# Formal Languages and Compilers
**Exam**

Using the LEX and YACC programs (or Flex and Bison), realize a syntax-driven translator capable of recognizing a language that describes a set of products and gives them some scores.

## Input Language

The input language defines a series of product types; for each of them a **set of attributes** that apply to its type may be defined. A **weight** can be assigned to the attributes, useful for the comprehensive assessment of the product.
Product descriptions that assign a value to the attributes may follow the definition of the product types. The definition of a type consists of a list of attributes followed by an arrow ("->") and the name of the products type. The attributes list is enclosed within round brackets and consists of a sequence of definitions of attributes, separated by commas (','). The attribute definitions are made by the attribute name, followed by a colon (':') and by the weight assigned to that attribute. The weight is an unsigned integer. The names of the types and of the attributes are compliant with the rules of C identifiers.
Several definitions of types may be present; and they won't be separated by any character. The last definition of a type will be followed by ('.'), that separates the definitions of the types from the product descriptions. Multiple product descriptions may appear, each of them having the following structure:

```
<type name> : <reviews> = <product name> ;
```

The type name must be one of the previously defined types. The reviews consist of a sequence of valuations separated by commas (','). The product name is represented by a sentence.
A valuation consists of a value, described by a symbol ('*', '+', '/''-') and by an attribute name. The meaning of symbols is described below:

| Symbol | Meaning | Value |
|---|---|---|
| * | Excellent | 3 |
| + | Good | 2 |
| / | Sufficient | 1 |
| - | Unsatisfactory | 0 |

Attribute names must be those defined for the type of product. It is not necessary for all the attributes defined for a product type to appear in the product description.
A sentence consists of a set of words and numbers separated by any number of white spaces. The words are sequences of alphabetic characters, that can be lower case or upper case characters. The numbers are unsigned integers.
The program must be able to recognize and ignore comments that follow the C++ syntax of C, namely beginning with the characters "/ /" and ending with a carriage return.

## The purpose of the program

The program should recognize the previously described language and must verify the consistency of the semantics (proper use of types and attributes names). After the analysis, the program must print a list of products with their score divided by type.
The score of a product is calculated as follows. For every attribute of the product type its weight is multiplied by the value assigned in the product description. The score of the product will be the sum

of all the values obtained in this way. In other words, being $\mathbf{p_i}$ the weight of the i-th attribute, $\mathbf{v_i}$ the value assigned to the i-th attribute within the product in question and **N** the number of attributes; the score of a product will be computed as follows:

$$Score = \sum_{i=1}^{N} p_i \cdot v_i$$

When a syntactic error is encountered, the program must report it and stop the execution: no need for management and error recovery.

Example
Input file:

```
// Definition of the product types:
( taste : 12, perfume : 8 ) -> wine
( taste : 10, transparency: 2 ) -> honey
.
// Description of the products:
wine: * taste, + perfume = barbera DOC;
wine: * taste, * perfume = barolo di annata;
wine: - taste, / perfume = a stinky wine;
honey: * taste, * transparency = acacia honey;
```

The program should produce the following output (the output format can be different, but it should contain the same information):

```
Achieved scores.
barbera DOC, 52
barolo di annata, 60
a stinky wine, 8
acacia honey, 36
```