Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 25/01/2024[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 **iterated expressions**.

This operation is available through a new statement called **`repeat_exp`**, having the following syntax:

`repeat_exp`(⟨*var.*⟩ = ⟨*exp. 1*⟩, ⟨*exp. 2*⟩, ⟨*exp. 3*⟩);

The `repeat_exp` statement implements a simple loop which, at every iteration, re-computes ⟨*exp. 3*⟩ and assigns it to ⟨*var.*⟩. ⟨*var.*⟩ is initialized to the value of ⟨*exp. 1*⟩ before the loop. The number of iterations is controlled by the value of ⟨*exp. 2*⟩.

If ⟨*exp. 2*⟩ is zero or negative, the loop should never iterate, leaving ⟨*var.*⟩ to its initial value, i.e. ⟨*exp. 1*⟩. Even though this is allowed by the syntax, assume that ⟨*exp. 2*⟩ does not use ⟨*var.*⟩ in its definition. As a result you can assume that ⟨*exp. 2*⟩ doesn't change during the execution of the loop.

The following code snippet exemplifies the operation of the `repeat_exp` statement. The first `repeat_exp` statement initializes a to zero, then performs the assignment $a \leftarrow a + 1$ ten times — as specified by the ⟨*exp. 2*⟩ argument — leaving $a = 10$ at the end. The second `repeat_exp` initializes b to $a \times 3 = 30$, but then ⟨*exp. 2*⟩ evaluates to -5, which is a negative value. As a result, b is left equal to 30 and is not changed again. The third occurrence of `repeat_exp` implements a variant of the well-known recurrent formula $x_{i+1} = \frac{1}{2}\left(x_i + \frac{N}{x_i}\right)$ to compute $\sqrt{4096576}$.

```
int a, b;

repeat_exp(a=0, 10, a+1);
// a == 10

repeat_exp(b=a*3, a-15, b-1);
// b == 30

repeat_exp(a=1000, 3, (a+4096576/a)/2);
// a == 2024
```

[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**). (1 point)
2. Define the syntactic rules or the modifications required to the existing ones. (2 points)
3. Define the semantic actions needed to implement the required functionality. (22 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 -Naur acse_2.0.2/acse/parser.h acse_2.0.2_patched/acse/parser.h
--- acse_2.0.2/acse/parser.h    2024-12-11 22:09:21
+++ acse_2.0.2_patched/acse/parser.h    2024-12-11 22:09:33
@@ -26,6 +26,16 @@
    t_label *lExit; ///< Label to the first instruction after the loop.
 } t_whileStmt;

+// Note that instead of defining a new semantic type in the union declaration,
+// we could have used global variables as the repeat_exp statement structurally
+// cannot be nested (being a statement, it cannot appear inside one of the
+// expressions!)
+typedef struct {
+  t_regID rIdx;
+  t_label *lLoop;
+  t_label *lExit;
+} t_repeatExpStmt;
+
 /**
  * @}
  */
diff -Naur acse_2.0.2/acse/parser.y acse_2.0.2_patched/acse/parser.y
--- acse_2.0.2/acse/parser.y    2024-12-11 22:09:21
+++ acse_2.0.2_patched/acse/parser.y    2024-12-11 22:09:39
@@ -50,6 +50,7 @@
    t_label *label;
    t_ifStmt ifStmt;
    t_whileStmt whileStmt;
+  t_repeatExpStmt repeatExpStmt;
 }

 /*
@@ -77,6 +78,7 @@
 %token <label> DO
 %token <string> IDENTIFIER
 %token <integer> NUMBER
+%token <repeatExpStmt> REPEAT_EXP

