## 1. General Guidelines

Make a program in assembly language that evaluates mathematical expressions.

In the next section we will expose a series of functionalities that may have or not the project

The first functionality (common functionality) will be present in all the equipment.

Each one of these functionalities contributes a number of points to the project note and to the final grade.

Students can achieve a grade higher than 5 in the project which helps them will give with the final grade of the subject.

The best tasks will be considered for the prize of the subject.

It may be that no task reaches the award category.

The task will be done in teams of 2 students.

The teams can be students from the same group or from different groups.

The teams are defined in the 1st practical class after the orientation.

Teams of only one student are accepted (in rare cases) but never of 3.

The review is by team. If in a team there is a student who knows a lot and another student that knows little the note of the one that knows more does not have to be high.

The main thing that will be evaluated in the project is the ability to program in blador and the ability to solve problems independently.

The tasks that are not complete will have a happier ending but the members of the team know everything that happens (and are able to make modifications and arrangements) that the tasks in which the functionalities are done with high level of completion but team members do not know how to defend the project.

The project note is a combination of:

- Running program.
- Number of errors detected during execution.
- Functions that the team has chosen.
- Questions that arise during the review.
- Project defense, explanation, etc.

The total sum of points of all functionalities is 13 which corresponds to the maximum points a team can reach.

## 2. Functionalities

### 2.1. Common Part (2 points)

1. The program must allow 2 types of instructions:

   <variable> = <expression> (assignments)

   print <expression> (print)

   <variable> is the valid name for an identifier in any programming language.
   mation The maximum length that the names of the variables will have is 3 characters.

   <expression> is a mathematical expression composed of the following:

   16-bit signed integers

   Arithmetic operators (+ - * /%)

   signs of grouping (parentheses)

2. The program entry is a sequence of instructions.
3. The output of the program is a result for each print statement
4. the communication between the program and the users will be using the keyboard and the screen.

5. The user at each moment enters a line of text that will be processed.
6. If the line is empty the program ends.
7. Spaces are always ignored. The program must allow the following two cases (the the only required space is the one that goes after the print)

```
a = 5 + 1

a = 5         + 1
```

8. If an expression can not be evaluated, an ERROR poster will be printed and the program happens to wait for the next instruction.
9. If a variable that has not been declared is used in the right part, it should be displayed an error poster.
10. Example:

```
> a = 10
> b = a + 4
> print a * a + b
= 114
>
END
```

2.2. Descriptive Errors (0.5 points)

Instead of setting ERROR for all errors, write descriptive errors to all Possible situations, for example:

```
ERROR: poorly balanced expression.

ERROR: variable not declared.

ERROR: result of the operation out of range

ERROR: division by zero
```

2.3. Matrices and Vectors (2 points)

1. A new type of data appears which are the matrices and as a particular case they are single-column arrays (column vectors) and single-row arrays (row vectors)

2. The representation format is as follows:

   column vectors (example: [1; 3; 4] or [20; 30])

   row vectors (example: [1 & 2 & 3] or [21 & 31])

   matrices (example [1 & 2; 2 & 3; 3 & 4])

3. The arithmetical operations that from now on exist are:

   sum: (+)

   - integer + integer = integer
   - column + column = column
   - row + row = row
   - matrix + matrix = matrix

   subtraction: (-) similar to the sum

   multiplication: (*)

   - whole * whole = integer
   - row * integer = row
   - column * whole - column
   - array * integer = array
   - row * row = row (scalar product)
   - column * column = column (scalar product)
   - row * column = matrix
   - matrix * matrix = matrix

   division: (/)

   - integer / integer = integer
   - row / integer = row
   - column / integer = column

   - matrix / integer = matrix

4. The cases that are not contemplated in the previous point are considered ERROR.

5. Notes:

   The column vectors are separated by semicolons.

   The row vectors are separated by ampersand (&).

   The column vectors can be interpreted as matrices of a single element by row.

   Row vectors can be interpreted as single-row arrays.

   When an operation can not be carried out, ERROR will be printed.

   When adding two rows or two columns or two matrices they must have the the same dimension otherwise ERROR is printed.

   In multiplication, the dimensions of the matrices and the
   If you can not multiply, ERROR must be printed.

2.4. Extra Operations with Matrices (1 point)

1. Depends on Matrices and Vectors (2.3)

2. Each student will define what the syntax will be like for operations.

3. The new operations that will appear are the following:

   Reverse matrices

   Find the transpose.

   Find determinant.

2.5. Additional Arithmetic Operations (1 point)

1. the new operations are:

   power (^)

   square root

   factorial

   nth root

   summation

   shift left (<<)

   shift right (>>)

   not (~)

   or (!)

   and (&)

2. The student will decide the syntax for operations not defined by the operator.

2.6. Functions (2 points)

1. From now on, functions of any number of variables can be effected.

2. Both the name of the function and the name of the variables can be a sequence up to 3 valid characters for an identifier.

3. The left part of the function (what is to the left of the sign =) is of the type f (x) or fun (pri, sec).

4. The right part is an expression that may depend on:

   The parameters defined in the left part.

   Other variables defined above (they must be defined if they are not im- first error)

   Other functions already defined evaluated in the parameters: f (x) = g (x) + 5

2.7. Boolean (1 point)

1. The comparison operators appear (<<=>> = ==! =) That bool returns example: a = 3> 2 (a = true)

2. The logical operators appear between boolean (||, &&,!)

3. Does the operator appear: (condition)? result_true: result_false

2.8. Large numbers (1 point)

1. All operands and results can have up to 10 digits.

2. It is recommended to implement multiplication and division in galley (method that learn in elementary school)

2.9. Float (1 point)

1. Depends on Large Numbers (2.8)

2. All operations can be carried out with floating point numbers.

3. The numbers may have 10 digits before the decimal point and 10 after.

2.10. Recursive functions (1 point)

1. Depends on Functions (2.6)

2. Depends on Float (2.9)

3. Can be defined as follows
   fact (n) = n <2? 1: n * fact (n-1)

2.11. Long variable names (0.5 points)

1. Depends on Functions (2.6)

2. The names of the variables and functions can have any length.

3. There may be functions and variables with the same name.

4. There may be functions with the same name as long as they receive different amounts of parameters (overload).

2.12. Other Functions

Any other functionality that is implemented will be assessed at the time of review the number of points that can or can not contribute to the note of it.