Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 08/06/2023

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 search an array for elements with a given value, and then reassign them with another value.

This operation is performed using a new statement called **`replace`**. The syntax of the statement is the following:

`replace``(`⟨*array*⟩`,` ⟨*exp. 1*⟩`,` ⟨*exp. 2*⟩`);`

The first argument, ⟨*array*⟩, is the identifier of the array being modified by the statement. The second argument, ⟨*exp. 1*⟩, is the expression whose value must be searched in the array. The third and last argument, ⟨*exp. 2*⟩, is the new value that will be assigned to all items found with the same value as ⟨*exp. 1*⟩. If no elements of the array have the same value of ⟨*exp. 1*⟩, the array is left unchanged. Additionally, if ⟨*array*⟩ is not a valid array identifier, a syntax error is raised at compile-time.

The following code snippet exemplifies the use of the statement. Initially, we assume the contents of array `a` are initialized to the integer numbers from 1 to 10 (the code to initialize `a` is not shown for brevity). The first use of `replace` replaces all items in `a` whose value is $b - 17 = 7$, with 5. Only the item at index 6 is equal to 7, therefore its value is replaced with 5. The second use of `replace` replaces all items whose value is 5 with $\frac{b}{2} = 12$. The items at index 4 and 6 satisfy the condition and are indeed replaced with the value 12. Finally, the third `replace` finds all items of value $-b \times 10 = -120$ and replaces them with $-24 + b = 0$. Since there are no items in the array with the specified value, the array stays unchanged.

```
int a[10], b=24;

/* a == {1, 2, 3, 4, 5, 6, 7, 8, 9,10}; */
replace(a, b-17, 5);
/* a == {1, 2, 3, 4, 5, 6, 5, 8, 9,10}; */
replace(a, 5, b/2);
/* a == {1, 2, 3, 4,12, 6,12, 8, 9,10}; */
replace(a, (-b)*10, -24+b);
/* a == {1, 2, 3, 4,12, 6,12, 8, 9,10}; */
```

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:

```
5 || b | 7 && 12 < 7 == c * 2
```

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**) Briefly explain in plain English words how to modify the solution given to the questions about the `replace` statement in order to add support for performing multiple replacements in sequence, like in the following example.

```
int a[10];

/* a == {1, 2, 3, 4, 5, 6, 7, 8, 9,10}; */
replace(a, 1, 2, 2, 3); // Replace 1 with 2, then 2 with 3
/* a == {3, 3, 3, 4, 5, 6, 7, 8, 9,10}; */
replace(a, 10, 9, 8, 7, 6, 5); // 10->9, 8->7, 6->5
/* a == {3, 3, 3, 4, 5, 5, 7, 7, 9, 9}; */
```