 /*
  * Non-terminal symbol semantic value type declarations
@@ -182,7 +184,46 @@
    | return_statement SEMI
    | read_statement SEMI
    | write_statement SEMI
+  | repeat_exp_statement SEMI
    | SEMI
+;
+
+repeat_exp_statement
+  : REPEAT_EXP LPAR var_id ASSIGN exp COMMA exp COMMA
+  {
+    // Generate the assignment of <exp. 1> to the variable.
+    genStoreRegisterToVariable(program, $3, $5);
+
+    // Reserve a new register which will be used as a loop counter, and
+    // generate the code which initializes the loop counter with
```

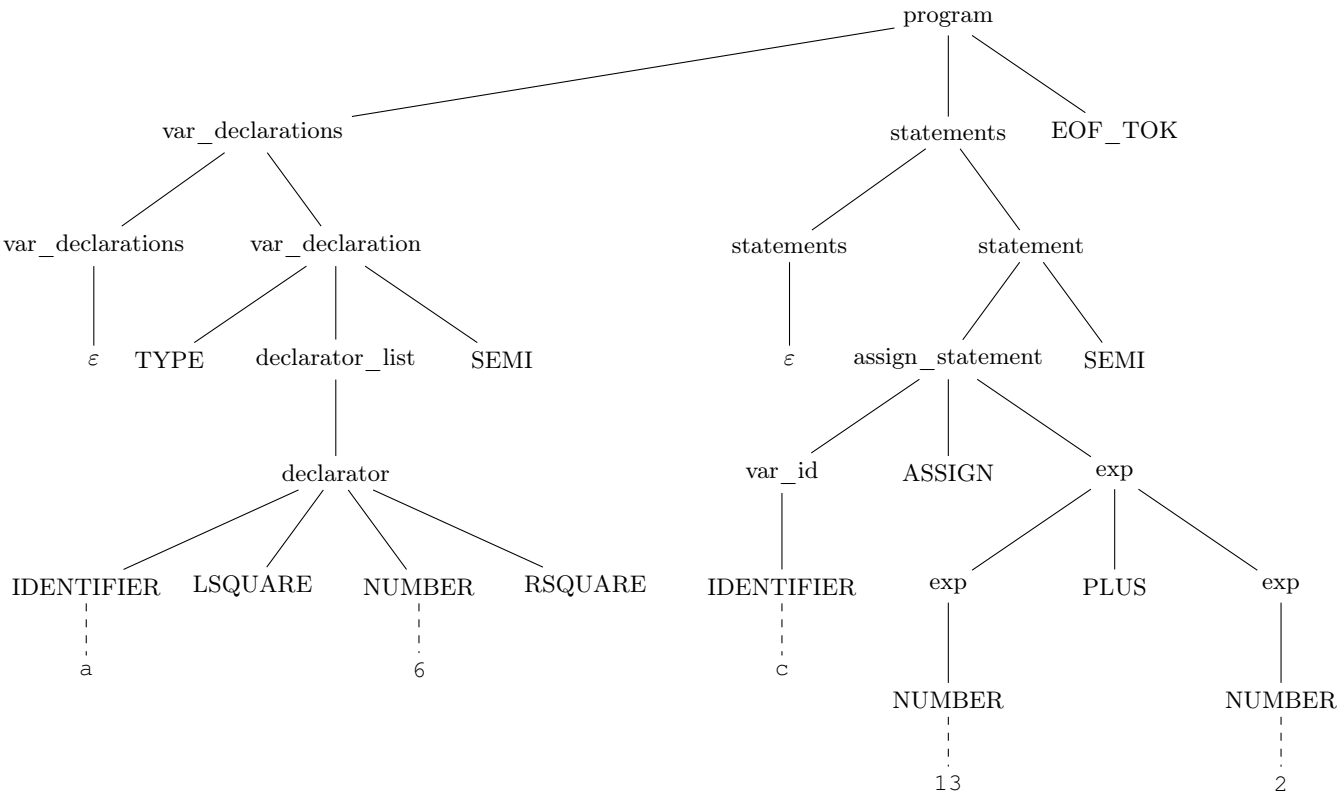
```
+    // the value of <exp. 2>.
+    // The loop counter will be decremented at every iteration.
+    $1.rIdx = getNewRegister(program);
+    genADDI(program, $1.rIdx, $7, 0);
+
+    // Generate the label for the back-edge of the loop
+    $1.lLoop = createLabel(program);
+    assignLabel(program, $1.lLoop);
+    // Generate the code for testing if the loop counter is <= 0 and exit the
+    // loop in that case
+    $1.lExit = createLabel(program);
+    genBLE(program, $1.rIdx, REG_0, $1.lExit);
+    // At this point the rule for `exp` will generate some code which computes
+    // the value of <exp. 3>.
+  }
+  exp RPAR
+  {
+    // Generate code which will assign the value of <exp. 3> of the variable
+    // at each loop iteration.
+    genStoreRegisterToVariable(program, $3, $10);
+
+    // Generate code to decrement the loop counter and jump back to the head
+    // of the loop
+    genSUBI(program, $1.rIdx, $1.rIdx, 1);
+    genJ(program, $1.lLoop);
+    // Generate the label which points just after the loop for exiting it.
+    assignLabel(program, $1.lExit);
+  }
 ;

 /* An assignment statement stores the value of an expression in the memory
diff -Naur acse_2.0.2/acse/scanner.l acse_2.0.2_patched/acse/scanner.l
--- acse_2.0.2/acse/scanner.l   2024-12-11 22:09:21
+++ acse_2.0.2_patched/acse/scanner.l   2024-12-11 22:09:21
@@ -81,6 +81,7 @@
 "return"                  { return RETURN; }
 "read"                    { return READ; }
 "write"                   { return WRITE; }
+"repeat_exp"              { return REPEAT_EXP; }

 {ID}                      {
                             yylval.string = strdup(yytext);
diff -Naur acse_2.0.2/tests/exp_loop/exp_loop.src acse_2.0.2_patched/tests/exp_loop/exp_loop.src
--- acse_2.0.2/tests/exp_loop/exp_loop.src      1970-01-01 01:00:00
+++ acse_2.0.2_patched/tests/exp_loop/exp_loop.src      2024-12-11 22:09:21
@@ -0,0 +1,16 @@
+int a, b;
+
+repeat_exp(a=0, 10, a+1);
+write(a); // 10
+
+repeat_exp(b=a*3, a+5, b-1);
+write(b); // 15
+
+repeat_exp(a=b/3, 0, 123456);
+write(a); // 5
+
+repeat_exp(a=10000, 3, (a+200000000/a)/2);
+write(a); // 14142
+
+repeat_exp(a=1000, 3, (a+4096576/a)/2);
+write(a); // 2024
```

4. Given the following `Lance` code snippet:

```
int a[6]; c=13+2;
```

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

Solution



5. (**Bonus**) Briefly explain in plain English words how to modify the solution given to the questions about the `repeat_exp` statement in order to stop the loop earlier if the value of the variable stays the same for at least two consecutive iterations.