

# Formal Languages and Compilers

## Prof. Breveglieri, Morzenti, Agosta

### Written exam: laboratory question

04/07/2024<sup>1</sup>

**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: .....

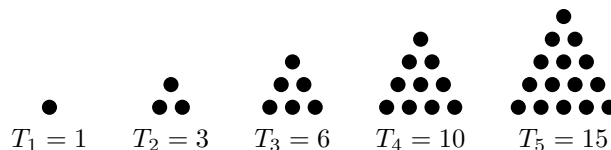
Instructor: ☐ Prof. Breviglieri ☐ Prof. Morzenti ☐ Prof. Agosta

The laboratory question must be answered taking into account the implementation of the Acse compiler given with the exam text.

Triangular numbers are integers that count objects arranged in a triangle. The value of the  $n$ th triangular number is defined by the following equation:

$$T_n = \frac{n(n+1)}{2}$$

The zero-th triangular number is defined equal to zero ( $T_0 = 0$ ) and there are no preceding triangular numbers (in other words,  $n$  cannot be less than zero). Examples of triangular numbers are shown in the following picture.



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 compute the  $n$ th triangular number using a new **expression operator**, called **tri**, with the following syntax:

**tri** ( $\langle exp \rangle$ )

The only **argument** to the `tri` operator is an **arbitrary expression** which specifies the index of the triangular number to compute. For example, `tri(5)` will compute the value 15. The operator is also **applicable to negative numbers**, but it always returns **zero**. For example, `tri(-10)` has the value 0.

## Example

The following program shows an example of use of the `tri` operator.

```
int i = 0;
// Print T_1, T_2, ... T_10
while (i < 10) {
    write(tri(i + 1));
    i = i + 1;
}
```

---

<sup>1</sup>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:

```
a[b[c[d*10*e]]]
```

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 discuss in plain English words why it is **not necessary** to provide precedence and associativity declarations for the `tri` operator.

Additionally, show an alternative syntax for `tri` that **does require** the declaration of its precedence and associativity with respect to other expression operators.