Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Prüfungsaufgabe A vom 16.02.2004

Eine $n \times n$ Matrix A ist ein quadratisches Zahlenschema der Form

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

Das Produkt zweier Matrizen wird durch folgende Definition beschrieben:

$$AB = (c_{ij}) \text{ mit } c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad i = 1, \dots, n \quad j = 1, \dots, n$$

Beispiel:

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 8 & 10 \\ 7 & 8 \end{pmatrix}$$

Zur Berechnung eines Elements c_{ij} der Ergebnismatrix multipliziert man also paarweise die Elemente der i-ten Zeile von A und der j-ten Spalte von B und addiert die entstehenden Produkte.

Beispiel:

$$c_{12} = 1 * 2 + 2 * 4 = 10$$

oder anders dargestellt:

$$\begin{pmatrix} 2 & \boxed{2} \\ 3 & \boxed{4} \end{pmatrix}$$

$$\begin{pmatrix} \boxed{1} & \boxed{2} \\ 2 & 1 \end{pmatrix} \quad \begin{pmatrix} 8 & \boxed{10} \\ 7 & 8 \end{pmatrix}$$

In Scheme kann man eine Matrix etwa durch folgende Listenstruktur darstellen:

```
((a11 ... a1n)
 ...
 (an1 ... ann))
```

Implementieren Sie nun eine Prozedur `matrix-mul`, die das Produkt zweier $n \times n$ Matrizen berechnet.
Gehen Sie dazu wie folgt vor:

- Die Prozedur `(matrix-col j matrix)` liefert die j-te Spalte der Matrix `matrix` zurück.

```
(define bsp-matrix (list '(1 2 3)
                         '(4 5 6)
                         '(7 8 9)))

(matrix-col 3 bsp-matrix) => (3 6 9)
```

- `(skalar-prod row col)` berechnet ein Element c_{ij} der Ergebnismatrix, wenn `row` die i-te Zeile der ersten und `col` die j-te Spalte der zweiten Matrix ist. `row` und `col` sind einfache Listen.

```
(skalar-prod '(4 5 6) '(3 6 9)) = > 96
```

- `(line-mul a b)` bekommt eine Zeile `a` und eine Matrix `B` übergeben. Die Prozedur soll, unter Verwendung von `matrix-col` und `skalar-prod`, die Multiplikation der Zeile `a` mit der Matrix `B` durchführen.

```
(line-mul '(4 5 6) bsp-matrix) => (66 81 96)
```

- `(matrix-mul a b)` erstellt nun, mit Hilfe von `line-mul`, Zeile für Zeile die Ergebnis-Matrix.

```
(matrix-mul bsp-matrix bsp-matrix)
=> ((30 36 42) (66 81 96) (102 126 150))
```

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**

Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Prüfungsaufgabe B vom 16.02.2004

Eine $n \times n$ Matrix A ist ein quadratisches Zahlenschema der Form

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

Die transponierte Matrix A^T von A ist gleich (a_{ji})

Beispiel:

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 2 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \text{ dann ist } A^T = \begin{pmatrix} 1 & 2 & 7 \\ 3 & 5 & 8 \\ 4 & 6 & 9 \end{pmatrix}$$

Eine symmetrische Matrix, ist eine Matrix bei der $A = A^T$ gilt.

Eine Matrix kann man in Scheme durch folgende Listenstruktur darstellen:

```
((a11 ... a1n)
 ...
 (an1 ... ann))
```

Ihre erste Aufgabe ist es nun, eine Prozedur `(transpose matrix)` zu schreiben, die die Matrix `matrix` transponiert zurück gibt.
Dazu schreiben Sie am besten zwei Prozeduren:

- Als erstes konstruieren Sie eine Prozedur `(matrix-ref i j matrix)`. Diese Prozedur soll das Element in der i-ten Zeile und der j-ten Spalte der Matrix `matrix` zurückgeben.

```
(define bsp-matrix (list '(1 2 3)
                         '(4 5 6)
                         '(7 8 9)))
(matrix-ref 2 3 bsp-matrix) => 6;
```

- Im zweiten Schritt konstruieren Sie die Prozedur `(transpose matrix)`, die die transponierte Matrix erzeugt.

```
(transpose bsp-matrix)
=> ((1 4 7) (2 5 8) (3 6 9))
```

Die zweite Aufgabe besteht darin, eine Prozedur `(symmetric? A)` zu schreiben, die mit Hilfe der Prozedur `transpose` überprüft, ob die Matrix `A` symmetrisch ist oder nicht.

```
(symmetric? bsp-matrix) => #f
```

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**

Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Prüfungsaufgabe C vom 17.02.2004

Eine $n \times n$ Matrix A ist ein quadratisches Zahlenschema der Form

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

Um die Determinante einer $n \times n$ Matrix A, mit $n > 1$ zu berechnen, kann man folgende Formel verwenden:

$$\det A := |A| := \sum_{j=1}^{n} (-1)^{1+j} \cdot a_{1j} \cdot \det(A_{1j})$$

A_{1j} ist dabei eine $(n-1) \times (n-1)$ Matrix, die durch Streichen der ersten Zeile und der j-ten Spalte von `A` entsteht.

Im Fall $n = 1$ gilt:

$$A = (a) \text{ und } \det(A) = a$$

Beispiel:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} =$$

$$1 * \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 2 * \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3 * \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} =$$

$$1 * (5 * 9 - 6 * 8) - 2 * (4 * 9 - 6 * 7) + 3 * (4 * 8 - 5 * 7) =$$

$$-3 + 12 - 9 = 0$$

In Scheme kann man ein Matrix etwa durch folgende Listenstruktur darstellen:

```
((a11 ... a1n)
 ...
 (an1 ... ann))
```

Schreiben Sie nun eine Prozedur `(det matrix)`, die die Berechnung der Determinante nach obiger Formel ausführt.
Gehen Sie dabei wie folgt vor.

