Sabancı University
Faculty of Engineering and Natural Sciences

CS305 Programming Languages

Homework 3

Due: March 26, 2020 - Thursday @ 23:55

1 Introduction

In this homework you will implement a small scale semantic analyzer and a type checker for BMO language. Your tool will check if a given BMO program has problems with respect to some type compatibility rules, which will be explained below. In this homework, we will try only the inputs that are grammatically correct with respect to the grammar given in the bison file attached to this homework. However, these BMO examples may or may not have the type compatibility problems that you are supposed to check in this homework. In the case that there is a type compatibility issue, you can assume that there will be only one such problem in the input. Therefore, when you detect an error, you can simply terminate after producing the required error message.

```
int vector digits, digits2 = [1,2,3,4,5,6,7,8,9,0];
int matrixM = 3;
int matrix result = [1,2;3,4;5,6;7,8;9,0];
real realNum = 4.0E-12;
int matrix resultT = transpose([1,2;3,4;5,6;7,8;9,0]);
if(digits == digits2)
    realNum = 5.0E-12;
endif
```

Below is the detailed syntactic features of the BMO language. **The grammar of BMO in HW3 is slightly different than HW2 (in HW3 we only allow constants to be used in expressions).** Note that, a scanner and a parser for BMO are provided as an attachment to this homework. You can use these scanner/parser, or you can implement a scanner/parser pair of your own.

1. A BMO program consists of a non–empty sequence of statements.
2. Each statement in our BMO can be a declaration, or an assignment statement or an `if` statement.
3. A declaration is given by the type, followed by a comma separated list of variable names, followed by an assignment sign, followed by an expression, and followed by a semicolon.

4. The type can be one of the following: `int`, `int vector`, `int matrix`, `real`, `real vector`, `real matrix`.

5. An assignment statement given by a variable name, and an equality sign (i.e. the character =), and then an expression, finally followed by a semicolon. E.g.

$$a = 1+2;$$

6. An if statement starts with keyword `if`. After the keyword `if`, a boolean expression resides betweeen (and). This is followed by the body of the if statement, which is a non–empty sequence of statements. Finally, the keyword `endif` is given.

7. An expression can be a number (integer or real), or a vector literal, or a matrix literal, or two expressions being applied to one of the operators $*$,+,−,/,.$*$ or a transpose expression.

8. A vector literal is given by a `[`, followed by a value row, followed by a `]`.

9. A value row is a comma separated list of values, where a value is either a number (integer or real), or a variable name.

10. A matrix literal is given by a `[`, followed by a semicolon separated list of value rows, followed by a `]`. There has to be at least two rows in a matrix literal.

11. A transpose expression is given by the keyword `transpose`, followed by a `(`, followed by an expression, followed by a `)`.

12. A boolean expression is either a simple comparison or two boolean expressions connected with a boolean operator, where we only have `&&` and `||` as the boolean operators.

13. A simple comparison must have exactly two variables separated with one of the comparison operators `>,<,>=,>=,!=,==`.

2 Errors

There are two different type of errors that your tool will check. The errors are related to the suitability of the dimensions of the matrix/vector/scalar values used in the expressions. They are quite intuitive as these rules are the usual rules that we know from matrix operations. We explain these two errors and the error messages that you need to produce below.

1. A matrix literal consists of a sequence of rows, rows are separated by semicolons. In a matrix literal, each row must have the same number of columns. If a matrix literal has rows with different number of columns, then you must produce an error message. The format of the error message is as follows:

`ERROR 1:` *lineNo* `inconsistent matrix size`

In this error message, "*lineNo*" is the line number on which this matrix literal is seen. Note that, the line number of the opening left bracket of the matrix literal is considered to be the line number of a matrix literal. Please see Section 3 for a concrete example of this error message.

2. In an expression, we may have scalar values, vector values, and matrix values as the operands. An expression is "ill formed" if the size of the operands are not appropriate for the operator of the expression. For example, a 2×3 matrix cannot be multiplied by a 5×4 matrix, or a 4×1 vector cannot be added with a scalar value, etc. Here, we consider the usual matrix, vector, scalar operations with the usual size compatibility rules. Your parser will analyze the expressions on the right hand side of declaration and assignment statements. For each binary expression of the form

   operand_1 operator operand_2

   your parser will infer the dimension of operand_1 and operand_2, and will check if the dimensions of operand_1 and operand_2 are suitable for the operand in this expression.

