

Formal Languages and Compilers

Prof. Breveglieri, Morzenti, Agosta

Written exam: laboratory question

13/06/2024¹

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. Breviglieri ☐ 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 new **if-repeat-until** statement. This new statement has the following syntax:

```
if_repeat (<exp. 1>) <block> until (<exp. 2>);
```

The `if-repeat-until` statement is a loop controlled by the two expressions $\langle exp. 1 \rangle$ and $\langle exp. 2 \rangle$. $\langle exp. 1 \rangle$ is only evaluated before the loop itself: if it is false (i.e. equal to zero), the loop is not executed. Instead, when $\langle exp. 1 \rangle$ is true (i.e. different than zero) the $\langle block \rangle$ is executed in a loop which *stops* when $\langle exp. 2 \rangle$ is evaluated to true. In all cases $\langle exp. 1 \rangle$ is only evaluated once.

The following program exemplifies the operation of the `if-repeat-until` statement. First, an integer value is read from standard input and assigned to variable i . Then, the `if-repeat-until` statement checks if $i < 3$. If that is not the case, the entire statement is skipped and execution continues at the “`write(99);`” line. For instance, if the number inserted is 5, the program skips the loop and only prints 99.

If the $i < 3$ condition is met, the loop starts executing. At each iteration, the loop body prints i and then increments it. The loop stops when i reaches 6, and then the program reaches the “`write(99);`” line. For instance, if the number inserted is 2, the program prints 2 3 4 5 99. Note that – while performing the loop – i can violate the $i < 3$ condition, as it is only checked before the loop.

```
int i;
read(i);
if_repeat (i < 3) {
    write(i);
    i = i + 1;
} until (i == 6);
write(99);
```

¹The 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 -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 22:10:05
+++ acse_2.0.2_patched/acse/parser.h 2024-12-11 22:10:15
@@ -26,6 +26,12 @@
     t_label *lExit; ///< Label to the first instruction after the loop.
 } t_whileStmt;

+/** Utility structure used to store information about a if-repeat statement. */
+typedef struct {
+    t_label *lLoop; ///< Label to the beginning of the loop.
+    t_label *lExit; ///< Label to the first instruction after the loop.
+} t_ifRepeatStmt;
+
+
+/**
+ * @}
+ */
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 22:10:05
+++ acse_2.0.2_patched/acse/parser.y 2024-12-11 22:10:18
@@ -50,6 +50,7 @@
     t_label *label;
     t_ifStmt ifStmt;
     t_whileStmt whileStmt;
+    t_ifRepeatStmt ifRepeatStmt;
 }

/*
@@ -70,6 +71,7 @@
%token TYPE
%token RETURN
%token READ WRITE ELSE
+%token UNTIL

// These are the tokens with a semantic value.
%token <ifStmt> IF
@@ -77,6 +79,7 @@
%token <label> DO
%token <string> IDENTIFIER
%token <integer> NUMBER
+%token <ifRepeatStmt> IF_REPEAT

/*
 * Non-terminal symbol semantic value type declarations
@@ -182,6 +185,7 @@
| return_statement SEMI
| read_statement SEMI
| write_statement SEMI
+ | if_repeat_statement SEMI
| SEMI
;

@@ -303,6 +307,27 @@
t_regID rTmp = getNewRegister(program);
genLI(program, rTmp, '\n');
genPrintCharSyscall(program, rTmp);
```

```

+ }
+;
+if_repeat_statement
+ : IF_REPEAT LPAR exp RPAR
+ {
+ // Allocate the object for the label used to jump out of the statement
+ $1.lExit = createLabel(program);
+ // Generate code to skip the entire statement body if the expression is
+ // equal to zero
+ genBEQ(program, $3, REG_0, $1.lExit);
+ // Create and assign the label for the back-edge to the
+ // beginning of the loop
+ $1.lLoop = createLabel(program);
+ assignLabel(program, $1.lLoop);
+ }
+ code_block UNTIL LPAR exp RPAR
+ {
+ // Generate code to jump back to the loop body if the condition is not true
+ genBEQ(program, $9, REG_0, $1.lLoop);
+ // Assign the label used to exit from the statement
+ assignLabel(program, $1.lExit);
+ }
+;

```

```
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 22:10:05
```

```
+++ acse_2.0.2_patched/acse/scanner.l 2024-12-11 22:10:05
```

```
@@ -81,6 +81,8 @@
```

```

"return"          { return RETURN; }
"read"            { return READ; }
"write"           { return WRITE; }
+"if_repeat"      { return IF_REPEAT; }
+"until"          { return UNTIL; }

```

