Dirk Praetorius,
Thomas Führer

Sommersemester 2014
11.04.2014

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Write functions `int countValueInRow(int** matrix, int m, int n, int val, int row)` and `int countValueInColumn(int** matrix, int m, int n, int val, int col)` which return the number of entries that equal `val` in the given row `row` resp. column `col` of an $m \times n$-matrix.

Aufgabe 6.2. The Frobenius-norm of a matrix $A \in \mathbb{R}^{m\times n}$ is defined by

$$\|A\|_F := \Big(\sum_{j=1}^{m}\sum_{k=1}^{n} A_{jk}^2\Big)^{1/2}.$$

Write a function `frobeniusnorm` which computes the Frobenius-norm of a given matrix A. Furthermore, write a main program that reads in the dimensions m, n and the matrix A. The matrix should be stored as a dynamic matrix (of type `double**`).

Aufgabe 6.3. Many of the mathematical libraries store matrices $A \in \mathbb{R}^{m\times n}$ columnwise, i.e., in a vector $a \in \mathbb{R}^{mn}$, where $a_{j+km} = A_{jk}$ (the indices start from 0). The *row-sum norm* of a matrix $A \in \mathbb{R}^{m\times n}$ is defined by

$$\|A\| = \max_{j=1,\dots,m} \sum_{k=1}^{n} |A_{jk}|.$$

Write a function `rowsumnorm`, which computes the row-sum norm of a columnwise stored matrix A. Furthermore, write a main program that reads in A and computes $\|A\|$ thereof. Use a dynamic array for the storage of A.

Aufgabe 6.4. Write a function `merge` that joins two arrays $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$, which are sorted in ascending order, into the array $c \in \mathbb{R}^{m+n}$ such that the array c is sorted in ascending order as well, e.g., $a = (1, 3, 3, 4, 7)$ and $b = (1, 2, 3, 8)$ should be joined into $c = (1, 1, 2, 3, 3, 3, 4, 7, 8)$. Use the fact that the arrays a,b are sorted! The input of the function should be a base-pointer to the array c and the length m, n. It should hold $c_j = a_j$ for $j = 0, \dots, m-1$ and $c_j = b_{j-m}$ for $j = m, \dots, m+n-1$, i.e. the array c reads $c = (a, b)$. The input array should be overwritten by the function. You can use a temporary array of length $m + n$ in your function. Furthermore, write a main program that reads in $m, n \in \mathbb{N}$ as well as $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$, and prints out the result $c \in \mathbb{R}^{m+n}$.

Aufgabe 6.5. Write a recursive function `mergesort` that sorts an array a in ascending order and returns the correctly sorted array. Use the following strategy:

- If the length of a is ≤ 2, then sort the array a explicitely.
- If the length of a is > 2, then split a into two arrays b, c of half length. Call the function `mergesort` recursively for b and c, and rejoin the arrays with the function `merge` from Exercise 6.4.

Think of this strategy with help of the example $a = (1, 3, 5, 2, 7, 1, 1, 3)$. Test your program appropriately. *Note:* If the length of a is $2n + 1$ with $n \geq 1$, then a is split into b with length $n + 1$ and c with length n. You might want to use *pointer arithmetics*, i.e. if `a` is an array and `p` is a pointer which contains the address of `a[k]` (i.e. `p = &a[k]`), then `p+n` is the address of `a[k+n]` (i.e. `*(p+n)` coincides with `a[k+n]`). Recall that `a` is the base pointer which contains the address of `a[0]`.

Aufgabe 6.6. As for the contents of variables of elementary type (`double,int,...`), you can print out the content of a pointer with help of `printf`. The place-holder `%p` is used for addresses (which are the contents of pointers!). The output is system-dependent, but mostly in hexadecimal numbers. Write a function `void charPointerDist(char* startaddress, char* endaddress)` that prints out the following three values tabularly:

- Starting address
- End address
- Distance (difference) between both addresses (take care of the place-holder in `printf`!)

Since arrays are stored connectedly, the distance between two successive elements corresponds to the memory used for the specific datatype. Check your function with a `char`-array `c[2]` and the follwoing calls:

```
charPointerAbstand(&c[0],&c[1]);
charPointerAbstand(c,c+1);
```

Then, write a function `void doublePointerDist(double* startaddress, double* endaress)` and test it with a `double`-array. Compare the differences between the results of the two functions.
Optionally: Find out how much memory is used for the types `short`, `int`, and `long` on the `lva.student` server.

Aufgabe 6.7. Where are the errors in the program? Explain why!

```
#include <stdio.h>

void square(double* x)
{
  double* y;
  x=(*y)*(*x);
}

int main(){
  double x=2.1;
  square(&x);
  printf("x^2=%f\n",x);
  return 0;
}
```

Change *only* the function `square`, such that the output of the code is correct.

Aufgabe 6.8. Explain the differences between variables and pointers. What are advantages resp. disadvantages of these?
Write a function `swap` that swaps the contents of two variables `x`, `y`. What is the problem with the following code?

```
void swap(double x, double y)
{
   double tmp;
   tmp = x;
   x = y;
   y = tmp;
}
```