Project 3

Syntax Analyzer (Parser)

Due: December 11, 2020.

In this project, you will build a parser for MYC++ language. You may (and should) use a modified version of the lexer (or scanner) from Project 2. The parser will receive tokens generated by the scanner, and the parser should determine if tokens represent a syntactically valid program. A program in MYC++ has the following form:

PROGRAM *declarations* **BEGIN** *body* **END;**

There can be any number of variable declarations, and they have the following format:

type variable;

Where *type* can be **INTEGER**, **REAL**, or **STRING**, and the *variable* has to follow the format described in Project 2.

Both parts, `declarations` and `body`, can have any number of statements (separated by semicolons). In `body`, there are three types of statements: assignment, write, and expression:

variable := *expression;*

WRITE *expression;*

expression;

expressions can have one of the following forms:

```
operand binary_op operand
operand
```

`binary_op` could be any of the following operators: `+`, `-`, `*`, `/`, and `**`. An `operand` can be either a variable or a number. Operands must have the same data type. For example, `5 + 5.5` should result in a syntax error. Operations can be performed on integer and real numbers. Operations cannot be performed on strings. String variables can be declared, and strings can be assigned to string variables or printed on the screen using WRITE statements.

Operations are prioritized as follows:

- Parentheses
- `**`

- * and /
- + and −

All operations have left to right associativity (except **, which has right to left associativity).

The output of your parser must be either:

(1) A message indicating **no syntax errors** in addition to the output of the **write** statements.

(2) An error (exception). For the following errors, the line number where the error occurs and a specific error message should be displayed (Example are provided below):

- Division by zero
- The use of an undeclared identifier
- The operands of an expression have different data types
- Redefinition of a variable

Example 1

Input:

```
PROGRAM
     INTEGER a;
     REAL b;
BEGIN
       b := 2.3;
       b := b * 5.0;
       a := 10;
       a := a *  (a * 10);
       WRITE a;
       WRITE b;
END;
```

Output:

```
       1000
       11.5

       --- No syntax errors ---
```

Example 2

Input:

```
PROGRAM
```

```
     INTEGER a;
     REAL b;
BEGIN
      b := 2.3;
      a := 10;
      WRITE a;
      WRITE b;
      WRITE c;
END;
```

Output:

```
     Uncaught exception: Stdlib.Parsing.Parse_error
     10
     2.3
     **Error** 9: Use of undeclared identifier c
     Fatal error: exception Stdlib.Parsing.Parse_error
```

Example 3

Input:

```
PROGRAM
      REAL a;
BEGIN
      a  := 1.5;
      WRITE 10 * a;
END;
```

Output:

```
     Uncaught exception: Stdlib.Parsing.Parse_error
     **Error** 5: Multiplication of different data types
     Fatal error: exception Stdlib.Parsing.Parse_error
```

The example above should also apply to +, -, /, and **.

Example 4 (no output)

Input:

```
PROGRAM
      REAL a;
BEGIN
      a  := 1.5;
      10 * 10;
```

```
        100 * 100;
        1.0 * a;
END;
```

Output:

```
        --- No syntax errors ---
```

Example 5

Input:

```
PROGRAM
        REAL a;
        REAL a;
BEGIN
        a  := 1.5;
        WRITE 10 * a;
END;
```

Output:

```
Uncaught exception: Stdlib.Parsing.Parse_error
**Error** 3: Redefinition of 'a'
Fatal error: exception Stdlib.Parsing.Parse_error
```

Example 6

Input:

```
PROGRAM
        REAL a;
BEGIN
        a  := 1;
        WRITE 10.0 * a;
END;
```

Output:

```
Uncaught exception: Stdlib.Parsing.Parse_error
**Error** 4: The assignment of different data types
Fatal error: exception Stdlib.Parsing.Parse_error
```

Example 7

Input:

```
PROGRAM
       STRING s;
BEGIN
       s := "Hello World!";
       WRITE s;
END;
```

Output:

```
     "Hello World!"
     --- No syntax errors ---
```

Submit three files `lexer.mll`, `parser.mly`, and `main.ml`.