Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
21/09/2017

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 **iterateon** statement. Statement **iterateon** is intended to allow for accessing array locations without the use of the standard indexing $[i]$. An example of the statement is provided in the following LANCE program. One by one, the program assigns (through the assignment `c = a + b;` at line 10) the value `a[`i_a`]+b[`i_b`]` to `c[`i_c`]` and, after every assignment, it prints out the content of the entire array (line 14-15). Indexes i_a, i_b, i_c and i'_c are implicit array indexes that are part of the semantics of **iterateon** statement. The commented code on the right-hand side exemplifies the semantics of the statement related to the indexes.

```
1  int a[5], b[8], c[10];
2  ...
3  iterateon c{                    //i_c = 0
4    iterateon b{                  //i_b = 0
5      ...
6      iterateon a                 //i_a = 0
7        while(i){
8          c = a + b;              //if i_a ≥ 5 goto 17 else i_a = i_a + 1;
9                                  //if i_b ≥ 8 goto 19 else i_b = i_b + 1;
10                                 //if i_c ≥ 10 goto 21 else i_c = i_c + 1;
11
12         iterateon c             //i'_c = 0
13           while(i) write(c);    //if i'_c ≥ 10 goto 16 else i'_c = i'_c + 1;
14       }
15     ...
16   }
17   ...
18 }
19 ...
```

The token **iterateon** is followed by the identifier of an array and a code block bounding the scope of the statement, where the array variable can be accessed through its identifier (the

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

standard accessing by explicit indexing $[i]$ is still available). Every **iterateon** has its own implicit index (indexes i_a, i_b, i_c and i'_c in the previous code) that is initially set to 0 before the execution of the code block. Every time an array a is accessed through its variable identifier within the code block of an **iterateon** statement then the index of *inner-most* **iterateon** statement applied to array a is incremented. When the index value that is used to access array a, defined by the inner-most **iterateon** statement embracing the instruction accessing to a, is greater or equal to the array size, then the execution of the program is modified and the flow is set to the first instruction following the code block of the statement. The use of $a[]$ does not affect the value of the implicit indexes. Finally, if an array identifier is used outside an **iterateon** statement a compiling error is raised, as usual.

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 and, specifically, to deal with:
   - **iterateon** statements,
   - assignments `x = ...` within a **iterateon** statement, where `x` is an array,
   - the reading of an array position within a **iterateon** statement, according to the semantics explained before.

   (19 points)

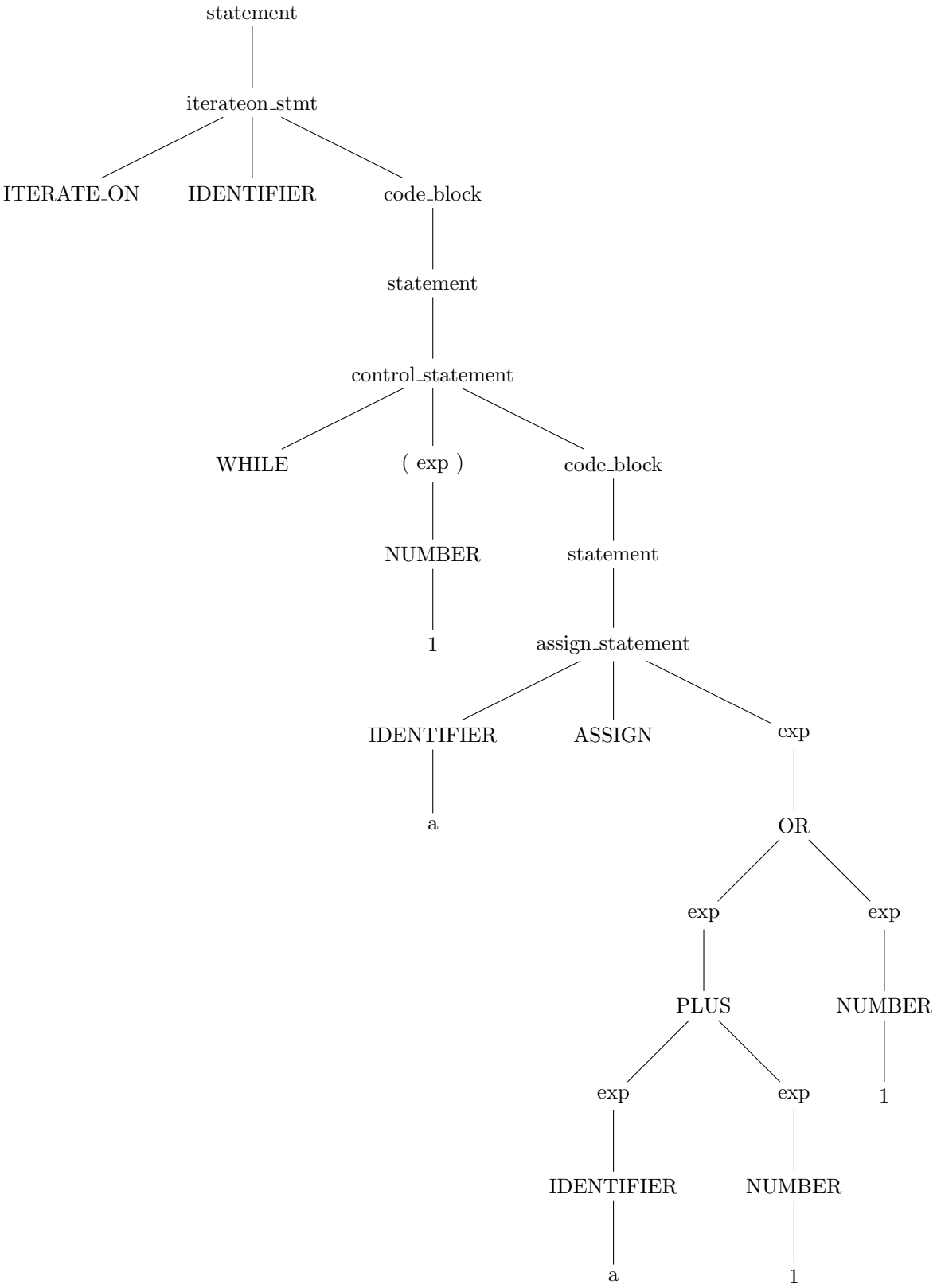
   The solution is in the attached patch.

4. Given the following `Lance` code snippet:

```
iterateon a
  while(1)
    a = a + 1 | 1;
```

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

5. (**Bonus**) Discuss how to modify your solution in order to extend **iterateon** with an initialization value and a step value. The former defines the initial value of the implicit index and the latter specifies the increment/decrement that is enforced when the index value is updated after every access to the array variable. An example of the syntax is provided hereafter.


statement
iterateon_stmt
ITERATE_ON
IDENTIFIER
code_block
statement
control_statement
WHILE
(exp)
code_block
NUMBER
statement
1
assign_statement
IDENTIFIER
ASSIGN
exp
a
OR
exp
exp
PLUS
NUMBER
exp
exp
1
IDENTIFIER
NUMBER
a
1

```
1 int a[5];
2 int init_value, step_value;
3 ...
4 iterateon a:init_value:step_value{
5   ...
6 }
```