Formal Languages and Compilers
Proff. Breveglieri, Morzenti
Written exam[1]: laboratory question
18/01/2018

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 new **foreach-every** construct.

The **foreach-every** construct iterates over an array without the need to explicitly handle the index of the array element. At each iteration, it runs the code block placed before the *every* keyword. On the first iteration, `elem` takes the value of `v[0]`, on the second one is updated to `v[1]`, on the third is updated to `v[2]` and so on for each element of the array.

Every `k` iterations the construct executes once the code block placed after the *do* keyword **instead of** the normal loop body. The `k` parameter must be an expression whose value is known at compile time. Moreover, the behaviour of the **foreach-every** construct enforces the first iteration to always run the loop body that has been defined before the *every* keyword.

The implementation must check: • the variable placed before the *in* keyword must be a scalar; • the variable placed after the *in* keyword must be an array; • the value of `k` must be positive and greater than 1; • the value of `k` must allow the code block placed after the *do* keyword to be executed at least once.

In the case one or more of the aforementioned conditions is not met, the implementation must raise an error. Whenever it is possible, these errors should be notified by the compiler.

An example is provided in the following.

```
int elem, v[10], a;
a = 0;

foreach elem in v {
  a = a + elem;
} every 2 do {
  write(a);
  a = elem;
};
```

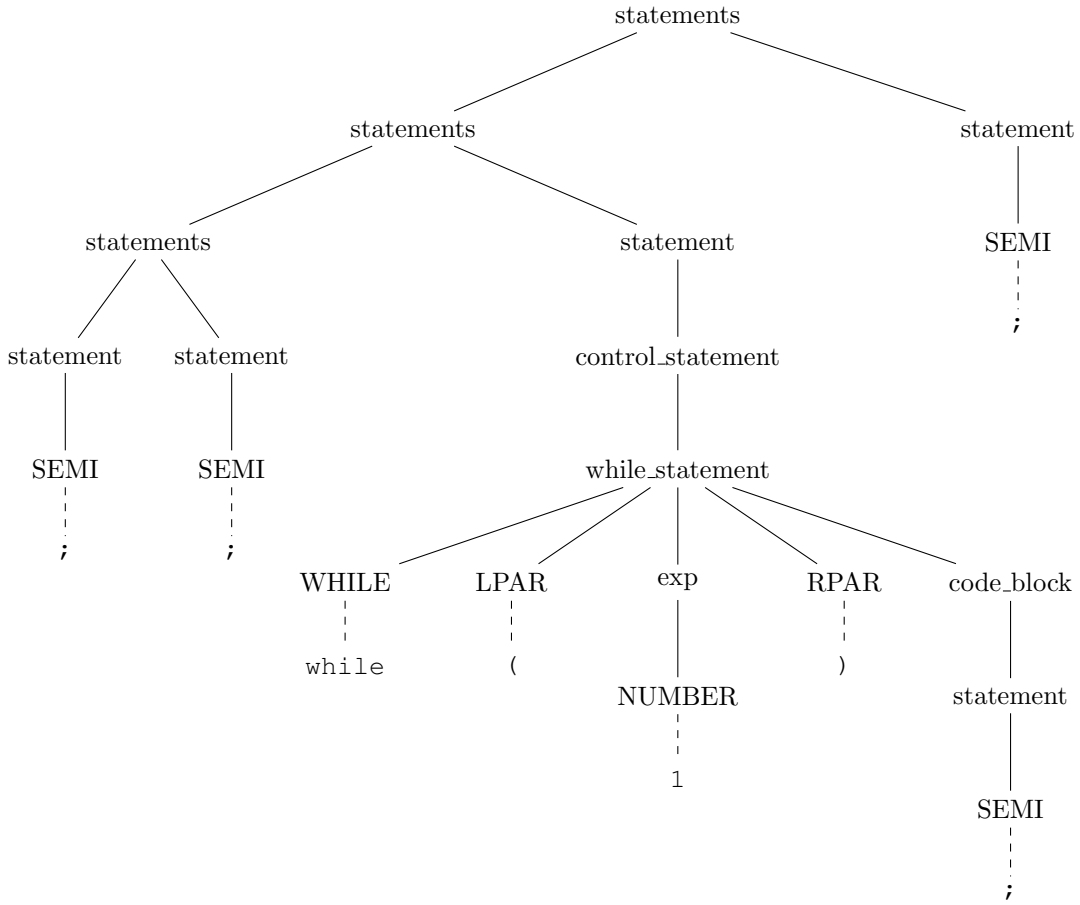
[1] Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

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:

```
;;while(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)



5. (**Bonus**) Describe how to modify your solution to change the behaviour of the aforementioned construct and allow the second code block to be executed **in addition to** the loop body once every `k` iterations.

在label_end后
再加个beq $3%k ==0 跳来
code_block2