```
%% calc.erl
%% v1.3, 2008-09-17
%% Declarative Programming, BME VIK
%%
%% G. Patai, P. Hanák, BME IIT;
%% P. Szeredi, G. Lukácsy, BME SZIT

-module(calc).
-export([process/1,processfix/1]).

%% process(Sheet): the value of the spreadsheet after a single evaluation step.
%% Sheet is a list of lists where each element is a cell in the sheet.
%%
%% sheet:indexlist(L) returns a list of {X,Y} pairs where X is an element of L
%% and Y is its index in L; e.g.
%%
%%  [[{Cell,ColNum,RowNum} || {Cell,ColNum} <- sheet:indexlist(Line)]
%%   || {Line,RowNum} <- sheet:indexlist([[a,b,c],[d,e,f]])].
%%
%% results in [[{a,1,1},{b,2,1},{c,3,1}],[{d,1,2},{e,2,2},{f,3,2}]]
%%
%% Note the application of process/4 in the list comprehension!
%%
process(Sheet) ->
    [ [ process(Cell, ColNum, RowNum, Sheet)
          || {Cell,ColNum} <- sheet:indexlist(Line) % cells with colnum
      ]
        || {Line,RowNum} <- sheet:indexlist(Sheet)  % lines with rownum
    ].

%% processfix(Sheet): the value of the fully evaluated spreadsheet.
%%
%% Sheet is the same as in process/1.
%%
%% A speadsheet cell that cannot be further evaluated contains a number, an
%% atom ('empty', 'error'), or an {Expr,Binds} pair leading to circularity.  In
%% case of circularity the cell content is replaced by 'loop'.
%%
processfix(Sheet) ->
    Sheet1 = process(Sheet),
    case Sheet1 == Sheet of
        true -> [ [ case Cell of
                        {_,_} -> loop ;  % {Expr, Binds} pair
                        _     -> Cell    % number or atom
                    end
                      || Cell <- Line     % each cell in the line
                  ]
                    || Line <- Sheet      % each line in the spreadsheet
                ];
        _    -> processfix(Sheet1)
    end.

%% process(Cell, ColNum, RowNum, Sheet): the computed value of the cell at
%% {ColNum,RowNum} in the spreadsheet.
%%
% Empty cell.
process(empty, _, _, _) -> empty;
% Cell containing a number.
process(Num, _, _, _) when is_number(Num) -> Num;
% Cell containing an expression with references.
process({Expr,Binds}, ColNum, RowNum, Sheet) ->
    Evaluated = fun({_,B}) -> not is_tuple(B) end,
    Invalid = fun({_,B}) -> is_atom(B) end,
    Binds1 = [{Var,findval({Sheet,ColNum,RowNum}, Ref)} || {Var,Ref} <- Binds],
    case lists:all(Evaluated, Binds1) of
        true -> case lists:any(Invalid, Binds1) of
                    true -> error;
                    _    -> sheet:eval(Expr, Binds1) % eval Erlang expression
                end;
        _    -> {Expr,Binds1}
    end;
% Anything else.
process(_, _, _, _) -> error.

```

```
%% findval(Context, X): the value of the cell referenced by X if X points to a
%% canonical expression, the result of the calculation if X is a built-in
%% function that can be evaluated, otherwise X itself unchanged.  Context is a
%% {Sheet,ColNum,RowNum} triple consisting of the spreadsheet and the
%% coordinates of the current cell.
%%
% Resolve one level of indirection
findval({Sheet,CColNum,CRowNum}, {AdrType,AColNum,ARowNum}) ->
    {ColNum,RowNum} = absaddress(CColNum, CRowNum, {AdrType,AColNum,ARowNum}),
    ValidCoords = (1 =< RowNum) and (RowNum =< length(Sheet)) and
                  (1 =< ColNum) and (ColNum =< length(hd(Sheet))),
    case ValidCoords of
        true -> case lists:nth(ColNum, lists:nth(RowNum, Sheet)) of
                    Num when is_number(Num) -> Num;
                    Atom when is_atom(Atom) -> Atom;
                    _                       -> {AdrType,AColNum,ARowNum}
                end;
        _    -> error
    end;
% Evaluate the sum function
% sum(Context, RefTL, RefBR): the sum of the cell block specified by references
% RefTL (top left) and RefBR (bottom right).  If any cell in the block is not
% yet fully evaluated then the function call is returned unchanged.  If any
% cell in the block is empty or contains an error then the result is also an
% error.
findval(Context, {sum,[RefTL,RefBR]}) ->
    func:sum(Context, RefTL, RefBR);
% This value was found earlier
findval(_, Val) -> Val.

%% absaddress(ColNum, RowNum, Ref): the absolute address of a reference from
%% the given cell.
%%
absaddress(CColNum, CRowNum, {AdrType,AColNum,ARowNum}) ->
    case AdrType of
        aa -> {AColNum,ARowNum};
        ar -> {AColNum,CRowNum+ARowNum};
        ra -> {CColNum+AColNum,ARowNum};
        rr -> {CColNum+AColNum,CRowNum+ARowNum}
    end.
```