Dirk Praetorius,
Michele Ruggeri

Wintersemester 2015/16
25.-27.11.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 7

Aufgabe 7.1. 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;
}
```

Aufgabe 7.2. The function `squareVec` should square all entries of a given vector $x \in \mathbb{R}^n$, i.e., the input $(-1, 2, 0)$ should be turned into $(1, 4, 0)$. The input vector should be passed as a pointer.

```
#include <stdio.h>

int squareVec(double vec, int n) {
  int j=0;
  for(j=1, j<dim; --j) {
    *vec[j] = &vec[j] * &vec[j];
  }
  return vec;
}

main() {
  double vec[3] = {-1.0,2.0,0.0};
  int j=0;

  squareVec(vec,3);
  for(j=0; j<3; ++j) {
    printf("vec[%d] = %f ",j,vec[j]);
  }
  printf("\n");
}
```

Change *only* the function `squareVec`, such that the `main` programm prints out the correct result. How many errors do you find? What is the computational complexity (Aufwand) of `squareVec`?

Aufgabe 7.3. 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* endaddress)` 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 7.4. Given a differentiable function $f : [a, b] \to \mathbb{R}$ and $x \in [a, b]$, the derivative $f'(x)$ can be approximated by the different quotient

$$\Phi(h) := \frac{f(x + h) - f(x)}{h} \quad \text{for } h > 0.$$

Write a function `double* diff(double x, double h0, double tau, int* n)`, which computes the sequence $\Phi(h_n)$, where $h_n := 2^{-n}h_0$, until

$$|\Phi(h_n) - \Phi(h_{n+1})| \leq \begin{cases} \tau & \text{if } |\Phi(h_n)| \leq \tau, \text{or} \\ \tau\,|\Phi(h_n)| & \text{else.} \end{cases}$$

The function uses a suitable implementation of the object function `double f(double x)` and returns the vector of the complete sequence $(\Phi(h_0), \dots, \Phi(h_n))$, as well as the length of the vector. Save your source code as `diff.c` into the directory `serie07`.

Aufgabe 7.5. An alternative root-finding algorithm is the *Newton method*. Let $f : [a, b] \to \mathbb{R}$. Given an initial guess x_0, define the sequence $(x_n)_{n\in\mathbb{N}}$ via

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Implement the algorithm in a function `newton`. Given x_0 and a tolerance $\tau > 0$, the function performs the Newton iteration until

$$|f'(x_n)| \leq \tau$$

or

$$|f(x_n)| \leq \tau \quad \text{and} \quad |x_n - x_{n-1}| \leq \begin{cases} \tau & \text{for } |x_n| \leq \tau, \\ \tau|x_n| & \text{else.} \end{cases}$$

In the first case, print a warning to inform that the result is presumably wrong. The function uses suitable implementations of the object function `double f(double x)` and its derivative `double fprime(double x)`. Then, write a main program which reads x_0 from the keyboard and returns x_n. Save your source code as `newton.c` into the directory `serie07`.

Aufgabe 7.6. The Newton method from Aufgabe 7.5, besides a function `f` to evaluate the object function, also requires a function `fprime` to evaluate the derivative f' of f. Alternatively, you might replace $f'(x_k)$ with the different quotient $\Phi_h(x_k)$ from Aufgabe 7.4. Realize this idea in a function `newton2(x0,h0,tau)`, which implements the Newton method from Aufgabe 7.5 replacing the derivative $f'(x_k)$ with the result of `diff(xk,h0,tau,n)`. Save your source code as `newton2.c` into the directory `serie07`.

Aufgabe 7.7. 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 7.8. 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 b and c to a sorted field a.

Think about how to rejoin the subarrays to a sorted array a in the second step. Think of this strategy with help of the example $a = (1, 3, 5, 2, 7, 1, 1, 3)$. Test your program appropriately. Moreover, write a main programme in which you read in the array a, sort it with `mergesort` and print the sorted array a. The length of a should be a constant in the main-programme, but `mergesort` should work for arbitrary lengths. Save your source code as `mergesort.c` into the directory `serie07`.