Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
17/03/2012

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 perform the `shift` operation over **arrays**:

```
/* Example 1 */
a >> 2;
/* Example 2 */
a << x;
```

As shown in the code, an array-shift is a statement formed by the name of an array, an operator (`<<` for left shift, `>>` for right shift) and an expression indicating the amount of the shift. The array-shift statement moves the elements of an array to the left (`<<`) or to the right (`>>`). The elements are moved k cells in the specified direction, where k is the value of the second operand of the statement. The elements moved beyond one of the boundaries of the array, are discarded. All the places in the array where no number is shifted in from the original one are to be padded with zeros.

So, if `a` is a 6 elements array initialized as `[0, 1, 2, 3, 4, 5]`:

- the result of Example 1 is `a=[0, 0, 0, 1, 2, 3]`.
- assuming `x=4`, the result of Example 2 is `a=[4, 5, 0, 0, 0, 0]`.

If the operand on the left hand side of the shift operator is not an array, Acse should print an error message and terminate.

Implement the array-shift as a statement, not as an expression.

Explicit any other assumption you made to implement the support for the `array-shift` operator.

[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**). (1 points)
   The solution can be found in the attached patch.

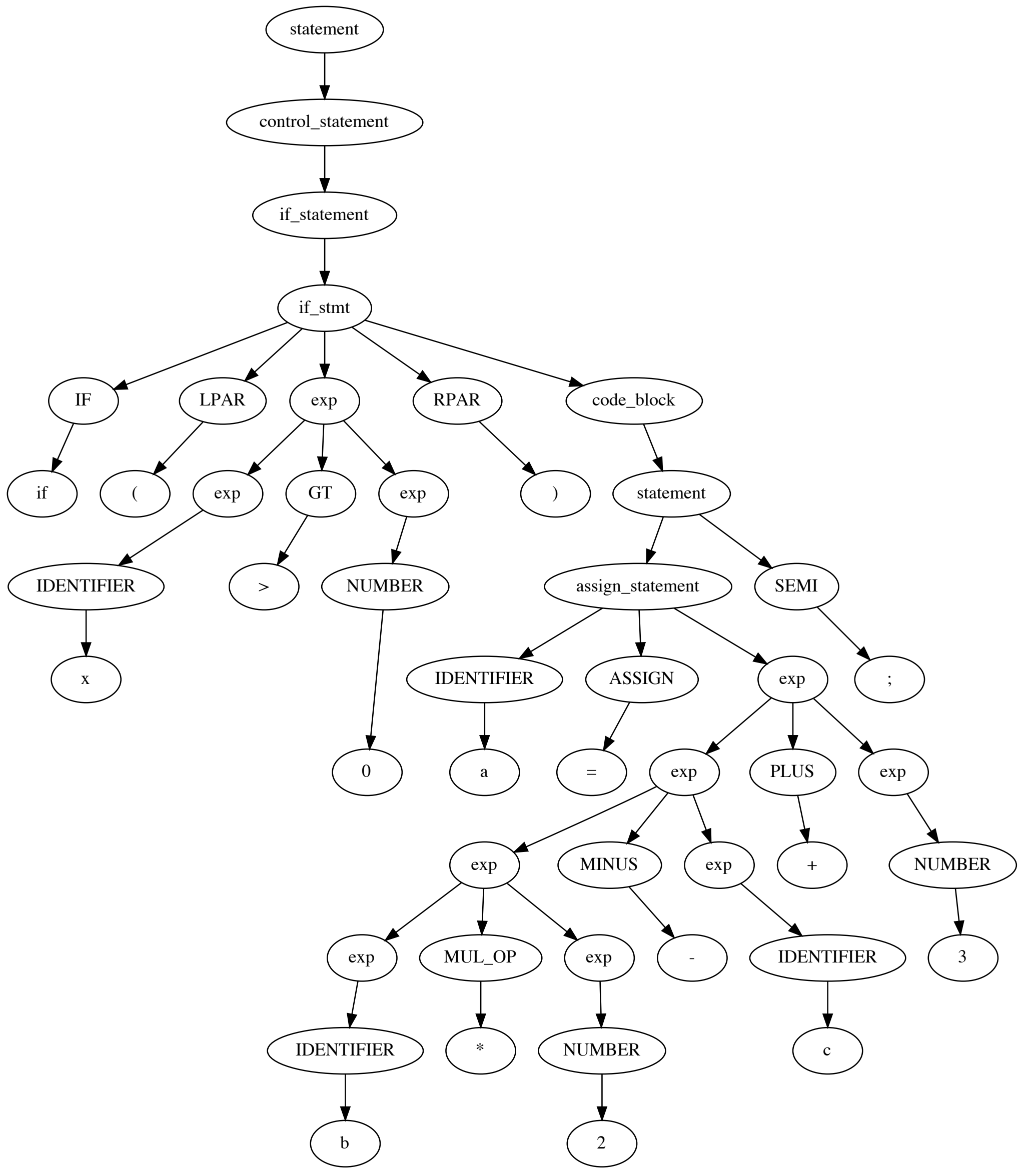
2. Define the syntactic rules or the modifications required to the existing ones. (4 points)
   The solution can be found in the attached patch.

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 snippet:

```
if(x>0)
  a=b*2-c+3;
```

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



5. (Bonus) Describe how it would be possible to implement the support for explicitly indicating which element should be inserted into the empty positions instead of the zeros. (3 points)

```
Example:
 a << 2 (x)
```

If, for example, a=[0, 1, 2, 3, 4, 5] and x=2, after the execution of the statement it will result:
a=[2, 3, 4, 5, 1, 1].
As another example, if a=[0, 1, 2, 3, 4, 5] and x=7, after the statement execution it will result:
a=[2, 3, 4, 5, 7, 7].

The syntactic rule for the array-shift operation must be modified as follows:

```
array_shift_statement :  IDENTIFIER SHL_OP exp LPAR exp RPAR
                      |  IDENTIFIER SHR_OP exp LPAR exp RPAR
```

In the semantic action, the rule should copy the computed value bound to \$5 in all the cells left without a shifted-in value.
This can be accomplished through employing a loop structure, exploiting the computed value bound to \$3 to know how many cells should be padded (since a shift by x positions will leave exactly x cells without a defined value).
It is *not* possible to generate a sequence of stores as the number of cells to be padded with the specific value is not known at compile time, thus the use of a proper loop structure is mandatory in this case.