Assignment 2 to 4: Lambda Spreadsheet

In these assignments, we wish to implement a small language for computing on spreadsheets. A spreadsheet (similar to an Excel sheet) is a dynamic data structure essentially made up cells, arranged in rows and columns, that can help store and process data.

An example use is to calculate the total weighted score of each student including the assignments, quizzes and minor1 marks. You will be given a spreadsheet filled with marks of each student in different tests and assignments. You need to evaluate some *formulas* based on those values. For this, you will need to learn lex and yacc to design the language of the formulas and then evaluate the results.

In Assignment 2: you only need to provide the lex specifications, and so do not need to concern yourself with the meaning and implementation of the operations and how they are used.
In Assignment 3: you need to specify the structure of formulas as a yacc grammar, and call the appropriate operations that will be provided to you as a module. Your implementation will be modular, in that you should be able to define your parser and create the interpreter framework irrespective of the actual implementation of the backend functions.
In Assignment 4: you will implement the backend, type-check the operations, and integrate the entire work.

Each assignment is to be submitted by the deadline. However, the demonstrations will be taken together. You may make changes to the work submitted earlier, but must clearly document any such improvements.

In each Assignment, you need to write clear specifications of the OCaml functions which you implement, and document your code.

Informal Syntax

Informally the grammar of our spreadsheet formula language is:

- An instruction has a head and a body. The head consists of a cell indexing the output destination of the formula, and the body consists of an expression being evaluated with respect to the spreadsheet. The head and the body are separated by an assignment sign (**:=**)
- A formula ends with a semicolon (**;**)
- The arguments of a function are either indices, ranges, constants or expressions using mathematical operations
- An index is a pair of integers. General format for an index *I* of a cell is [*i, j*] Examples are [0,0], [3,4], …
- A range consists of two indices standing for the top left index and the right bottom index. General format for an range *R* of cells is **(** *I* **:** *I* **)** with an example being **([** 5, 6**] : [**100, 6] **)**

Tokens for OCamllex would be, but not limited to:

- Float constants (with optional sign, no redundant initial zeroes before the decimal point or unnecessary trailing zeroes after the decimal point)
- Parenthesis — **(** and **)**
- Brackets — **[** and **]**
- Comma — **,**
- Colon — **:**
- Indices *I* — **[** *i* **,** *j* **]**
- Ranges *R* — **(** *I* **:** *I* **)**
- Unary operators: **SUM**, **AVG**, **MIN**, **MAX**, **COUNT**, etc. (see below)
- Binary operators: **ADD** (addition), **SUBT** (subtraction), **MULT** (multiplication), **DIV** (division)
- Assignment operator **:=**
- Formula termination **;** (semicolon)

Note that in our spreadsheet, indexing will begin at index 0, and unlike Excel will be numeric for both rows and columns.

Formulas will be of 3 kinds: overall, row-wise, column-wise. The cell selection would be made by specifying the top-left and bottom-right cell indices. Results would be filled into the sheet by specifying only the top-left cell of the target cell/row/column.

The function operations can be one of the following:

Type 1 (unary operations on ranges of cells):

- COUNT
- ROWCOUNT
- COLCOUNT
- SUM
- ROWSUM
- COLSUM
- AVG
- ROWAVG
- COLAVG
- MIN
- ROWMIN
- COLMIN
- MAX
- ROWMAX
- COLMAX

Type 2 (binary operations on ranges of cells):

- ADD
- SUBT
- MULT
- DIV

The arguments of these functions may be either the cell ranges or might be constants. Think of weighted sum as an example. You will multiply a column by a constant and add it to another column to give the final result. Then you might multiply by 100 and divide by the total marks in order to get the percentage.

The formulas provided to you are one on each line. The general format of a formula is given below:

Type 1 formula (unary operations on a range of cells)
I **:= FUNC** *R* **;**

Type 2 formula (binary operations on ranges)
I **:= FUNC** *R R* **;**
I **:= FUNC** *C R* **;**
I **:= FUNC** *R C* **;**
I **:= FUNC** *I R* **;**
I **:= FUNC** *R I* **;**

Note: If any entry reference in a formula is missing, the result is undefined.