   A vector literal with n values is considered to have dimension $1 \times n$. A matrix literal with m rows where each row having n columns is considered to have dimension $m \times n$.

   Note that, you will have to infer the dimensions of the subexpressions. For example consider the following expression consisting of three operands multiplied:

   ```
   [1,2,3;4,5,6] * [7;8;9] * [10,11]
   ```

   The first operand has dimension 2×3, the second operand has dimension 3×1, and the last operand has dimension 1×2.

   Since the multiplication is left associative, first the following subexpression will be considered:

   ```
   [1,2,3;4,5,6] * [7;8;9]
   ```

   The operands have appropriate dimensions for this multiplication. Therefore no error will be produced. Furthermore, the dimension of the resulting expression will be inferred as 2×1. And this dimension is suitable for multiplying from left with the last operand which has dimension 1×2. The dimension of the overall expression will be inferred as 2×2.

   When the parser sees that there is a dimension mismatch for a binary expression, the following error message must be produced:

   `ERROR 2:` *lineNo* `dimension mismatch`

   In this error message, "*lineNo*" is the line number of the operator of the binary expression.

3 Example Outputs

1. INPUT:

```
int vector digits = [1,2,3,4,5,6,7,8,9,0];
int matrixm = 3;
int a = 5;
int matrix result = [1,2;3,4;5,6;7,8;9,0];
int matrix resultT = transpose([1,2;3,4;5,6;7,8;9,0]);
int matrix myMat = [1,2,3;3,4;6,1;7,8;9,0,1];
if(matrixm >= a)
    matrixm = 4;
endif
```

```
OUTPUT:
ERROR 1: 6 inconsistent matrix size
```

2. INPUT:

```
int a = 3;
int b = 4;
real realNum = 4.0E-12;
int matrix mult = [1,2;3,4;5,6;7,8;9,0] * [1,2,3;3,4,5;5,6,7;7,8,9;9,0,1];
if(a >= b)
    a = 1;
endif
```

```
OUTPUT:
ERROR 2: 4 dimension mismatch
```

3. INPUT:

```
int abc = 6;
int matrix result = [1,2;5,6;7,8;9,0];
int matrix resultT = transpose([1,2;2,3]);
int matrix calc = [1,2;3,4] + [1,2,3;3,4,5;5,6,7];
abc = 8;
```

```
OUTPUT:
ERROR 2: 4 dimension mismatch
```

4. INPUT:

```
int vector digits, digits2 = [1,2,3,4,5,6,7,8,9,0];
int matrixM = 3;
int matrix result = [1,2;3,4;5,6;7,8;9,0];
real realNum = 4.0E-12;
int matrix resultT = transpose([1,2;3,4;5,6;7,8]);

OUTPUT:
OK
```

4 How to Submit

Submit your Bison file named as username-hw3.y. Before using the Flex file shared with you, do not forget to change the include part correctly as in second homework. You are also going to submit it along with your Bison file. We will compile your files by using the following commands:

```
flex username-hw3.flx
bison -d username-hw3.y
gcc -o username-hw3 lex.yy.c username-hw3.tab.c -lfl
```

So, make sure that these three commands are enough to produce the executable parser. If we assume that there is a text file named test20, we will try out your parser by using the following command line:

```
./username-hw3 < test20
```

If test20 includes a grammatically correct BMO program then your output should be OK otherwise, your output should be the error information as explained above.

5 Notes

- **Important**: Name your files as you are told and **don't zip them.** [-10 points otherwise]
- **Important**: Make sure you include the right file in your scanner and make sure you can compile your parser using the commands given in the Section 4. If we are not able to compile your code with those commands your **grade will be zero for this homework.**

- Make sure your output is exactly as it is supposed to be.
- No homework will be accepted if it is not submitted using SUCourse.
- No late submission will be accepted.
- Since the grading will be done automatically on the flow.sabanciuniv.edu machine, we strongly encourage you to at least test your code on flow before submitting it.