Formal Languages and Compilers
Proff. Breveglieri, Morzenti
Written exam[1]: laboratory question
02/07/2014

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other: ...
Instructor: ○ Prof. Breveglieri ○ 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 analyser (`flex` input) and the syntactic analyser (`bison` input) and any other source file required to extend the `Lance` language with the ability to process the ternary operator *splice* that, given two expressions e_1 ed e_2 and an expression with constant value e_c, is syntactically defined as e_1 \$ e_2 @ e_c. The resulting *expression* is obtained by combining the e_c most significant bits of e_1 with the $32 - e_c$ less significant bits of e_2. The implentation has to check that the third operand

```
int a, b, splice;

read(a);
read(b);

// a = 65536; b = 32768
// a = 00000000000000010000000000000000
// b = 00000000000000001000000000000000

splice = a $ b @ 16;

// splice = 98304
// splice = 00000000000000011000000000000000

write(splice);
```

Figura 1: Esempio

is a constant and that its value is between 0 and 32. If the value of e_c is greater that 32 then the splicing is realized by 32 bits.

[1]Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.y**). (3 points)
2. Define the syntactic rules or the modifications required to the existing ones. (4 points)
3. Define the semantic actions needed to implement the required functionality. (18 points)

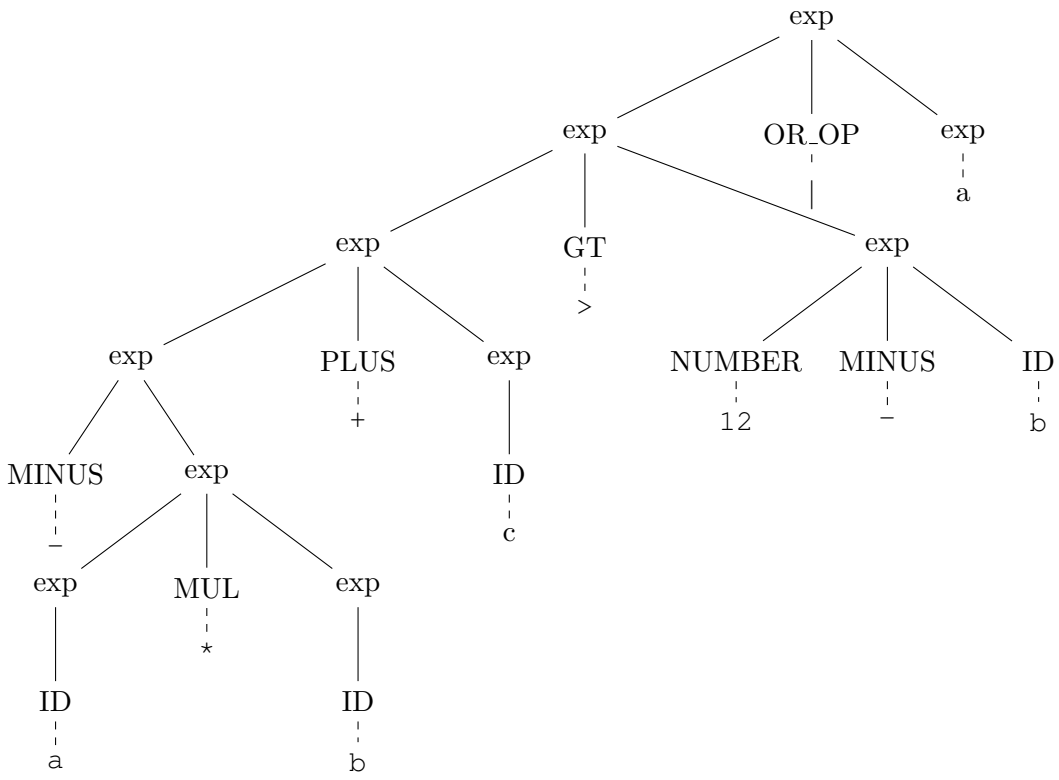
   The solution is in the attached patch.

4. Given the following `Lance` code snippet:

```
int a, b = 6, c;
read(a);
read(c);

write(-a * b + c > 12 -b | a);
```

write the syntactic tree of the expression which is the argument of function **write**, with the Bison grammar described in Acse.y and *by starting from* `exp` *nonterminal.* (5 points)



5. (**Bonus**) Describe the extension of the splice operator to support generic expressions defining e_c. The following snippet shows an example.

```
int a, b, c, splice;

read(a);
read(b);
read(c);

splice = a $ b @ (c+a-1);
```

Figura 2: Generalized splice