Examples:

Initial Sheet:

1	10	100			
2	20	200			
3	30	300			

[0, 3] := ADD ([0, 0] : [2, 0]) ([0, 1] : [2, 1])

Sheet representation:

1	10	100	11		
2	20	200	22		
3	30	300	33		

[0, 4] := MULT ([0, 0] : [2, 0]) ([0, 2] : [2, 2])
Sheet representation:

1	10	100	11	100	
2	20	200	22	400	
3	30	300	33	900	

[0, 5] := ADD ([0, 4] : [2, 4]) **10**
Sheet representation:

1	10	100	11	100	110
2	20	200	22	400	410
3	30	300	33	900	910

[0, 3] := DIV ([0, 4] : [2, 4]) [1,1]
Sheet representation:

1	10	100	5	100	110
2	20	200	20	400	410
3	30	300	45	900	910

Backend

In order to evaluate the above functions, you might want to implement the following in OCaml:

- full_count: sheet -> range -> index -> sheet : Fills count of valid entries in the given range into the specified cell
- row_count: sheet -> range -> index -> sheet : Fills count of valid entries per row in the given range into the column starting from the specified cell
- col_count: sheet -> range -> index -> sheet : Fills count of valid entries per column in the given range into the row starting from the specified cell.

- full_sum: sheet -> range -> index -> sheet : Fills the sum of entries of cells in the given range into the specified cell
- row_sum: sheet -> range -> index -> sheet : Fills the sum of entries of cells per row in the given range into the column starting from the specified cell
- col_sum: sheet -> range -> index -> sheet : Fills the sum of entries of cells per column in the given range into the row starting from the specified cell

- full_avg: sheet -> range -> index -> sheet : Fills the average of entries of cells in the given range into the specified cell
- row_avg: sheet -> range -> index -> sheet : Fills the average of entries of cells per row in the given range into the column starting from the specified cell
- col_avg: sheet -> range -> index -> sheet : Fills the sum of entries of cells per column in the given range into the row starting from the specified cell

- full_min: sheet -> range -> index -> sheet : Fills the min of entries of cells in the given range into the specified cell
- row_min: sheet -> range -> index -> sheet : Fills the min of entries of cells per row in the given range into the column starting from the specified cell
- col_min: sheet -> range -> index -> sheet : Fills the min of entries of cells per column in the given range into the row starting from the specified cell

- full_max: sheet -> range -> index -> sheet : Fills the max of entries of cells in the given range into the specified cell
- row_max: sheet -> range -> index -> sheet : Fills the max of entries of cells per row in the given range into the column starting from the specified cell
- col_max: sheet -> range -> index -> sheet : Fills the max of entries of cells per column in the given range into the row starting from the specified cell

- add_const: sheet -> range -> float -> index -> sheet : adds a constant to the contents of each cell in the selected cell range
- subt_const: sheet -> range -> float -> index -> sheet : subtracts a constant from the contents of each cell in the selected cell range
- mult_const: sheet -> range -> float -> index -> sheet : multiplies the contents of each cell in the selected cell range by a constant.

- div_const: sheet -> range -> float -> index -> sheet : divides the contents of each cell in the selected cell range by a constant.

- add_range: sheet -> range -> range -> index -> sheet : adds the cell contents for each corresponding pair of cells in two selected cell ranges
- subt_range: sheet -> range -> range -> index -> sheet : performs a subtraction on the cell contents for each corresponding pair if cells in two selected cell ranges
- mult_range: sheet -> range -> range -> index -> sheet : multiplies the cell contents for each corresponding pair of cells in two selected cell ranges
- div_range: sheet -> range -> range -> index -> sheet : performs a division on the cell contents for each corresponding pair of cells in two selected cell ranges

The data to be entered into the sheet would be provided subsequently. For the time being, you may assume any grid size and feed in the data of your choice to build the spreadsheet functionality.