Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
31/01/2011

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 analyser (`flex` input) and the syntactic analyser (`bison` input) and any other source file required to extend the `Lance` language with the ability to handle vector xor operations.

```
1 int a[10];
2 int b[10];
3 int c[10];
4
5 vec_xor(c, a, b);
```

Figure 1: Vector operation: computes the vector xor $\overline{c} = \overline{a} \oplus \overline{b}$ using `vec_xor`.

The vector operation is identified by the `vec_xor` keyword:

Vector xor `vec_xor(c, a, b)` computes the xor element by element between `a` and `b` saving the result in `c`. The generic element `c[i]` will be equal to `a[i] ^ b[i]`

The new operation works upon vectors of the same size. In the examples of Figure 1 the three operands of the vector operations all contain 10 elements. If the vectors do not have the same length, the translation cannot be performed and a compile-time error must be issued.

Explicit any other assumption you made to implement the support for vector operations.

[1] Time 45'. 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** e **Acse.y**). (3 points)

   The solution is in the attached patch.

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

   The solution is in the attached patch.

3. Define the semantic actions needed to implement the required functionality. (18 points)

   The solution is in the attached patch.

4. Given the code in Figure 2:

```
1 do{
2   b = a * b + c;
3 } while (a == 7);
```

Figure 2: `do-while` construct.

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

The solution is in Figure 3.

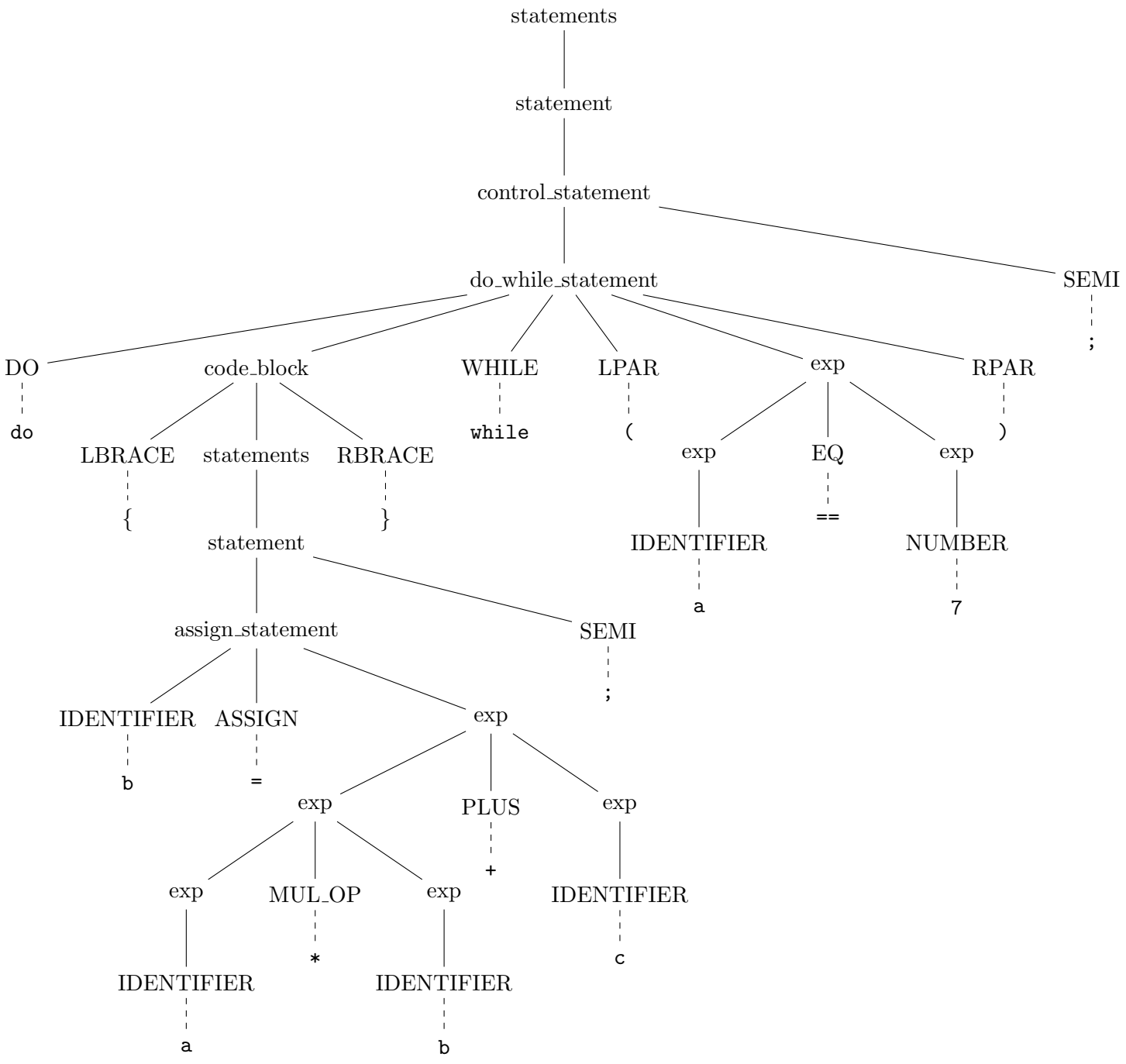

Figure 3: Syntactic tree of a `do-while` statement.

5. (Bonus) Extend the behaviour of the `vec_xor` operation in order to work with vectors of different sizes.

   In case the first operand is longer than the second, the vector xor operation compensates the missing cells through re-using the shortest vector from the beginning, i.e. obtains a second operand long enough through repeating it enough times.

   In case the first operand is shorter than the second, the vector xor operation ignores the extra cells of the second vector.

   In both cases, the result vector c must be long enough to contain the result, if this is not the case, a compile time error shall be risen.