Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
06/03/2008

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other:.....
Instructor: ○ Prof. Breveglieri ○ Prof. Crespi ○ Prof Morzenti

The laboratory question must be answered taking into account the implementation of the `Acse` compiler given with the exam text.

Modify the specification of the lexical analyzer (`flex` input) and the syntactic analyzer (`bison` input) and any other source file required to extend the `Lance` language with the ability to handle *the module operator and the power operator* with the following syntax (the same as C programming language) :

```
int x,y,z;

y = 3;
z = (3 + 8) % y;
write(z);

x = z ** 5;
write(x);

y = 3 ** 4;

y = z ** z;
```

This code snippet prints out "2" (i.e., 11 mod 3) and "32" (i.e., 2^5) when compiled and run.

The solution needs to comply to the following specifications:

- The module operator must have a priority *strictly* enclosed between addition and multiplication.
- The power operator must have the *least* priority of all binary operators.
- The student may specify a proper associativity for the operators.

[1] Time 45'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

An optimal solution should be able not to generate any code if the operations have only immediate values as operands, but to directly compute the value and propagate it.

Your modifications have to allow the `Acse` compiler to both correctly analyze the syntactical correctness of the aforementioned constructs and to generate a correct translation in the `Mace` assembly language.

We recall that the meaning of the module operator mod is "take the remainder of the integer division between the two operands". You might find useful the following function:

```
int gen_load_immediate(t_program_infos *program, int immediate);
```

1. Define the tokens and the Acse.lex and Acse.y declarations needed to achieve the required functionality. (3 points)

2. Define the syntactic rules needed (or the modifications to the existing ones) to achieve the required functionality (8 points).

3. Define the semantic actions (or the modifications to the existing ones) needed to handle one of the two operators at your choice (13 points for the module operator, 19 for the power).

4. **Bonus** Write the syntactic tree for the following code snippet:

```
int a = 10;
int b = 20;
a = a ** b % c ;
```

using the Bison grammar contained in Acse.y, considering the modifications made in the previous points and *starting from the nonterminal "statements"*