

Aufgabe 1. Das folgende Programm sollte die Fakultätsfunktion implementieren und $5! + 16!$ auf der Konsole ausgeben. Dabei haben wir 6 Fehler dabei gemacht. Schnappen Sie sich alle!

```

1 #include <iostream>
2
3 int fakultaet (n)
4 {
5     int ergebnis = 0;    /* speichert die Fakultatet */
6
7     while (n > 0)        /* verkleinere n, bis es */
8         ergebnis *= n;  /* null ist und multi- */
9         n--;            /* pliziere mit ergebnis */
10    return ergebnis;
11 }
12
13 int main ()
14 {
15     int add2fak;
16     const int a = 5;
17     const int b = 16;
18
19     add2fak = fakultaet (a) + fakulatet (b);
20     std::cout << a << "!" + " << b "!" = " << add2fak << std::endl;
21     return 0,
22 }
```

Was passiert, wenn man nach der Korrektur **b** statt mit 16 mit 17 initialisiert?

Aufgabe 2. Implementieren Sie den Cosinus über seine Reihendarstellung mit einer **for**-Schleife. Sie können die Formel bei Wikipedia nachschlagen, sie selbst entwickeln oder diese hier verwenden:

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \cdot \frac{x^{2k}}{(2k)!}$$

Aufgabe 3. Implementieren Sie den Absolutbetrag **betrag(x)**, **cos(x)** und die Wurzelfunktion **wurzel(x)** (mit dem Heron-Verfahren, Algorithmus 2 vom ersten Zettel) als Funktionen und lagern Sie sie in ein eigenes Modul aus.

Aufgabe 4. a) Implementieren Sie für $x \in \mathbb{R}$ und $n \in \mathbb{N}$ eine Potenzfunktion $x^n = \text{power}(x, n)$ mit der Double-and-Add-Methode:

$$\text{power}(x, n) = \begin{cases} 1 & \text{wenn } n = 0 \\ x \cdot \text{power}\left(x^2, \frac{n-1}{2}\right) & \text{wenn } n \text{ ungerade} \\ \text{power}\left(x^2, \frac{n}{2}\right) & \text{wenn } n \text{ gerade} \end{cases}$$

zuerst mal rekursiv.

- b) Implementieren Sie eine Potenzfunktion **naiv_power(x, n)**, indem Sie eine Schleife von 1 bis n laufen lassen und bei jedem Durchlauf eine mit 1 Initialisierte Variable mit x multiplizieren (ähnlich wie in der Vorlesung). Berechnen Sie $0,9999999999^{2000000000}$ einmal mit **power(x, n)** von oben und einmal mit **naiv_power(x, n)** (es sollte ca. 0,818731 raus kommen).
- c) Implementieren Sie die Double-and-Add-Methode mit einer Schleife, also ohne rekursiven Aufruf.
- d) Welches dieser Verfahren ist am langsamsten? Warum?

Aufgabe 5. Die Bell-Zahlen $(B_n)_{n \in \mathbb{N}}$ (die natürlichen Zahlen enthalten bei uns stets die 0) sind wie folgt rekursiv definiert: $B_0 = 1$ und

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

für $n \in \mathbb{N}$. Implementieren Sie eine rekursive Funktion, die zu gegebenem n den Wert B_n ausrechnet.