Formal Languages and Compilers Proff. Breveglieri and Morzenti Written exam[1]: laboratory question 21/02/2017

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 **count-when-into** construct.

An instance of the **count-when-into** construct is provided in the following code snippet:

```
int a,b;
count {
  when (3){
    write(2);
  },
  when (b-5){
    if(a+b==7){
     b=b+1;
    }
  },
  when (b<0){ }
} into a;
```

The **count-when-into** construct has the following semantics: the curly braces following the `count` keyword enclose a comma separated list of `when` statements which are executed in program order.

Each `when` statement executes the contents enclosed in the curly braces in program order if and only if the expression enclosed in the round braces is true (i.e. different from zero).

Finally, when the contents of the curly braces have been executed, the `count-when-into` statement stores the number of times a `when` statement has executed the content between his curly braces in the variable following the `into` keyword.

The `count-when-into` statements may be arbitrarily nested; the statements enclosed between the curly braces of the `when` statement can be any valid Lance statement.

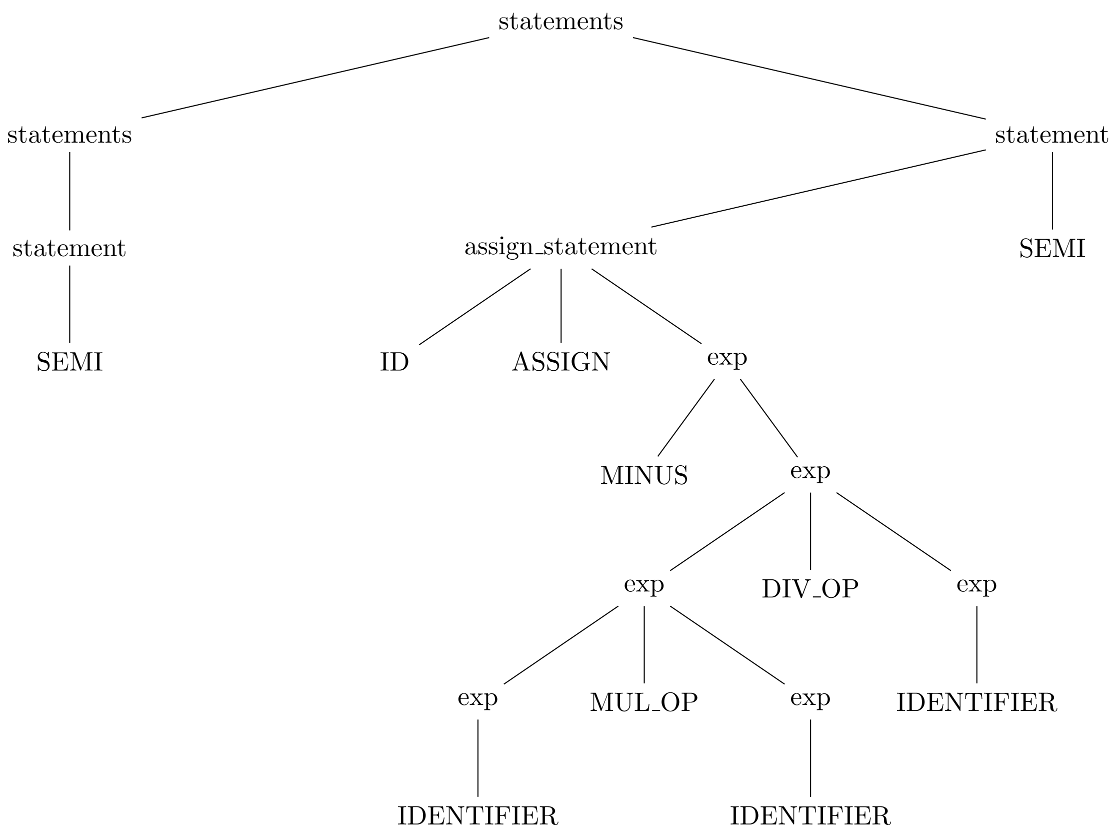
[1]Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.y**). (1 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. (20 points)
   The solution is in the attached patch.

4. Given the following `Lance` code snippet:

```
;a = -a * b / c;
```

write the syntactic tree generated during the parsing with the Bison grammar described in Acse.y starting from `statements` *nonterminal.* (5 points)



5. (**Bonus**) Describe how to modify your solution to implement an `all-when-into` construct, analogous in semantics to the `count-when-into` save for the fact that the variable following the `into` keyword is assigned to 1 only if all the `when` statements are run, and to 0 otherwise.