Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 13/06/2024[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` 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 ⟨*exp. 1*⟩ and ⟨*exp. 2*⟩. ⟨*exp. 1*⟩ is only evaluated before the loop itself: if it is false (i.e. equal to zero), the loop is not executed. Instead, when ⟨*exp. 1*⟩ is true (i.e. different than zero) the ⟨*block*⟩ is executed in a loop which *stops* when ⟨*exp. 2*⟩ is evaluated to true. In all cases ⟨*exp. 1*⟩ 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);
```

[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 **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)

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)

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);
```