2. Übung ALP 1: Funktionale Programmierung WS 14/15

Klaus Kriegel **Abgabe: 04.11.2014, 10:00 Uhr**

Aufgabe 1: **k-näre Zahldarstellung** $(2 + 2 + 2$ Punkte$)$

a) Stellen Sie die Zahl 1375 im Binär-, Oktal- und Hexadezimalsystem (d.h. Basis $k = 2, k = 8$ und $k = 16$) dar.

b) Übertragen Sie die Schulmethoden zur Addition und Multiplikation von Dezimalzahlen auf andere Systeme. Stellen Sie dazu die Zahlen 14 und 33 im Binärsystem dar, führen Sie die Addition und Multiplikation dieser beiden Zahlen aus und überprüfen Sie die Korrektheit der Ergebnisse.

c) Führen Sie die Berechnungen aus Teilaufgabe b) noch einmal im System mit der Basis $k = 5$ aus.

Aufgabe 2: **Algorithmen in Pseudocode I** $(2 + 1 + 2$ Punkte$)$

a) Beschreiben Sie einen Algorithmus zur Berechnung des abgerundeten Logarithmus $\lfloor \log_k n \rfloor$ für natürliche Zahlen $k \geq 2$ und $n \geq 2$ als Pseudocode (in Anlehnung an die Beispiele im Vorlesungsskript). Sie können dazu die Operation / für die ganzzahlige Division (also mit Abrunden) verwenden.

b) Überprüfen Sie die Korrektheit der Implementierung an den Beispielen $\lfloor \log_2 14 \rfloor$ und $\lfloor \log_3 20 \rfloor$.

c) Modifizieren Sie den Algorithmus, um $\lceil \log_k n \rceil$ zu bestimmen, wobei aber keine direkte Operation zur aufgerundeten Division zur Verfügung steht.

Aufgabe 3: **Algorithmen in Pseudocode II** $(2 + 1$ Punkte$)$

Verwenden Sie die Prozeduren `primtest(n)` und `teilbarkeit(n,k)` aus dem Skript, um Algorithmen für die folgenden Problemstellungen zu beschreiben:
a) grQuadratTeiler(n) zur Berechnung der gößten Quadratzahl, die `n` teilt.
b) grPrimQuadratTeiler(n) zur Berechnung der größten Quadrats einer Primzahl, das `n` teilt.

Aufgabe 4: **Pseudocode verstehen** (6 Punkte)

Betrachten Sie die folgenden Prozeduren und erkennen Sie, was dabei eigentlich berechnet wird. Beschreiben Sie das Ergebnis mit einer kurzen Formel oder mit einem einfachen Satz und begründen Sie die Antwort.

```
magic1(n):  n∈ ℕ
k=0

while (n > 0)
      k = 2-k
      n = n-1
return k
```

```
magic2(x):  x∈ ℕ
k=1

while (x > 1)
      k = 3*k
      x =⌈x/3⌉
return k
```

```
magic3(n,m):  n,m∈ ℕ
k=min(n,m)
l=max(n,m)
p = 0
if (l-k <= 1) then p = k
    else p = magic3(k+1,l-1)
return p
```