Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 13/06/2022

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 **`zip`** statement. The `zip` statement takes two source arrays, and copies their elements to a destination array in pairs – that is, by alternating elements from the first array and elements from the second.

The source arrays may be of different lengths, or shorter than required to fill the entire destination array. In these cases, the number of pairs of values copied is determined by the length of the shortest input array. The rest of the destination array is left unchanged. In alternative, the destination array may be shorter than required to contain all the elements from the source arrays. In that case the `zip` statement stops as soon as the last element of the destination array is written. If any of the identifiers does not refer to an array, a syntax error is generated and compilation is stopped.

The syntax and operation of the `zip` statement is shown in the example below. In the first example, the c array is filled with the contents of a and b alternatively, in the order in which they appear in the syntax (first a, then b). The a array is shorter than b, (5 elements vs. 6), so the length of a determines the number of pairs to be copied (5 pairs). The number of elements copied is 10 (5 from a, 5 from b) and the rest of the destination array c is left unchanged. In the second example, a is filled with elements from b and c alternatively. The number of pairs to be copied is determined by the shortest array (b) but the a array is shorter than required. As a result only 2 pairs are copied fully, and the last pair is copied only half-way.

```
int a[5], b[6], c[12];
/* a == [1, 2, 3, 4, 5] */
/* b == [10, 20, 30, 40, 50, 60] */
/* c == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] */

c = zip(a, b);
/* c == [1, 10, 2, 20, 3, 30, 4, 40, 5, 50, 0, 0] */

a = zip(b, c);
/* a == [10, 1, 20, 10, 30] */
```

1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.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.**

