Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
16/09/2010

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other:.....
Instructor: ○ Prof. Breveglieri ○ Prof. Crespi ○ 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 analyzer (`flex` input) and the syntactic analyzer (`bison` input) and any other source file required to extend the `Lance` language with the ability to handle the `after-do` construct:

```
1  int a;
2  int b;
3
4  a = 0;
5  b = 0;
6  after 3 do {
7    a=5;
8    b=7;
9  }
10 while(a<10) {
11   a=a+1;
12 }
13 b=1;
14 write(a);
15 write(b);
```

(a) after-do construct

```
1  int a;
2  int b;
3
4  a = 0;
5  b = 0;
6  while(a<10) {
7    a=a+1;
8  }
9  b=1;
10 a=5;
11 b=7;
12 write(a);
13 write(b);
```

(b) Equivalent code

With the `after-do` construct, the code specified in the code block after the `do` keyword is executed k statements after the end of the code block itself, instead of being executed in the position where it is written. Note that k is a positive immediate constant and that the number of statements is counted statically (it is referred to the number of statements as appearing in the source code of the program, not to the number of executed statements).

[1] Time 45'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

In the code sample of Figure 1(a), the instructions `a=5` and `b=7` will be executed after `b=1` and before `write(a)`, as it can be seen looking at the semantically equivalent code of Figure 1(b) .

Assume that in a program there can be at most one `after-do` construct.

Explicit any other assumption you made to implement the support for the `after-do` construct.

1. Define the tokens (and the related declarations in **Acse.lex** e **Acse.y**). (1 points)

   The solution can be found in the attached patch.

2. Define the syntactic rules or the modifications required to the existing ones. (4 points)

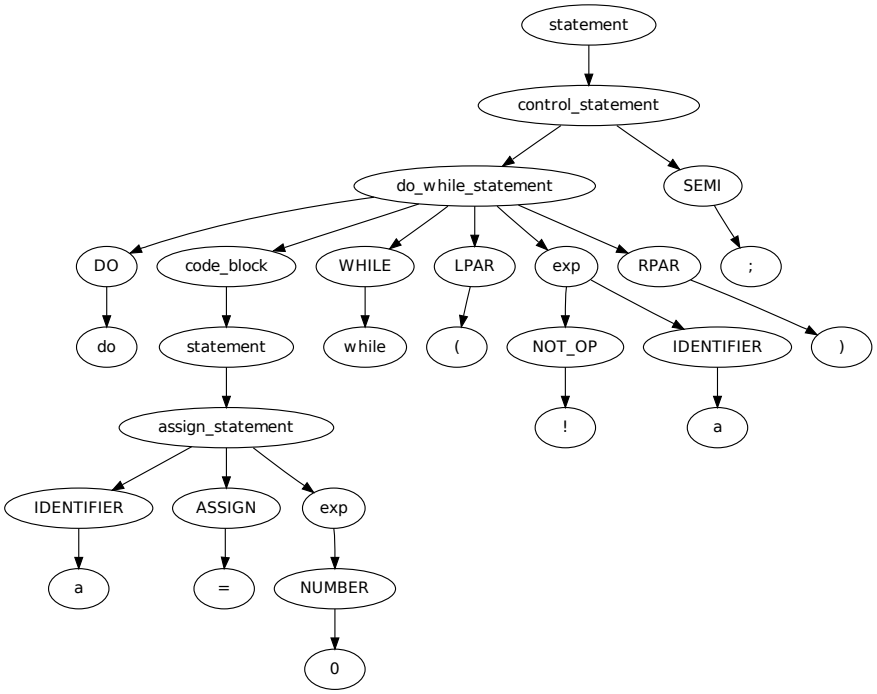
   The solution can be found in the attached patch.

3. Define the semantic actions needed to implement the required functionality. (20 points) The solution can be found in the attached patch.

4. Given the following code snipped:

   ```
   do
     a=0;
   while ( !a );
   ```

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



5. (Bonus) Describe how it would be possible to implement support for having multiple `after-do` constructs in a single program. (3 points)

   To have multiple `after-do` constructs, the labels they need should be stored into a dynamically linked list.

   Each element of the list is a struct containing two labels, and one counter. The counter marks the statement distance from the the `after-do` construct.

   When the right distance (for each instance of the construct) has been reached, the right labels and jumps are generated, as in the single-construct version.

   Beware that, in order for the semantics of the construct to be unchanged, statements generated by the construct itself should not be accounted when computing the distance.