Bachelor Computerwetenschappen
Academiejaar 2013/2014 - eerste semester
Interpretatie van Computerprogramma's II
Examenopdracht: Meerdimensionale tabellen in Pico

Theo D'Hondt
29 november 2013

Gevraagd wordt Pico uit te breiden met meerdimensionale tabellen, zoals bij voorbeeld in:

```
{ matrix[p, p]: 0;
  for (k: 1, k:= k+1, k<=p, matrix[k, k]:= 1);
  matrix[read(), read()] }
```

waar indices worden ingelezen om een element van de p x p eenheidsmatrix op te zoeken.

Meerdimensionale tabellen hebben steeds een (hyper)rechthoekige structuur, dat wil zeggen dat alle (hyper)kolommen dezelfde lengte hebben. Intern worden tabellen voorgesteld als één lineaire structuur door alle (hyper)kolommen fysiek in één *chunk* achter mekaar te plaatsen[1]. Dit is dus verschillend van geneste tabellen, waarbij via indirecties gewerkt wordt.

De syntax van meerdimensionale tabellen is als volgt:

```
<tabulation> ::= <name> [ <commalist> ]
```

en de definitie/wijziging via de `:` en `:=` syntax volgt het oorspronkelijke stramien.

Het zal nodig zijn de abstracte grammatica, de reader, evaluator, primitieve functies en printer aan te passen.

De abstracte grammatica moet meerdimensionale tabellen ondersteunen.

De reader moet in staat zijn om de uitgebreide syntax te toetsen en de correcte abstracte grammatica te genereren.

De evaluator moet in staat zijn om de definitie, wijziging en raadpleging van meerdimensionale tabellen toe te laten; in het geval van een ééndimensionale tabel moet de evaluator resultaten identiek aan de oorspronkelijke versie van Pico opleveren. Denk na over de initialisatie van een meerdimensionale tabel bij definitie aan de hand van de meegeleverde uitdrukking in de definitie.

Veralgemeen de primitieve functies **`size`** en **`tab`** om meerdimensionale tabellen te ondersteunen; voeg een functie **`dimension`** toe om de dimensie van een meerdimensionale tabel te bepalen.

Pas de printer aan om meerdimensionale tabellen te kunnen uitschrijven.

---

[1] dit is dezelfde organisatie als bij meerdimenionale vectoren in C, alhoewel de notatie afwijkt

Als eindresultaat van deze opdracht wordt verwacht (1) de metacirculaire versie van Pico-met-meerdimensionale tabellen als rechtstreekse uitbreiding van de oorspronkelijke metacirculaire Pico, (2) de daarmee overeenstemmende C-versie als rechtstreekse uitbreiding van de oorspronkelijke C-versie, (3) als niet-triviaal werkend voorbeeld een Gauss-eliminatie algoritme gebaseerd op de C-versie in bijlage en (4) een (summiere) beschrijving van het gebruikte ontwerp en de verschillen ten opzichte van de oorspronkelijke Pico.

Bijlage: Gauss-eliminatie in C:

```
//
// Solve Ax = b for x using Gaussian elimination
//
// select optimal pivot by using row and column permutations via
// indirection vectors
//

#include <stdlib.h>
#include <math.h>
#include "Gauss.h"

typedef char Result;
typedef unsigned Index;
typedef double Element;
typedef Element * Vector;
typedef Vector * Matrix;

const static Element epsilon = 1.0e-6;

static Index * row, * column;

static Element Select_Pivot(Index Size, Matrix A, Index i)
  { Index R, C, save_R, save_C, hold;
    Element ext;
    ext = A[row[i]][column[i]];
    save_R = i;
    save_C = i;
    for (R = i; R < Size; R += 1)
      for (C = i; C < Size; C += 1)
        if (fabs(A[row[R]][column[C]]) > fabs(ext))
          { ext = A[row[R]][column[C]];
            save_R = R;
            save_C = C; }
    if (save_R != i)
      { hold = row[i];
        row[i] = row[save_R];
        row[save_R] = hold; }
    if (save_C != i)
      { hold = column[i];
```

```
        column[i] = column[save_C];
        column[save_C] = hold; }
    return ext; }

Result Gauss(Index size, Matrix A, Vector b, Vector x)
  { const Index first = 0, last = size - 1;
    Index i, j, k;
    Element pivot, coef_A, coef_b;
    Result res = (Result)0;
    row = calloc(size, sizeof(Index));
    column = calloc(size, sizeof(Index));
    for (i = first; i < size; i += 1)
      row[i] = column[i] = i;
    for (i = first; i < last; i += 1)
      { pivot = Select_Pivot(size, A, i);
        if (fabs(pivot) < epsilon)
          goto exit;
        coef_b = b[row[i]] / pivot;
        for (j = i + 1; j < size; j += 1)
          { coef_A = A[row[i]][column[j]] / pivot;
            for (k = i + 1; k < size; k += 1)
              A[row[k]][column[j]] = A[row[k]][column[j]]
                                     - A[row[k]][column[i]] * coef_A;
            A[row[i]][column[j]] = coef_A;
            b[row[j]] = b[row[j]] - A[row[j]][column[i]] * coef_A; }
        b[row[i]] = coef_b; }
    if (fabs(A[row[last]][ column[last]]) < epsilon)
      goto exit;
    b[row[last]] = b[row[last]] / A[row[last]][column[last]];
    for (i = last; i > first; i -= 1)
      for (j = last; j > first; j -= 1)
        b[row[j]] = b[row[j]] - b[row[i]] * A[row[j]][column[i]];
    for (i = first; i < size; i += 1)
      x[column[i]] = b[row[i]];
    res = (Result)!0;
  exit:
    free(row);
    free(column);
    return res; }
```