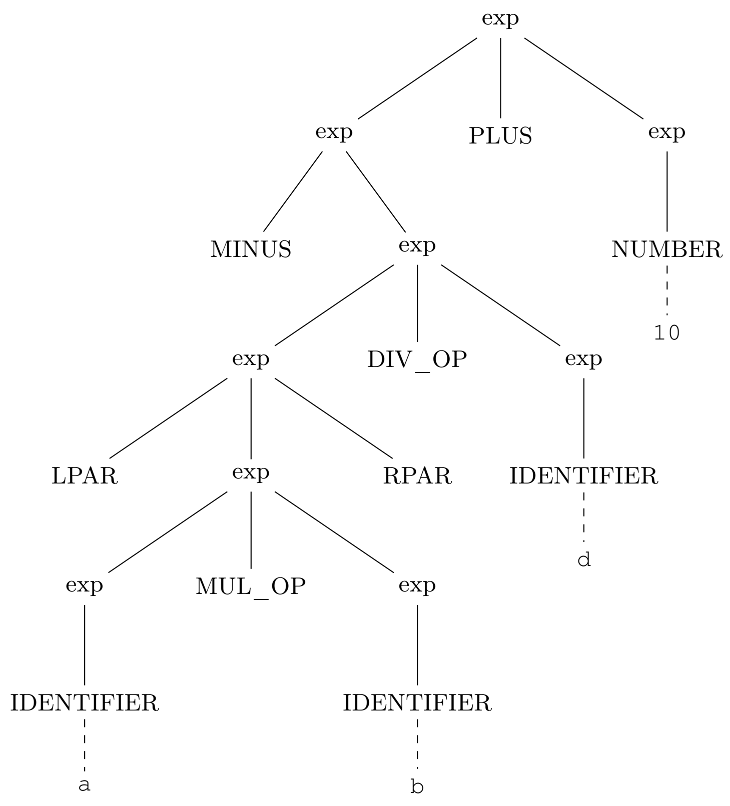
```
diff --git a/acse/Acse.lex b/acse/Acse.lex
index 35f9b73..5ba002d 100644
--- a/acse/Acse.lex
+++ b/acse/Acse.lex
@@ -93,6 +93,7 @@ ID       [a-zA-Z_][a-zA-Z0-9_]*
 "return"          { return RETURN; }
 "read"            { return READ; }
 "write"           { return WRITE; }
+"zip"             { return ZIP; }

 {ID}              { yylval.svalue=strdup(yytext); return IDENTIFIER; }
 {DIGIT}+          { yylval.intval = atoi( yytext );
diff --git a/acse/Acse.y b/acse/Acse.y
index 0636dc6..f3d2005 100644
--- a/acse/Acse.y
+++ b/acse/Acse.y
@@ -125,6 +125,7 @@ extern void yyerror(const char*);
 %token RETURN
 %token READ
 %token WRITE
+%token ZIP

 %token <label> DO
 %token <while_stmt> WHILE
@@ -247,6 +248,7 @@ statements  : statements statement       { /* does nothing */ }
 statement   : assign_statement SEMI      { /* does nothing */ }
             | control_statement          { /* does nothing */ }
             | read_write_statement SEMI  { /* does nothing */ }
+            | zip_statement SEMI
             | SEMI            { gen_nop_instruction(program); }
 ;

@@ -260,6 +262,79 @@ read_write_statement : read_statement  { /* does nothing */ }
                      | write_statement { /* does nothing */ }
 ;

+zip_statement: IDENTIFIER ASSIGN ZIP LPAR IDENTIFIER COMMA IDENTIFIER RPAR
+   {
+      /* Check if all three input/output arguments are in fact arrays */
+      t_axe_variable *v_dest = getVariable(program, $1);
+      t_axe_variable *v_src1 = getVariable(program, $5);
+      t_axe_variable *v_src2 = getVariable(program, $7);
+      if (!v_dest->isArray || !v_src1->isArray || !v_src2->isArray) {
+         yyerror("at least one source/destination to zip is not an array!");
+         YYERROR;
+      }
+
+      /* Compute the number of pairs of values to copy to the destination
+       * array */
+      int n_pairs = v_src1->arraySize;
+      if (n_pairs > v_src2->arraySize)
+         n_pairs = v_src2->arraySize;
+
+      /* Reserve two registers used for keeping track of the current index
+       * in the source and destination arrays. Generate code for initializing
+        * both registers to zero. */
+       int srci_reg = gen_load_immediate(program, 0);
+       int dsti_reg = gen_load_immediate(program, 0);
+
+       /* Generate a label at the beginning of the loop which at runtime copies
+        * the values. */
+       t_axe_label *l_loop = assignNewLabel(program);
+       /* Reserve a label for exiting the loop */
+       t_axe_label *l_exit = newLabel(program);
+
+       /* Generate code for copying an element from the first source array to
+        * the destination array */
+       int r_tmp = loadArrayElement(program,
+             $5, create_expression(srci_reg, REGISTER));
+       storeArrayElement(program,
+             $1, create_expression(dsti_reg, REGISTER),
+             create_expression(r_tmp, REGISTER));
+       /* Generate code to increment the index in the destination array. */
+       gen_addi_instruction(program, dsti_reg, dsti_reg, 1);
+       /* Generate code for checking if we are at the end of the destination
+        * array, and exiting the loop in that case. */
+       gen_subi_instruction(program, REG_0, dsti_reg, v_dest->arraySize);
+       gen_bge_instruction(program, l_exit, 0);
+
+       /* Generate code for copying an element from the second source array to
+        * the destination array */
+       r_tmp = loadArrayElement(program,
+             $7, create_expression(srci_reg, REGISTER));
+       storeArrayElement(program,
+             $1, create_expression(dsti_reg, REGISTER),
+             create_expression(r_tmp, REGISTER));
+       /* Generate code to increment the index in the destination array. */
+       gen_addi_instruction(program, dsti_reg, dsti_reg, 1);
+       /* Generate code for checking if we are at the end of the destination
+        * array, and exiting the loop in that case. */
+       gen_subi_instruction(program, REG_0, dsti_reg, v_dest->arraySize);
+       gen_bge_instruction(program, l_exit, 0);
+
+       /* Generate code to increment the index in the source arrays and continue
+        * the loop if we are not finished. */
+       gen_addi_instruction(program, srci_reg, srci_reg, 1);
+       gen_subi_instruction(program, REG_0, srci_reg, n_pairs);
+       gen_blt_instruction(program, l_loop, 0);
+
+       /* Assign (generate) the label for exiting the loop. */
+       assignLabel(program, l_exit);
+
+       /* Free the identifiers. */
+       free($1);
+       free($5);
+       free($7);
+    }
+;
+
 assign_statement : IDENTIFIER LSQUARE exp RSQUARE ASSIGN exp
             {
                /* Notify to 'program' that the value $6
diff --git a/tests/zip/zip.src b/tests/zip/zip.src
new file mode 100644
index 0000000..070a21a
--- /dev/null
+++ b/tests/zip/zip.src
@@ -0,0 +1,32 @@
+int a[5], b[6], c[12];
+int i;
+
+i = 0;
+while (i < 5) {
+   a[i] = i + 1;
+   i = i + 1;
+}
+i = 0;
+while (i < 6) {
+   b[i] = (i + 1) * 10;
+   i = i + 1;
+}
+i = 0;
+while (i < 12) {
+   c[i] = 0;
+   i = i + 1;
+}
+
+c = zip(a, b);
+i = 0;
+while (i < 12) {
+   write(c[i]);
+   i = i + 1;
+}
+
+a = zip(b, c);
+i = 0;
+while (i < 5) {
+   write(a[i]);
+   i = i + 1;
+}
```

4. Given the following `Lance` code snippet:

```
-(a * b) / d + 10
```

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

Solution



5. (**Bonus**) Describe how the given implementation of the `zip` statement can be extended such that, when the destination array has not been declared yet, it is instantiated automatically.

   In particular, describe which ACSE utility function needs to be called in order to instantiate the array, and how the size of the array is derived.