Introduction to Mathematical Software 5th Exercise Sheet

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Mathematics
PD Dr. Ulf Lorenz
Christian Brandenburg

winter term 2009/2010
30/11/2009

Important Notice

Before starting this exercise sheet, you should at least have completed exercises **2.1**, **2.2**, **3.1**, **3.2**, and **4.1**.

Exercise 5.1 Decimal Expansion of Rational Numbers

Let two integers a, b be given with $b > 0$. Write a program that prints the rational number $\frac{a}{b}$ to the screen in *decimal expansion*.

Browse the internet for a suitable algorithm to accomplish this task.

The output might be finite or periodic and should be of the form `0.75` for $a = 3$, $b = 4$ in the finite case and `0.1p6` for $a = 1$, $b = 6$ in the periodic case, where the `p` denotes the beginning of the period.

Solution:

A good starting point is http://en.wikipedia.org/wiki/Decimal.

```
#include <stdio.h>
#include <string.h>

int main(void)
{

  int num;  // numerator
  unsigned int denom = 0; // denominator

  printf("Please enter numerator: ");
  scanf("%d", &num);

  while (denom == 0)
  {
    printf("Please enter denominator (!=0): ");
    scanf("%d", &denom);
  }
  printf("The decimal expansion of %d/%d is:\n", num, denom);

  if (num < 0)
    printf("-");

  // integer part; if a<b then dec is zero
  int rem = num % denom;
  int dec = num / denom;
  printf("%d.", dec);

  // initialize arrays to store the remainders and the values of the decimal places
  // there are at most denom-1 different remainders
```

```
  int decarray[denom];
  memset(decarray, 0, denom);
  int remarray[denom];
  memset(remarray, 0, denom);

  int i = 1;
  while (1)
  {
    int dd = (10*rem) / denom;
    rem = (10*rem) % denom;

    // check for period
    // if a remainder occurs for the second time, we have found a period
    int j;
    for (j = 1; j < i; j++)
    {
      if (rem == remarray[j])
      {
        if (decarray[j] != dd)
        {
          // index j does not belong to the period, so we have to add index i
          decarray[i++] = dd;
          j++;
        }

        // create output
        int k;
        for (k = 1; k < j; k++)
          printf("%d", decarray[k]);
        printf("p");
        for (k = j; k < i; k++)
          printf("%d", decarray[k]);
        printf("\n");

        return 0;
      }
    }

    remarray[i] = rem;
    decarray[i] = dd;

    // check if the expansion is finite
    if (rem == 0)
    {
      for (j = 1; j <= i; j++)
        printf("%d", decarray[j]);

      printf("\n");
      return 0;
    }

    i++;
  }

  printf("undefined");
  return 0;
}
```

Exercise 5.2 Matrix-Vector Multiplication

Write a function that computes matrix-vector products. Your program should dynamically allocate the memory needed for the matrices and vectors.

You have two possibilities to represent matrices:

- a two-dimensional array of type `double**` (i.e. an array of pointers to arrays of type `double`) with the first index being the row index and the second index being the column index. Your function for computing the matrix vector product $A \cdot x = y$ should be of the form
  `void MatMult(double** A, double* x, double* y, int rows, int cols);`
  where `m` denotes the number of rows of A (and y) and `n` denotes the number of columns of A (and rows of x).

- a one-dimensional array of type `double*` that contains all matrix rows one after the other in a single long array. The different rows can then be accessed with offsets. Your function for computing the matrix vector product $A \cdot x = y$ should be of the form
  `void MatMult(double* A, double* x, double* y, int rows, int cols);`
  with the variables defined as above.

You can proceed in the following way:

- Ask the user for the matrix dimensions (number of rows, columns).
- dynamically allocate the required memory (you need to `#include <stdlib.h>`).
- fill the matrix A and the vector x with random values or ask the user for the data. The statement `(double)rand()/(double)RAND_MAX` produces “random” numbers between 0 and 1.
- compute the matrix-vector product
- free the memory that you have allocated.
- you might want to add functions like
  `void print_matrix(double **A, unsigned int rows, unsigned int cols);`
  (or `void print_matrix(double *A, unsigned int rows, unsigned int cols);`)
  and
  `void print_vector(double *x, unsigned int rows);`
  to visualize your results.

