Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
30.11./01.12.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Write a function `int anagram(char* firstStr, char* secondStr)` which checks if a given word is an anagram of a second given word. An anagram of a word is a letter sequence which is generated by permutation, e.g., "Elvis" is an anagram of "lives".
Moreover, write a main program which reads in two words and checks if one is an anagram of the other. If this is the case, let `anagram` return 1. Else, let `anagram` return 0. How did you test your implementation? Save your source code as `anagram.c` into the directory `serie06`.

Aufgabe 6.2. For given values $x_1 < \cdots < x_n$ and function values $y_j \in \mathbb{R}$ linear algebra yields existence of a unique polynomial $p(t) = \sum_{j=1}^{n} a_j t^{j-1}$ of degree $n-1$, which satisfies $p(x_j) = y_j$ for all $j = 1, \dots, n$. For a given $t \in \mathbb{R}$, the so-called *Neville's algorithm* is a method to evaluate $p(t)$ without the calculation of the coefficient vector $a \in \mathbb{R}^n$: For $j, m \in \mathbb{N}$ with $m \geq 2$ and $j + m \leq n + 1$ one defines

$$\begin{aligned} p_{j,1} &:= y_j, \\ p_{j,m} &:= \frac{(t - x_j)p_{j+1,m-1} - (t - x_{j+m-1})p_{j,m-1}}{x_{j+m-1} - x_j}. \end{aligned}$$

Then, it holds that $p(t) = p_{1,n}$. Write a function `neville` with the input parameters $t \in \mathbb{R}$ and $x, y \in \mathbb{R}^n$, which computes $p(t)$ with *Neville's algorithm*. To this end, follow the schematic approach

$$\begin{array}{ccccccccccccc}
y_1 & = & p_{1,1} & \longrightarrow & p_{1,2} & \longrightarrow & p_{1,3} & \longrightarrow & \cdots & \longrightarrow & p_{1,n} & = & p(t) \\
 & & & \nearrow & & \nearrow & & & & \nearrow & & & \\
y_2 & = & p_{2,1} & \longrightarrow & p_{2,2} & & & \nearrow & & & & & \\
 & & & \nearrow & & & & & & & & & \\
y_3 & = & p_{3,1} & \longrightarrow & \vdots & & & & & & & & \\
\vdots & & \vdots & & \vdots & \nearrow & & & & & & & \\
y_{n-1} & = & p_{n-1,1} & \longrightarrow & p_{n-1,2} & & & & & & & & \\
 & & & \nearrow & & & & & & & & & \\
y_n & = & p_{n,1} & & & & & & & & & &
\end{array}$$

The mathematical justification of *Neville's algorithm* will be given in the numerics lecture. At first, build the whole matrix $(p_{j,m})_{j,m=1}^{n}$. Save your source code as `neville.c` into the directory `serie06`. Test your code with a known polynomial p and $y_j = p(x_j)$.

Aufgabe 6.3. For the implementation Neville's algorithm from Exercise **??**, the storage of the complete matrix $(p_{j,m})_{j,m=1}^{n}$ *is unnecessary*. Instead, one can overwrite the values y_j in an appropriate way. In this way, no additional memory is required. Write a function `neville2` in which you implement this improvement. How did you test your implementation? Save your source code as `neville2.c` into the directory `serie06`.

Aufgabe 6.4. Write a library for *columnwise*(!) stored $m \times n$-matrices. Implement the following functions

- `double* mallocmatrix(int m, int n)`
  Allocates memory for a columnwise stored $m \times n$ matrix.
- `double* freematrix(double* matrix)`
  Frees memory of a matrix.
- `double* reallocmatrix(double* matrix, int m, int n, int mNew, int nNew)`
  Reallocates memory and initializes new entries.

Store the signatures of the functions in the header file `dynamicmatrix.h`. Write also appropriate comments to these functions in the header file. The file `dynamicmatrix.c` should contain the implementations of the above functions. Use dynamical arrays. How did you test the correctness of your code?

Aufgabe 6.5. Expand the library from Exercise 6.4 by the following functions.

- `void printmatrix(double* matrix, int m, int n)`
  Prints the column-wise-saved $m \times n$-Matrix on screen. The 2×3-Matrix `double matrix[6]={1,2,3,4,5,6}` shall look like in the following example:

  ```
  1 3 5
  2 4 6
  ```

- `double* scanmatrix(int m, int n)`
  Allocates memory for a matrix and scans the coefficients from keyboard-entry.
- `double* cutOffRowJ(double* matrix, int m, int n, int j)`
  Cuts off the j-th line from a $m \times n$-Matrix.
- `double* cutOffColK(double* matrix, int m, int n, int k)`
  Cuts off the k-th column from a $m \times n$-Matrix.

Use dynamical arrays. How did you test the correctness of your code?

Aufgabe 6.6. The *row-sum norm* of a matrix $A \in \mathbb{R}^{m \times n}$ is defined by

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

Write a function `rowsumnorm`, which computes the row-sum norm of a columnwise stored matrix A. Furthermore, write a main program that reads A from the keyboard and prints the value $\|A\|$ to the screen. How did you test the correctness of your code? What is the computational cost of your function? If the function has a runtime of 0.1 seconds for $n = m = 10^4$, which runtime do you expect for $n = m = 3 \cdot 10^5$? Save your source code as `rowsumnorm.c` into the directory `serie06`..

Aufgabe 6.7. 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. How did you test the correctness of your code? Save your source code as `countValueInRowCol.c` into the directory `serie06`.

Aufgabe 6.8. Implement the well-known game *Tic Tac Toe*, e.g., in the following way:

```
int player=1;
int winner;
int** playboard = newPlayboard();
resetPlayboard(playboard);
while(!isGameFinished(playboard)){ // no winner yet & free squares.
    makeMove(playboard,player);  // read move from the keyboard
                                 // and save it.
    printPlayboard(playboard);   // print the current situation on screen.
    player=changePlayer(player); // change player.
}
winner=getWinner(playboard);
printWinnerMessage(winner);
delPlayboard(playboard);
```

Here, the function `makeMove` should ask the player for his move until it is valid (squares must not be overwritten). Moreover, use Exercise 6.7 for the function `getWinner`. Finally, the function `isGameFinished` may call the function `getWinner`. Save your source code as `tictactoe.c` into the directory `serie06`.