```

{ID}              {
                    yylval.string = strdup(yytext);

```

```
diff -Naur acse_2.0.2/tests/if_repeat/if_repeat.src acse_2.0.2_patched/tests/if_repeat/if_repeat.src
```

```
--- acse_2.0.2/tests/if_repeat/if_repeat.src 1970-01-01 01:00:00
```

```
+++ acse_2.0.2_patched/tests/if_repeat/if_repeat.src 2024-12-11 22:10:05
```

```
@@ -0,0 +1,7 @@
```

```

+int i;
+read(i);
+if_repeat (i > 0) {
+ write(i);
+ i = i - 1;
+} until (i == 0);
+write(0);

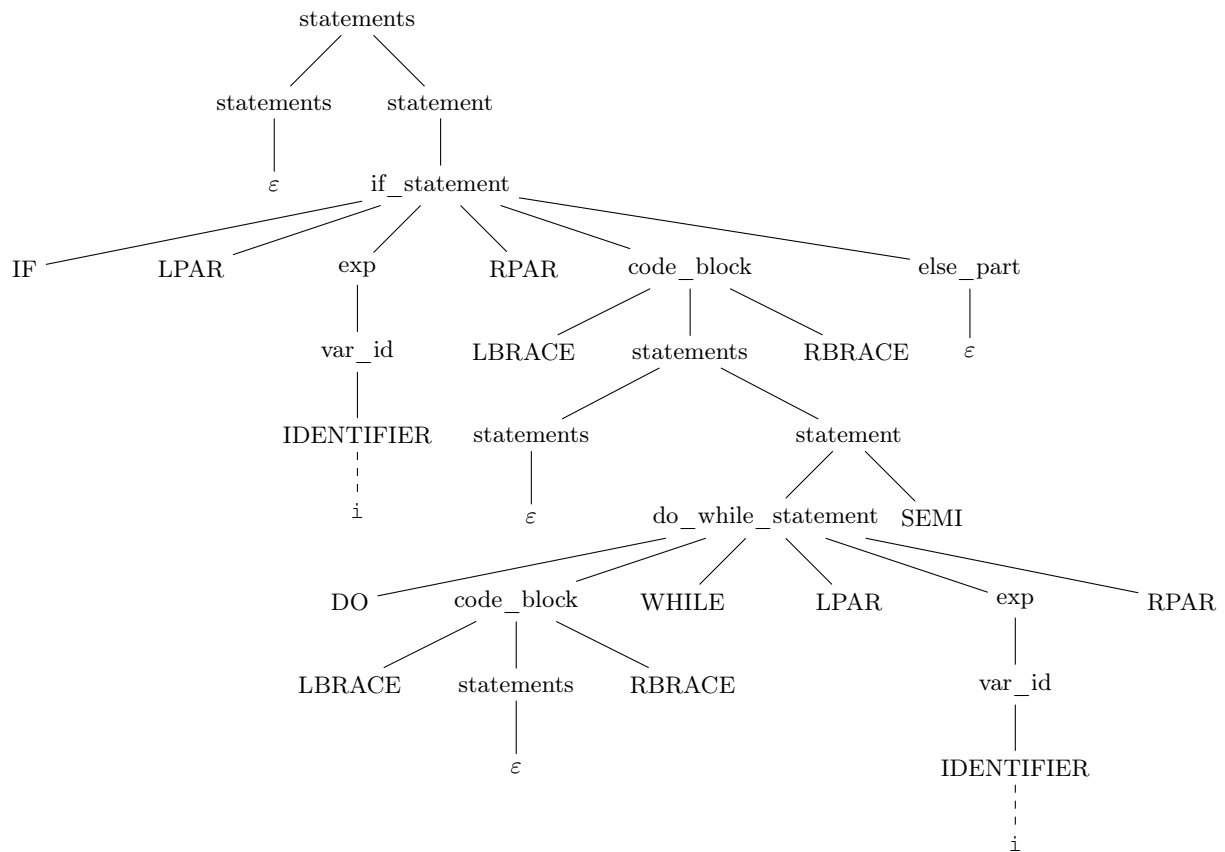
```

4. Given the following Lance code snippet:

```
if (i) { do {} while (i); }
```

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**) Write the required modifications to the token definitions and syntactic rules defined in the answers to questions (1) and (2) in order to also support a syntax that does not have the “if” part in the statement, as shown in the following example:

```
int i;  
read(i);  
repeat {  
    write(i);  
    i = i - 1;  
} until (i == 0);  
write(0);
```