Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 13/07/2016[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` with the **if statement with initializer** construct. The construct allows a programmer to specify, within the round braces following the *if* keyword of a common if-statement, both the condition which should be holding true and an arbitrary statement. The entire statement, including its mandatory terminating semicolon, is prefixed to the usual if-statement condition. The if statement with initializer construct is denoted by the `iif` keyword in place of the usual `if`. An example is provided in the following code snippet

```
int i,j;
i=5;
j=20;
iif (i=i+2; (j*5)>i) {
  j = i;
}
```

The computation of the initializing statement is always executed before the evaluation of the condition of the if statement is evaluated and its usual behaviour is followed, i.e., the code block is executed if the condition is true.

[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 **Acse.lex** and **Acse.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 -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 21:55:35
+++ acse_2.0.2_patched/acse/parser.y    2024-12-11 21:56:52
@@ -77,6 +77,7 @@
 %token <label> DO
 %token <string> IDENTIFIER
 %token <integer> NUMBER
+%token <ifStmt> IIF

 /*
  * Non-terminal symbol semantic value type declarations
@@ -177,6 +178,7 @@
 statement
   : assign_statement SEMI
   | if_statement
+  | iif_statement
   | while_statement
   | do_while_statement SEMI
   | return_statement SEMI
@@ -207,6 +209,32 @@
     // Generate a jump to the else part if the expression is equal to zero.
     $1.lElse = createLabel(program);
     genBEQ(program, $3, REG_0, $1.lElse);
+  }
+  code_block
+  {
+    // After the 'then' part, generate a jump to the end of the statement.
+    $1.lExit = createLabel(program);
+    genJ(program, $1.lExit);
+    // Assign the label which points to the first instruction of the else part.
+    assignLabel(program, $1.lElse);
+  }
+  else_part
+  {
+    // Assign the label to the end of the statement.
+    assignLabel(program, $1.lExit);
+  }
+;
+
+/* An iif statements first assign than computes the expression, then jumps to
+ * the 'else' part if the expression is equal to zero.
+ * Otherwise the 'then' part is executed.
+ * After the 'then' part the 'else' part needs to be jumped over. */
+iif_statement
+  : IIF LPAR assign_statement SEMI exp RPAR
+  {
+    // Generate a jump to the else part if the expression is equal to zero.
+    $1.lElse = createLabel(program);
+    genBEQ(program, $5, REG_0, $1.lElse);
   }
   code_block
   {
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 21:55:35
+++ acse_2.0.2_patched/acse/scanner.l   2024-12-11 21:55:35
```

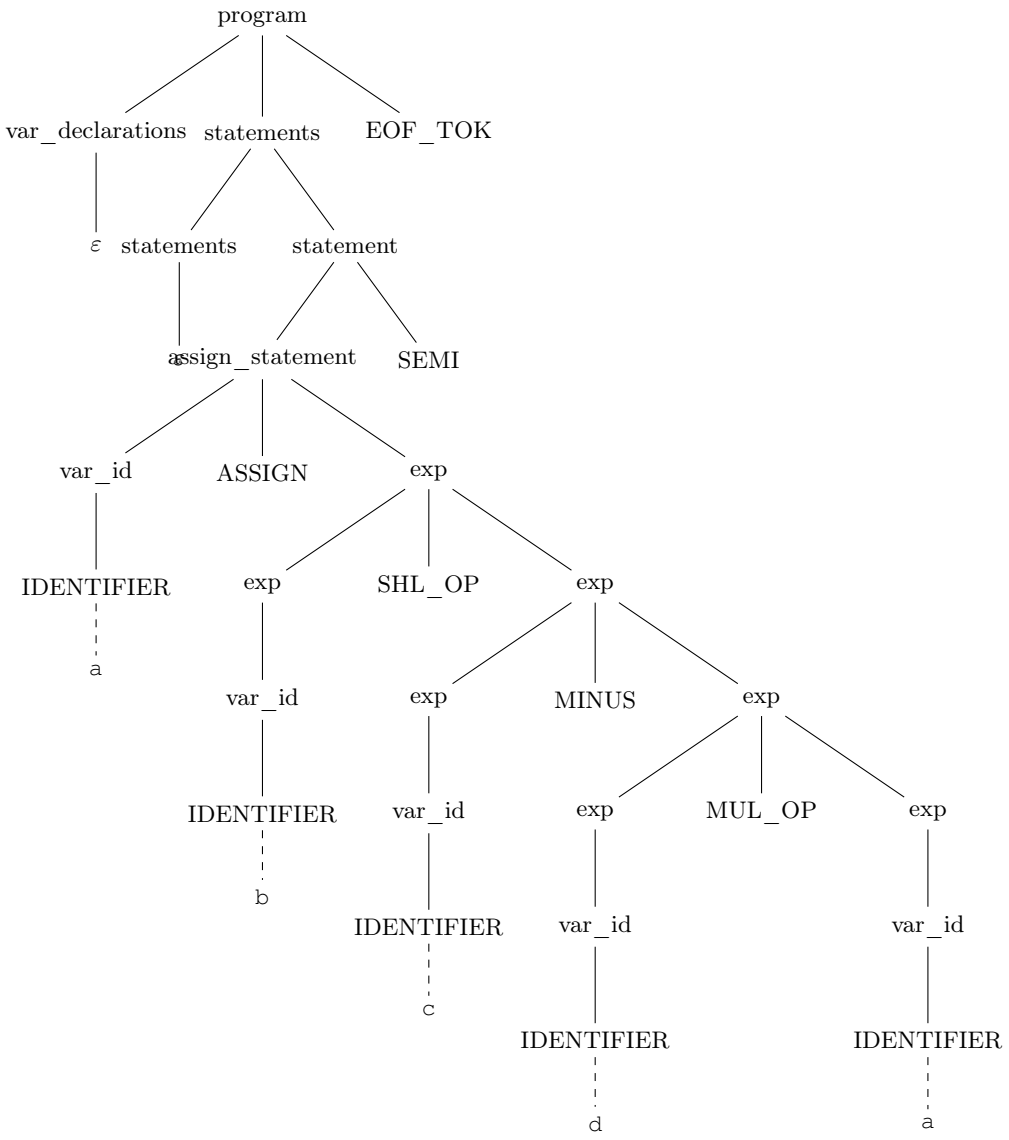
```
@@ -81,6 +81,7 @@
 "return"                  { return RETURN; }
 "read"                    { return READ; }
 "write"                   { return WRITE; }
+"iif"                     { return IIF; }

 {ID}                      {
                             yylval.string = strdup(yytext);
diff -Naur acse_2.0.2/tests/iif/iif.src acse_2.0.2_patched/tests/iif/iif.src
--- acse_2.0.2/tests/iif/iif.src      1970-01-01 01:00:00
+++ acse_2.0.2_patched/tests/iif/iif.src      2024-12-11 21:55:35
@@ -0,0 +1,8 @@
+int i,j;
+i=5;
+j=20;
+iif (i=i+2; (j*5)>i) {
+  j = i;
+}
+write(i);
+write(j);
```

4. Given the following `Lance` code snippet:

```
a = b << c - d * a;
```

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

Solution



5. (**Bonus**) Describe the modifications to be made to your solution to allow any valid ACSE statement, including the newly added `if statement with initializer`, to appear as the initializing statement of the new construct.