Formal Languages and Compilers
Proff. Breveglieri, Morzenti
Written exam[1]: laboratory question
04/02/2020

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Magistrale ○ 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 **`try-catch`** statement. This construct is composed of two parts: a block of code placed after the **`try`** keyword, and a list of **`catch`** blocks. The execution of the construct begins with the `try` block, inside this block, a **`throw`** statement may appear, which throws an exception. The execution of a `throw` statement outside a `try` block is undefined behaviour.

When an exception is thrown, the execution of the `try` block is interrupted, and control is transfered to the `catch` blocks. Each `catch` block is characterized by a numeric identifier, the parameter of the `throw` statement determines which catch blocks have to be executed. The argument of the `throw` statement and the identifier of the `catch` blocks are arbitrary expressions. An arbitrary number of catch blocks is allowed (including any). If no catch block matches the thrown value, the control flow proceeds from the end of the `try-catch` statement. If multiple catch blocks have the same identifier, all of them are executed, in the order in which they appear in the source code. The `try-catch` statement does not allow nesting.

The following code exemplifies its use.

```
int a;
read(a);
try {
        if (a == 1)
                throw 10;
        write(1);
        if (a == 2)
                throw 20;
        write(2);
} catch (a + 10) {
        write(10);
} catch (20) {
        write(20);
};
```

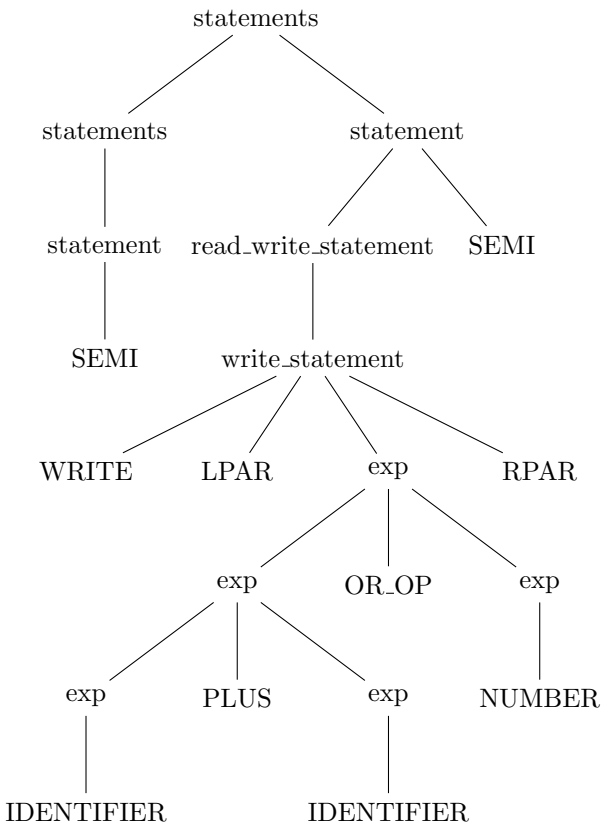
[1]Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.y**). (2 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. (19 points)
   The solution is in the attached patch.

4. Given the following `Lance` code snippet:

```
; write ( x + y | 1 );
```

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


statements
statements
statement
statement
read_write_statement
SEMI
SEMI
write_statement
WRITE
LPAR
exp
RPAR
exp
OR_OP
exp
exp
PLUS
exp
NUMBER
IDENTIFIER
IDENTIFIER

5. (**Bonus**) Describe how you would modify your implementation in order to make **`try-catch`** statements nestable.