Informatik I: Scheme-Programmieraufgaben zum Selbststudium

3. Rekursive Prozeduren
(Bei Fragen können Sie sich an die Tutoren des betreuten Programmierens wenden)

Aufgabe 1: Einfache rekursive Prozeduren

a) Schreiben Sie eine Prozedur `(: fac (natural -> natural))`, die die Fakultätsfunktion $n!$ nach folgender Vorschrift berechnet:

$$n! = \begin{cases} 1 & \text{falls } n = 0 \\ n \cdot (n-1)! & \text{sonst} \end{cases}$$

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (fac 5) 120)
(check-expect (fac 10) 3628800)
```

b) Schreiben Sie eine Prozedur `(: fib (natural -> natural))`, die die Fibonacci-Zahlen nach folgender Vorschrift berechnet:

$$\text{fib}(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{sonst} \end{cases}$$

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (fib 6) 8)
(check-expect (fib 13) 233)
```

c) Schreiben Sie eine Prozedur `(: ggt (natural natural -> natural))`, die den größten gemeinsamen Teiler zweier Zahlen nach folgender Vorschrift (euklidischer Algorithmus) berechnet:

$$\text{ggt}(m, n) = \begin{cases} m & \text{falls } n = 0 \\ \text{ggt}(n, m \text{ modulo } n) & \text{sonst} \end{cases}$$

Den Rest m modulo n bei ganzzahliger Division von m durch n können Sie mit Hilfe der eingebauten Funktion `(: modulo (integer integer -> integer))` bestimmen.

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (ggt 26741 5681) 13)
(check-expect (ggt 46410 641886) 714)
```

d) Schreiben Sie eine Prozedur `(: nCk (natural natural -> natural))`, wobei `(nCk n k)` den Binomialkoeffizient $\binom{n}{k}$ nach folgender Vorschrift berechnet:

$$\binom{n}{k} = \begin{cases} 1 & \text{falls } k = 0 \\ 0 & \text{falls } k > 0 \text{ und } n = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sonst} \end{cases}$$

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (nCk 4 2) 6)
(check-expect (nCk 5 3) 10)
```

e) Schreiben Sie eine Prozedur `(: countDigits (natural -> natural))`, die die Anzahl der Stellen einer natürlichen Zahl in Dezimaldarstellung zählt. Benutzen Sie hierfür die eingebaute Prozedur für ganzzahlige Division `(: quotient (integer integer -> integer))`. `(countDigits 0)` soll per Definition zu `0` ausgewertet werden.

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (countDigits 123456789022345) 15)
(check-expect (countDigits 12345678902234567890323456789O) 30)
```

f) Schreiben Sie eine Prozedur `(: countTrailingZeros (natural -> natural))`, die die Anzahl der Nullen am Ende der Dezimaldarstellung einer natürlichen Zahl zählt. Benutzen Sie hierfür die eingebauten Prozeduren für ganzzahlige Division `(: quotient (integer integer -> integer))` und für das Divisionsmodulo `(: modulo (integer integer -> integer))`. Der Fall `(countTrailingZeros 0)` sei undefiniert, für diesen darf die Prozedur jeden beliebigen Wert zurückgeben oder in eine Endlosschleife gehen.

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (countTrailingZeros 100203400000) 5)
(check-expect (countTrailingZeros 109001) 0)
```

Aufgabe 2: Endrekursion

Schreiben Sie Prozeduren `(: fac-iter (natural -> natural))` und `(: fib-iter (natural -> natural))`, die die in den Aufgaben (1.a) und (1.b) definierten Fakultäts- und Fibonaccifunktionen endrekursiv berechnen.

Zur Kontrolle können Sie folgende Testfälle verwenden:
```
(check-expect (countFinishingZeros (fac-iter 20000)) 4999)
(check-expect (countFinishingZeros (fac-iter 15000)) 3748)
(check-expect (countDigits (fib-iter 1000)) 209)
(check-expect (countDigits (fib-iter 10000)) 2090)
```