Dirk Praetorius,
Gregor Gantner

Wintersemester 2018/19
29./30.11.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 5

Aufgabe 5.1. Schreiben Sie eine Funktion `scanfpositive`, die vom Benutzer die Eingabe einer positiven Zahl $\tau > 0$ verlangt und diese dann zurückgibt. Die Eingabe soll solange wiederholt werden, bis die eingegebene Zahl $\tau \in \mathbb{R}$ strikt positiv ist, d.h. bei Eingabe einer Zahl $\tau \leq 0$ wird der Benutzer zu erneuter Eingabe aufgefordert. Schreibens Sie weiters ein aufrufendes Hauptprogramm. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `scanfpositive.c` in das Verzeichnis `serie05`.

Aufgabe 5.2. Die Quotientenfolge $(a_{n+1}/a_n)_{n\in\mathbb{N}}$ zur Fibonacci-Folge $(a_n)_{n\in\mathbb{N}}$,

$$a_0 := 1, \quad a_1 := 1, \quad a_n := a_{n-1} + a_{n-2} \quad \text{für } n \geq 2,$$

konvergiert gegen den goldenen Schnitt $(1+\sqrt{5})/2$. Insbesondere konvergiert die Differenz

$$b_n := \frac{a_{n+1}}{a_n} - \frac{a_n}{a_{n-1}}$$

gegen Null. Schreiben Sie eine *nicht-rekursive* Funktion `cauchy`, die zu gegebenem $k \in \mathbb{N}$ die kleinste Zahl $n \in \mathbb{N}$ mit $|b_n| \leq 1/k$ zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Zahl $k \in \mathbb{N}$ einliest und den zugehörigen Index $n \in \mathbb{N}$ ausgibt. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `goldenerSchnitt.c` in das Verzeichnis `serie05`.

Aufgabe 5.3. Die Cosinus-Funktion hat die Darstellung $\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$. Wir betrachten die Partialsummen

$$C_n(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{(2k)!}.$$

Schreiben Sie eine *nicht-rekursive* Funktion `cos_new`, die für gegebene $x \in \mathbb{R}$ und $\tau > 0$ den Wert $C_n(x)$ zurückliefert, sobald

$$|C_n(x) - C_{n-1}(x)|/|C_n(x)| \leq \tau \quad \text{oder} \quad |C_n(x)| \leq \tau$$

gilt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem $x \in \mathbb{R}$ und $\tau > 0$ eingelesen werden. Neben dem berechneten Wert $C_n(x)$ sollen auch der korrekte Wert $\cos(x)$ und der absolute Fehler $|C_n(x) - \cos(x)|$ ausgegeben werden sowie der relative Fehler $|C_n(x) - \cos(x)|/|\cos(x)|$ im Fall $\cos(x) \neq 0$. Schreiben Sie die Funktion möglichst so, dass diese mit einer Schleife auskommt und dass x^{2k} und $(2k)!$ möglichst kostensparend realisiert werden. Man vermeide also insbesondere (vor- oder selbst implementierte) Funktionen zur Berechnung der Potenz oder der Faktoriellen. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `cos.c` in das Verzeichnis `serie05`.

Aufgabe 5.4. Für $x > 0$ konvergiert die Folge

$$x_1 := \frac{1}{2}(1+x), \quad x_{n+1} := \frac{1}{2}\Big(x_n + \frac{x}{x_n}\Big) \quad \text{für } n \geq 1$$

gegen $\sqrt{x}$. Schreiben Sie eine *nicht-rekursive* Funktion `sqrt_new`, die für gegebene $x > 0$ und $\tau > 0$ als Ergebnis das erste Folgenglied $y = x_n$ zurückgibt, für das gilt

$$\frac{|x_n - x_{n+1}|}{|x_n|} \leq \tau \quad \text{oder} \quad |x_n| \leq \tau.$$

Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem x eingelesen und neben der Approximation x_n von $\sqrt{x}$ auch der exakte Wert sowie der absolute Fehler $|x_n - \sqrt{x}|$ ausgegeben werden. Speichern Sie den Source-Code unter `sqrt_new.c` in das Verzeichnis `serie05`. Vergleichen Sie Ihre Implementierung mit Ihrem Code aus Aufgabe 3.7. Diskutieren Sie Vor- und Nachteile der beiden Implementierungen!

Aufgabe 5.5. Schreiben Sie eine Funktion `eratosthenes`, die das *Sieb des Eratosthenes* realisiert. Dies ist ein Algorithmus, mit dem alle Primzahlen bis zu einer bestimmten Zahl `nmax` berechnet werden können (benannt nach dem griechischen Mathematiker Eratosthenes). Der Algorithmus sieht folgendermaßen aus:

- Man legt eine Liste (Vektor) $\texttt{prim} = (2, \ldots, \texttt{nmax}) \in \mathbb{N}^{\texttt{nmax}-1}$ an.
- Man streicht aus der Liste alle Vielfachen der ersten Zahl (also der Zahl 2).
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfachen.

Am Ende sollen alle Primzahlen von $2, \ldots,$`nmax` ausgegeben werden. Geben Sie außerdem die Anzahl der gefundenen Primzahlen mit aus. Realisieren Sie das Streichen, indem Sie die entsprechenden Einträge auf 0 setzen. Die Zahl `nmax` soll eine Konstante im Hauptprogramm sein. Speichern Sie den Source-Code unter `eratosthenes.c` in das Verzeichnis `serie05`.

Aufgabe 5.6. Schreiben Sie eine Funktion `primfaktoren`, die für eine natürliche Zahl $n \in \mathbb{N}$ deren Primfaktoren bestimmt und in der Konsole ausgibt. Für die Primfaktoren p_k gilt $n = \prod_{j=1}^{k} p_k$. Um eine Liste aller möglichen Primfaktoren zu erhalten, verwende man das Sieb des Eratosthenes aus Aufgabe **??**. Speichern Sie den Source-Code unter `primfaktoren.c` in das Verzeichnis `serie05`.

Aufgabe 5.7. Genauso wie der Inhalt von Variablen elementaren Datentyps kann auch der Inhalt eines Pointers mittels `printf` ausgegeben werden. Man verwendet hier `%p` als Platzhalter für Adressen. Die Ausgabe dafür erfolgt systemabhängig meist in Hexadezimaldarstellung. Schreiben Sie eine Funktion `void charPointerAbstand(char* anfangsadresse, char* endadresse)`, welche folgende drei Werte tabelliert:

- Anfangsadresse
- Endadresse
- Abstand (Differenz) der beiden Adressen (Platzhalter im `printf` beachten!)

Da Arrays zusammenhängend im Speicher liegen, entspricht der Abstand zweier aufeinanderfolgender Elemente genau dem Speicherverbrauch des entsprechenden Datentyps. Testen Sie Ihre Funktion für ein `char`-Array `c[2]` mit den beiden Aufrufen:

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

Schreiben Sie nun nach obiger Manier eine Funktion `void doublePointerAbstand(double* anfangsadresse, double* endadresse)`, testen diese mit einem `double`-Array und vergleichen die unterschiedlichen Ergebnisse. Finden Sie weiters heraus, wieviel Speicher die Typen `short`, `int` und `long` auf dem Übungsserver verbrauchen.

Aufgabe 5.8. Wo liegen die Fehler im folgenden Programm?

```
#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;
}
```

Verändern Sie *nur* die Funktion `square`, so dass der Output des Codes den Erwartungen entspricht.