eliminating them can be difficult so the best strategy is not introduce any. That is best achieved by making small incremental additions to the grammar and ensuring that no addition introduces any such errors.

An example of compilation listing output containing syntax errors is shown below:

```
   1  -- Multiple errors
   2
   3  function main a integer returns real;
Syntax Error, Unexpected INTEGER, expecting ':'
   4      b: integer is * 2;
Syntax Error, Unexpected MULOP
   5      c: real is 6.0;
   6  begin
   7      if a > c then
   8          b + / .4;
Syntax Error, Unexpected MULOP
   9      else
  10          case b is
  11              when => 2;
Syntax Error, Unexpected ARROW, expecting INT_LITERAL
  12              when 2 => c;
  13          endcase;
Syntax Error, Unexpected ENDCASE, expecting OTHERS or WHEN
  14      endif;
  15  end;

Lexical Errors 0
Syntax Errors 5
Semantic Errors 0
```

You are to submit two files.

- The first is a `.zip` file that contains all the source code for the project. The `.zip` file should contain the flex input file, which should be a `.l` file, the bison file, which should be a `.y` file, all `.cc` and `.h` files and a `makefile` that builds the project.
- The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:
    a. A discussion of how you approached the project
    b. A test plan that includes test cases that you have created indicating what aspects of the program each one is testing and a screen shot of your compiler run on that test case
    c. A discussion of lessons learned from the project and any improvements that could be made