Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 09/02/2015[1]

Time: 60 minutes. Textbooks and notes can be used. Pencil writing is allowed.
Important: Write your name on any additional sheet.

SURNAME (Cognome): .....

NAME (Nome): .....

Matricola: ..... or Person Code: .....

Instructor: ☐ Prof. Breveglieri ☐ Prof. Morzenti ☐ Prof. Agosta

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 evaluate the factorial operator of an expression and with the absolute value of an expression.

This operations are available through two new expressions ! (**factorial**) and $|\cdot|$ (**absolute value**), having the following syntax respectively:

$$\langle \textit{exp. 1} \rangle !$$
$$|\langle \textit{exp. 2} \rangle|$$

It's unnecessary to consider the case where the result of the factorial and absolute value are not representable in the register.

```
int x, y;

read(x);
read(y);

// factorial of x
write(x!);

// factorial of x!
write(x! !);

// abs of x
write(|x|);

// abs of |x| - y!
write(| |x| - y! |);
```

[1] The text and solution to this exam have been adapted to ACSE 2.0 from its initial formulation.

1. Define the tokens (and the related declarations in **scanner.l** and **parser.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)

Solution

The solution is shown below in *diff* format. All lines that begin with '+' were added, while the lines that begin with '−' were removed. **It is not required and it is not encouraged to provide a solution in *diff* format to get the maximum grade.**

```
diff --git a/acse/parser.y b/acse/parser.y
--- a/acse/parser.y
+++ b/acse/parser.y
@@ -438,6 +438,62 @@ exp
     $$ = getNewRegister(program);
     genOR(program, $$, rNormalizedOp1, rNormalizedOp2);
   }
+  | exp NOT_OP
+  {
+    // Factorial implemented as repeated multiplication from 1 to $1
+    //   int factorial = 1;
+    //   for (int i = 1; i <= $1; i++){
+    //     factorial *= i;
+    //   }
+    //   return factorial;
+    // Generate a new register that will contains the factorial
+    t_regID rFactorial = getNewRegister(program);
+    // Assign the factorial register to the result of exp
+    $$ = rFactorial;
+    // Generate initialization of the factorial to 1
+    genADDI(program, rFactorial, REG_0, 1);
+    // Generate a new iterator register and initialize it to 1
+    t_regID rIterator = getNewRegister(program);
+    genADDI(program, rIterator, REG_0, 1);
+    // Create a label for the beginning of the loop
+    t_label* lStartLoop = createLabel(program);
+    // Create a label for the end of the loop
+    t_label* lEndLoop = createLabel(program);
+    // Assign start of the loop
+    assignLabel(program, lStartLoop);
+    // Generate: if iterator > $1 jump to the end of the loop
+    genBGT(program, rIterator, $1, lEndLoop);
+    // Generate: factorial *= iterator
+    genMUL(program, rFactorial, rFactorial, rIterator);
+    // Generate: iterator += 1
+    genADDI(program, rIterator, rIterator, 1);
+    // Generate a jump back to the start of the loop
+    genJ(program, lStartLoop);
+    // Assign end of the loop
+    assignLabel(program, lEndLoop);
+  }
+  | OR_OP exp OR_OP %prec NOT_OP // precedence specified to be higher than OR_OP
                                 ↪ otherwise it will be interpreted as multiple or
+  {
+    // Absolute value implemented as:
+    //   if($1 < 0){
+    //     return 0 - $1;
+    //   }
+    //   return $1;
+    // Generate a new register to contains the absolute value
+    t_regID rAbsValue = getNewRegister(program);
+    // Assign the absolute value register to the result of exp
+    $$ = rAbsValue;
+    // Generate initialization of the absolute value with the argument $2
+    genADD(program, rAbsValue, REG_0, $2);
```

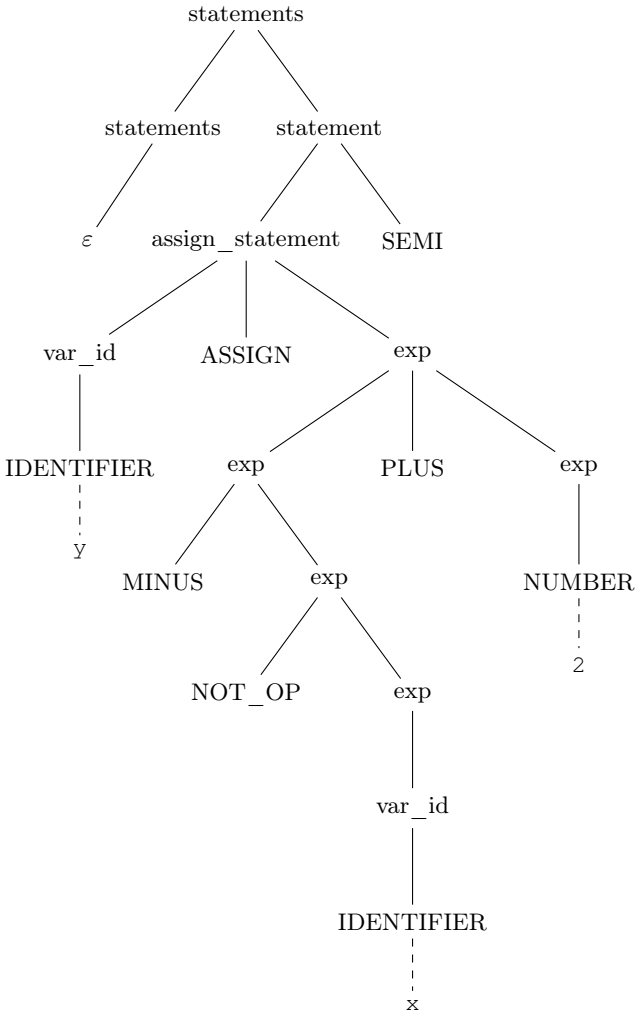
```
+    // Create a label for the end of the if
+    t_label* lEndIf = createLabel(program);
+    // Generate: if absolute value >= 0 jump to end of the if
+    genBGE(program, rAbsValue, REG_0, lEndIf);
+    // Generate: absolute_value = 0 - absolute_value
+    genSUB(program, rAbsValue, REG_0, rAbsValue);
+    // Assign end of the if
+    assignLabel(program, lEndIf);
+  }
 ;

 var_id
diff --git a/tests/abs_fact/abs_1.src b/tests/abs_fact/abs_1.src
--- /dev/null
+++ b/tests/abs_fact/abs_1.src
@@ -0,0 +1,10 @@
+int v;
+int y;
+
+read(v);
+
+write(|v|);
+
+write(|v|-5);
+
+write(| |v|-5|);
diff --git a/tests/abs_fact/fact_1.src b/tests/abs_fact/fact_1.src
--- /dev/null
+++ b/tests/abs_fact/fact_1.src
@@ -0,0 +1,9 @@
+int v;
+int y;
+
+read(v);
+
+write(v!);
+write(v!!);
+write(!v);
+write(!v!);
```

4. Given the following `Lance` code snippet:

```
y = -!x + 2;
```

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

Solution



5. (**Bonus**) We are asked to extend the absolute value operator to the arrays. Describe how to implement the absolute value operator for arrays inside assignment statements.

$$\langle id\ .1 \rangle \ = \ |\langle id\ .2 \rangle|$$

Where $\langle id\ .1 \rangle$ and $\langle id\ .2 \rangle$ are `ids` of arrays of the same lengths