Question 4 (30%)

We shall now consider *CALClite*, a simple spreadsheet system. The spreadsheet is defined by the following type declarations:

```
type Row = int
type Col = char
type CellAddr = Row * Col
type ArithOp = Add | Sub | Mul | Div
type RangeOp = Sum | Count
type CellDef =
    FCst of float
  | SCst of string
  | Ref of CellAddr
  | RangeOp of CellAddr * CellAddr * RangeOp
  | ArithOp of CellDef * ArithOp * CellDef
type CellValue =
    S of string
  | F of float
type Sheet = Map<CellAddr,CellDef>
```

A *cell address* (`CellAddr`) is defined as a *row*- and *column*-index, written `A1` for column `A` and row `1`. We assume columns are in the interval `'A' ... 'Z'` and rows greater or equal to 1. A *cell definition* (`CellDef`) can be a float constant (`FCst`), a string constant (`SCst`), a reference to another cell (`Ref`), an operation on a range (`RangeOp`) or an binary arithmetic operation (`ArithOp`). A range is defined as all cells within the rectangle defined by the top left cell address and the bottom right cell address. The only operations allowed on a range are summing all cells (`Sum`) and counting the number of cells (`Count`). Evaluating a cell definition (`CellDef`) will either result in a string or a float value represented with the type `CellValue`. A spreadsheet, type `Sheet`, is defined as a mapping from cell adresses to cell definitions.
The result of rolling 12 dice from question 1.1 is repeated below:

```
   |        A |        B |        C |        D |        E |        F |        G |        H
---+----------+----------+----------+----------+----------+----------+----------+---------
 1 |    #EYES |        1 |        2 |        3 |        4 |        5 |        6 |    Total
 2 |   RESULT |     2.00 |     1.00 |     5.00 |     0.00 |     2.00 |     2.00 |    12.00
 3 |      PCT |    16.67 |     8.33 |    41.67 |     0.00 |    16.67 |    16.67 |   100.00
```

The cell `H2` is the sum of the range `B2` to `G2`. The cell `B3` is the percentage number of 1's, i.e., `B2/H2*100.0`. The cells `C3`, ..., `H3` are defined similar to `B3`. The definition of the sheet, called `dice`, is as follows:

```
let header = [((1,'A'),SCst "#EYES");((1,'B'),SCst "1");((1,'C'),SCst "2");
              ((1,'D'),SCst "3");((1,'E'),SCst "4");((1,'F'),SCst "5");
              ((1,'G'),SCst "6");((1,'H'),SCst "Total")]
let result = [((2,'A'),SCst "RESULT");((2,'B'),FCst 2.0);((2,'C'),FCst 1.0);
              ((2,'D'),FCst 5.0);((2,'E'),FCst 0.0);((2,'F'),FCst 2.0);
              ((2,'G'),FCst 2.0);((2,'H'),RangeOp((2,'B'),(2,'G'),Sum))]
let calcPct col = ArithOp(FCst 100.0, Mul, ArithOp(Ref(2,col),Div,Ref(2,'H')))
let pct = [((3,'A'),SCst "PCT");((3,'B'),calcPct 'B');((3,'C'),calcPct 'C');
           ((3,'D'),calcPct 'D');((3,'E'),calcPct 'E');((3,'F'),calcPct 'F');
           ((3,'G'),calcPct 'G');((3,'H'),calcPct 'H')]
let dice = Map.ofList (header @ result @ pct)
```

Question 4.1

Declare an F# value, `heights`, of type `Sheet` corresponding to the sheet below:

```
    |        B |        C
----+----------+---------
  4 |     NAME |   HEIGHT
  5 |     Hans |   167.40
  6 |    Trine |   162.30
  7 |    Peter |   179.70
  8 |          |
  9 |     3.00 |   169.80
```

The cells `B4`, `B5`, `B6`, `B7` and `C4` are constant strings. The cells `C5`, `C6` and `C7` are constant floats. The cell `B9` is the count of cells in the range defined by the cell addresses `B5` and `B7`. The cell `C9` is defined as the sum of the range `C5` and `C7` divided by cell `B9`, i.e., the average height of the three persons.

Question 4.2

In order to evaluate a cell definition we need to evaluate range operations `Sum` and `Count` and to evaluate the arithmetic operations `Sum`, `Div`, `Sub` and `Mul`. The operation `Count` works on both float and string values because we are only counting the number of values. The other operations only work on float values. We use the function `getF` of type `CellValue -> float` to ensure a cell value is a float:

```
let getF = function
    F f -> f
  | S s -> failwith "getF: expecting a float but got a string"
```

Declare the following F# functions using `getF`:

1. `evalRangeOp xs op` of type `CellValue list -> RangeOp -> float`. For instance
   - `evalRangeOp [F 33.0; F 32.0] Sum` returns 65.0
   - `evalRangeOp [] Sum` returns 0.0
   - `evalRangeOp [F 23.0; S "Hans"] Sum` throws `System.Exception`
   - `evalRangeOp [F 23.0; S "Hans"] Count` returns 2.0
2. `evalArithOp v1 v2 op` of type `CellValue -> CellValue -> ArithOp -> float`. For instance
   - `evalArithOp (F 33.0) (F 32.0) Sub` returns 1.0
   - `evalArithOp (S "Hans") (F 1.0) Add` throws `System.Exception`

Question 4.3

In order to print a spreadsheet we need to evaluate all cells. This is done by two mutually recursive functions

- `evalValue v sheet` of type `CellDef -> Sheet -> CellValue`
- `evalCell ca sheet` of type `CellAddr -> Sheet -> CellValue`

Declare the two F# functions above. You can assume the sheet contains no cyclic cell definitions. You may use the following template

```
let rec evalValue v sheet =
  match v with
    FCst f -> F f
  | SCst s -> ..
  | Ref ca -> ..
  | RangeOp ((r1,c1),(r2,c2),op) -> ..
  | ArithOp (v1,op,v2) -> ..
and evalCell ca sheet =
  match Map.tryFind ca sheet with
    None -> S "" // We define an empty cell to be the empty string value.
  | Some v -> evalValue v sheet
```

For instance `evalCell (3,'G') dice` returns the cell value `F 16.67`.

Question 4.4

Declare a F# function `ppBoard sheet` of type `Sheet -> string` that returns a string similar to the layout used for `dice` and `heights` above. For instance, `ppBoard dice` returns the string

```
    |        A |        B |        C |        D |        E |        F |        G |        H
----+----------+----------+----------+----------+----------+----------+----------+---------
  1 |    #EYES |        1 |        2 |        3 |        4 |        5 |        6 |    Total
  2 |   RESULT |     2.00 |     1.00 |     5.00 |     0.00 |     2.00 |     2.00 |    12.00
  3 |      PCT |    16.67 |     8.33 |    41.67 |     0.00 |    16.67 |    16.67 |   100.00
```

You must include the actual output from the `ppBoard` function on the `dice` example in order to obtain full points.