Technische Universität München
Fakultät für Informatik, I-14
Prof. Harald Räcke
Robin Münk

SoSe 2025
Übungsblatt 5
23. Mai 2025

Algorithmen und Datenstrukturen

Aufgabe 1 Rekursion

a) Folgendes Produkt definiert eine Zahlenfolge:

$$C_n = \prod_{k=1}^{n} \frac{4k-2}{k+1}$$

(i) Stellen Sie C_n in einer Rekursionsformel dar, d.h. führen Sie C_n auf den Wert C_{n-1} zurück!

(ii) Schreiben Sie einen *rekursiven* Algorithmus in Pseudocode zur Berechnung von C_n

b) Gegeben sei folgende Rekursionsgleichung zur Annäherung von π:

$$f_n = \begin{cases} \frac{4}{2n+1} + f_{n-1} & n \text{ gerade} \\ \frac{-4}{2n+1} + f_{n-1} & n \text{ ungerade} \end{cases}$$

wobei $f_0 = 4$.

Schreiben Sie ein C/C++ Programm, das nach Eingabe des Index n die Annäherung f_n für π *rekursiv* berechnet und ausgibt. Sie können das Testing Framework verwenden.

Beispiellösung:

a) (i) $C_0 = 1$
$C_n = (4n-2)/(n+1) * C_{n-1}, \quad n \geq 1$

(ii)
```
catalan(n):
    if (n == 0)
        return 1
    return catalan(n-1) * (4*n-2) / (n+1)
```

b)
```
float pi (int n) {
    if (n == 0)
        return 4;
    return (4 - 8 * (n % 2)) / (2 * n + 1.0) + pi(n - 1);
}
// +/-4 == (4-8*(n%2))
```

Aufgabe 2 Heron-Verfahren

Eine Möglichkeit, eine Approximation für die Quadratwurzel einer positiven Zahl $a > 0$ zu berechnen, ist mithilfe des Heron-Verfahrens (auch babylonisches Wurzelziehen genannt). Es ist definiert durch die Iterationsvorschrift

$$x_0 = \frac{a+1}{2}$$
$$x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{a}{x_n} \right).$$

Diese Iterationsvorschrift wird solange angewendet, bis die Änderung kleiner als eine Konstante ε (z.B. $\varepsilon = 10^{-7}$) ist, also bis $|x_n - x_{n+1}| \leq \varepsilon$. Die Annäherung der Quadratwurzel von a ist dann als x_{n+1} gegeben.

Schreiben Sie in Pseudocode einen iterativen Algorithmus (mit Schleife) und einen rekursiven Algoithmus (ohne Schleifen), der die Quadratwurzel der Eingabe mit diesem Verfahren berechnet.

Beispiellösung:

Die iterative Funktion verwendet eine do...while Schleife.

```
WurzelIterativ(a):
    x_current = (a+1) / 2
    do
        x_old = x_current
        x_current = (x_old + a / x_old) / 2
    while (|x_old - x_current| > eps)
    return x_current
```

Für die rekursive Funktion wird eine Hilfsfunktion benötigt, um die initialen Parameter richtig zu setzen.

```
WurzelRekursiv(a):
    return Hilfsfunktion(a, 0, (a + 1) / 2)

Hilfsfunktion(a, x_old, x_current):
    if (|x_old - x_current| <= eps)
        return x_current
    x_old = x_current
    x_current = (x_old + a / x_old) / 2
    return Hilfsfunktion(a, x_old, x_current)
```

Aufgabe 3 (P) Binomialkoeffizienten

Der Binomialkoeffizient für gegebene integer $0 \leq k \leq n$ ist definiert als

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad .$$

Sie können für diese Aufgabe das Testing Framework verwenden.

a) Schreiben Sie eine C/C++ Funktion, die den Binomialkoeffizienten für gegebene n und k gemäß der Definition *iterativ* (mit Schleifen) berechnet!

b) Für die Binomialkoeffizienten gelten außerdem die folgenden Gleichungen.

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad \text{für } n \geq 0$$
$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \qquad \text{für } n > 0 \text{ und } 0 \leq k < n$$

Nutzen Sie diese Formeln und schreiben Sie eine C/C++ Funktion, die den Binomialkoeffizienten für gegebene n und k *rekursiv* (ohne Schleifen) berechnet!

Beispiellösung:

Der Code ist in der Datei `blatt5solution.cpp` zu finden.

Aufgabe 4 (P) Endrekursion

In dieser Aufgabe sei % der Modulo-Operator und / der Operator für ganzzahlige Division. D. h. $12345/10 = 1234$ und $12345\%10 = 5$. Sie können für diese Aufgabe das Testing Framework verwenden.

Gegeben ist die folgende Funktion $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

$$g(x,y) = \begin{cases} x^y & x < 10 \\ (x\%10)^y + g(x/10, y+1) & x \geq 10 \end{cases}$$

- Implementieren Sie die Funktion g als eine *endrekursive* C/C++ Methode. Sie benötigen dazu eine Hilfsfunktion. Überlegen Sie zunächst, welche und wie viele Parameter diese Hilfsfunktion benötigt.
- Implementieren Sie die Funktion g als eine *iterative* C/C++ Methode, indem Sie die Endrekursion entfernen, analog der Vorgehensweise die in der Vorlesung für die Funktion *ggT* gezeigt wurde.

Beispiellösung:

Der Code ist in der Datei `blatt5solution.cpp` zu finden.