Solution:

Matrix with `double **`:

```
#include <stdio.h>
#include <stdlib.h>

void print_matrix(double **A, unsigned int rows, unsigned int cols);

void print_vector(double *x, unsigned int rows);

void MatMult(double** A, double* x, double* y, int rows, int cols);

int main(void)
{
  unsigned int rows, cols;
  double **A;
  double *x, *y;
  unsigned int i, j;

  printf("\nEnter number of rows: ");
  scanf("%d", &rows);
  printf("\nEnter number of columns: ");
  scanf("%d", &cols);
```

```
  // allocate pointers to the rows of A
  A = (double **)malloc(rows*sizeof(double *));
  // allocate columns for each row
  for (i = 0; i < rows; i++)
    A[i] = (double*)malloc(cols*sizeof(double));

  // allocate space for x and y
  x = (double*)malloc(cols*sizeof(double));
  y = (double*)malloc(rows*sizeof(double));

  //fill A and x with random numbers
  for (i = 0; i < rows; i++)
    for (j = 0; j < cols; j++)
      A[i][j] = (double)rand()/(double)RAND_MAX;

  for (j = 0; j < cols; j++)
    x[j] = (double)rand()/(double)RAND_MAX;

  // compute matrix-vector product
  print_matrix(A, rows, cols);
  print_vector(x, cols);
  MatMult(A, x, y, rows, cols);
  print_vector(y, rows);

  // deallocate memory
  for (i = 0; i < rows; i++)
    free(A[i]);
  free(A);
  free(x);
  free(y);

  return 0;
}

void print_matrix(double **A, unsigned int rows, unsigned int cols)
{
  printf("\nmatrix:\n");
  unsigned int i, j;
  for (i = 0; i < rows; i++)
  {
    for (j = 0; j < cols; j++)
      printf("%5.5f  ", A[i][j]);
    printf("\n");
  }
  return;
}

void print_vector(double *x, unsigned int rows)
{
  printf("\nvector:\n");
  unsigned int i;
  for (i = 0; i < rows; i++)
    printf("%5.5f\n", x[i]);

  return;
}

void MatMult(double** A, double* x, double* y, int rows, int cols)
```

```
{
  unsigned int i, j;
  for (i = 0; i < rows; i++)
  {
    y[i] = 0.0;
    for (j = 0; j < cols; j++)
      y[i] += A[i][j]*x[j];
  }
  return;
}
```

Matrix with double *:

```
#include <stdio.h>
#include <stdlib.h>

void print_matrix(double *A, unsigned int rows, unsigned int cols);

void print_vector(double *x, unsigned int rows);

void MatMult(double* A, double* x, double* y, int rows, int cols);

int main(void)
{
  unsigned int rows, cols;
  double *A;
  double *x, *y;
  unsigned int i, j;

  printf("\nEnter number of rows: ");
  scanf("%d", &rows);
  printf("\nEnter number of columns: ");
  scanf("%d", &cols);

  // allocate elements of A as contiguous array
  A = (double *)malloc(rows*cols*sizeof(double));

  // allocate space for x and y
  x = (double*)malloc(cols*sizeof(double));
  y = (double*)malloc(rows*sizeof(double));

  //fill A and x with random numbers
  for (i = 0; i < rows*cols; i++)
      A[i] = (double)rand()/(double)RAND_MAX;

  for (j = 0; j < cols; j++)
    x[j] = (double)rand()/(double)RAND_MAX;

  // compute matrix-vector product
  print_matrix(A, rows, cols);
  print_vector(x, cols);
  MatMult(A, x, y, rows, cols);
  print_vector(y, rows);

  // deallocate memory
  free(A);
  free(x);
  free(y);
```

```
  return 0;
}

void print_matrix(double *A, unsigned int rows, unsigned int cols)
{
  printf("\nmatrix:\n");
  unsigned int i, j;
  for (i = 0; i < rows; i++)
  {
    for (j = 0; j < cols; j++)
      printf("%5.5f  ", A[i*rows + j]);
    printf("\n");
  }
  return;
}

void print_vector(double *x, unsigned int rows)
{
  printf("\nvector:\n");
  unsigned int i;
  for (i = 0; i < rows; i++)
    printf("%5.5f\n", x[i]);

  return;
}

void MatMult(double* A, double* x, double* y, int rows, int cols)
{
  unsigned int i, j;
  for (i = 0; i < rows; i++)
  {
    y[i] = 0.0;
    for (j = 0; j < cols; j++)
      y[i] += A[i*rows+j]*x[j];
  }
  return;
}
```