Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 20/01/2023

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 check if an array element exists or not before performing an assignment. This operation is performed by an expression operator called **inbounds**. Its syntax consists of the inbounds keyword itself, followed by two parameters enclosed in parenthesis: a *destination* and a *source*, in this order. The parameters are separated by "=", mimicking the syntax used for assignments. If the left-hand side of "=" is not a scalar variable, or if the right-hand side of "=" is not an array element, then the compilation fails.

When inbounds appears in an expression, it verifies that the index of the specified array element is within bounds, i.e. if the index is greater or equal than zero and lesser than the size of the array. If this condition is satisfied, the operator copies the array element to the scalar and returns 1, otherwise it returns zero and no copy is performed.

The following code snippet exemplifies the operation and syntax of inbounds. The first *if* statement checks the validity of index 0 (zero) in array a. The index is within bounds, therefore the value of a[0], which is 8, is put into v and the first section of the *if* is executed. In the following statement, inbounds appears in the expression argument of a write and checks if the index $-1 + v$ is valid. The variable v has just been assigned to 8, so the index is 7. Since this index is too large, a zero is printed and v is unmodified. Finally, the program uses inbounds to print the last 3 values in a. At every iteration the index i is checked, and as soon as it is not in bounds the condition of the *while* statement becomes false and the loop terminates.

```
int v, i, a[5];
a[0]=8; a[1]=6; a[2]=9; a[3]=4; a[4]=7;

if (inbounds(v = a[0])) {
  write(1);                  // v = a[0], writes 1
} else {
  write(2);
}

write(inbounds(v = a[-1+v]));  // check fails, writes 0

i = 2;
while (inbounds(v = a[i])) {
  write(v);
  i = i + 1;
}                            // the loop writes 9, 4, 7
```

1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.y**). (2 points)
2. Define the syntactic rules or the modifications required to the existing ones. (3 points)
3. Define the semantic actions needed to implement the required functionality. (20 points)

4. Given the following `Lance` code snippet:

```
-6 < -a & -b * -7
```

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

5. (**Bonus**) Describe the modifications to the specification and implementation of the `inbounds` operator required for supporting assignments to a second array.