Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
04/09/2018

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 **push-into** and **pop-from** instructions that realize, respectively, the insertion and the extraction of a value into and from a stack, implemented by means of an array. Both the operators have two parameters: the array identifier and either an expression defining the value that is added to the stack in the **push-into** instruction or an identifier of a variable in the **pop-from** instruction. Beside the previous instructions, two operators are introduced for managing a stack: **is-empty** and **is-full**, both followed by the array identifier, return a boolean value indicating if the stack is empty of full, respectively. All the instructions raise an error if the array has not been declared or the identifier refers to a scalar. Assume that only one array identifier per program occurs as argument of a **push-into** and a **pop-from** and that the bottom of the stack is the position 0. Define a reasonable semantics for the instructions in case of misuse (e.g., pop with no push, etc.).

```
int a[5];
int x, y, z;
...
push (x+1) into a;
push y*x into a;

pop z from a;
write(z);
pop z from a;
write(z);
if (is-empty a)
  write(1);
//stack empty, print 1.
else
  write(0);
```

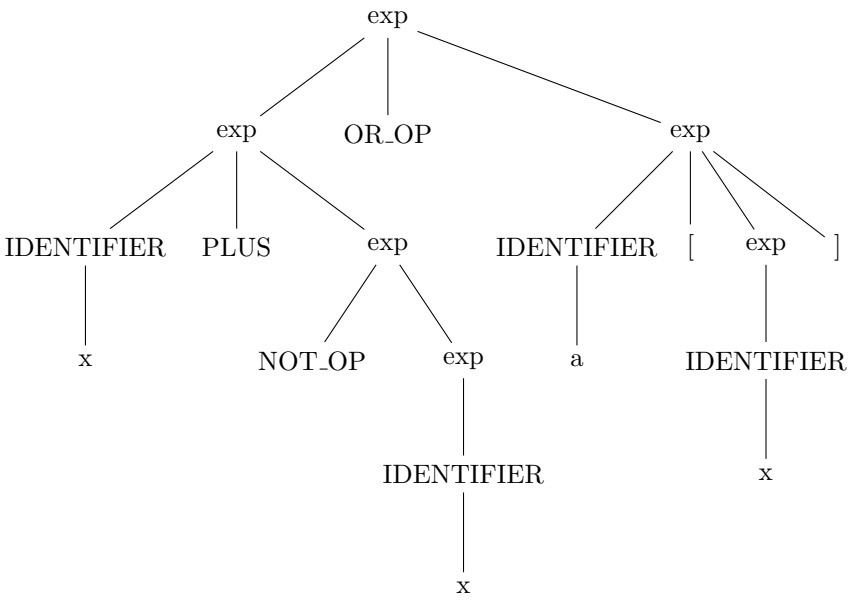
[1] Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

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:

```
x + !x | a[x]
```

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



5. (**Bonus**) Explain how to modify the solution in order to deal with multiple stacks in a program. In other words, it is possible to push into or pop from different arrays.

```
int a[5], b[10];
int x, y;
...
push x into a;
push y into b;
...
```