- Die erste Prozedur `(delete i liste)` soll die Liste `liste` ohne das i-te Element zurückgeben.

  ```
  (delete 2 '(1 2 3 4)) => (1 3 4)
  ```

- Schreiben Sie nun eine Prozedur `(delete-ij i j matrix)` analog zu der Prozedur `delete`, die mit Hilfe von `delete` die i-te Zeile und j-te Spalte aus der Matrix streicht.

  ```
  (define bsp-matrix (list '(1 2 3)
                           '(4 5 6)
                           '(7 8 9)))

  (delete-ij 2 3 bsp-matrix) => ((1 2) (7 8))
  ```

- Schreiben Sie eine dritte Prozedur `(det matrix)`, die die Determinante der Matrix `matrix` berechnet. Benutzen Sie dazu die Prozedur `matrix-ref` aus dem Teachpack `matrix.ss` und die von Ihnen programmierte `delete-ij` Prozedur.

  ```
  (det bsp-matrix) => 0
  ```

- Im Teachpack `matrix.ss` steht die Prozedur `(matrix-ref i j matrix)` zur Verfügung. Diese Prozedur gibt das Element zurück, das in der `i`-ten Zeile und der `j`-ten Spalte der Matrix `matrix` steht.

- Mittels der eingebauten Prozedur `(expt a b)` können Sie a^b berechnen.

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**

Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Prüfungsaufgabe D vom 17.02.2004

Die nach dem Schweizer Mathematiker Gabriel Cramer benannte Cramersche Regel berechnet die eindeutige Lösung eines linearen Systems $Ax = b$ mit n Unbekannten und n Gleichungen. Dabei ist A eine $n \times n$ Matrix, x und b sind Spaltenvektoren.

Beispiel:

$$\begin{aligned} x_1 + 0x_2 + 2x_3 &= 6 \\ -3x_1 + 4x_2 + 6x_3 &= 30 \\ -x_1 - 2x_2 + 3x_3 &= 8 \end{aligned}$$

$$A = \begin{pmatrix} 1 & 0 & 2 \\ -3 & 4 & 6 \\ -1 & -2 & 3 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \text{ und } b = \begin{pmatrix} 6 \\ 30 \\ 8 \end{pmatrix}$$

Die Lösung x wird dabei durch folgende Formel berechnet, wobei $\det(A)$ die Determinante der Matrix A bezeichnet:

$$x_1 = \frac{\det(A_1)}{\det(A)},\ x_2 = \frac{\det(A_2)}{\det(A)},\ \ldots,\ x_n = \frac{\det(A_n)}{\det(A)}$$

Dabei entsteht die Matrix A_j durch Ersetzen der j-ten Spalte von A durch die Spalte

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Fortsetzung des Beispiels:

A_1 wird somit zu:

$$A_1 = \begin{pmatrix} 6 & 0 & 2 \\ 30 & 4 & 6 \\ 8 & -2 & 3 \end{pmatrix}$$

Eingesetzt in die Formelt ergibt dies für x_1:

$$x_1 = \frac{\det(A_1)}{\det(A)} = -\frac{10}{11}$$

Die $n \times n$ Matrix

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nn} \end{pmatrix}$$

kann man in Scheme etwa durch die folgende Listenstruktur darstellen:

```
((a11 ... a1n)
 ...
 (an1 ... ann))
```

Ihre Aufgabe ist es nun, die Cramersche Regel zu implementieren.

- Die Funktion `(det A)`, die die Determinate der Matrix A berechnet, ist im Teachpack `det.ss` gegeben und muss geladen werden.
- Schreiben sie ein Prozedur `list-replace` mit den Parametern `liste`, `elem` und `j`. Diese Prozedur soll eine Kopie der Liste `liste` zurückgeben, bei der jedoch das j-te Element durch `elem` ersetzt ist.

```
(list-replace '(1 2 3 4 5) 10 2) => (1 10 3 4 5)
```

- Eine zweite Prozedur `(matrix-replace-col matrix col j)`, soll analog zu `list-replace` die Spalte `j` durch die Spalte `col` in der Matrix `matrix` ersetzen.

```
(define bsp-matrix (list '(1 0 2)
                         '(-3 4 6)
                         '(-1 -2 3)))

(matrix-replace-col bsp-matrix '(1 1 1) 3)
=> ((1 0 1) (-3 4 1) (-1 -2 1))
```

- Zu guter letzt soll die Prozedur `(cramer A b)` die eindeutige Lösung für $Ax = b$ nach der Cramerschen Regel berechnen.

```
                                      10    7    5
(cramer bsp-matrix '(6 30 8)) =>  (- --  1--  3--)
                                      11   11   11
```

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**

Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Programmieren I
WS 2003/2004

Prüfungsaufgabe E vom 29.3.2004

Die *Chebyshev*-Polynome sind durch

$$\begin{aligned} C_0(x) &:= 1 \\ C_1(x) &:= x \\ C_n(x) &:= 2xC_{n-1}(x) + C_{n-2}(x) \text{ für } n \geq 2 \end{aligned}$$

und die *Legendre*-Polynome durch

$$\begin{aligned} P_0(x) &:= 1 \\ P_1(x) &:= x \\ P_n(x) &:= \frac{(2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)}{n} \text{ für } n \geq 2 \end{aligned}$$

definiert. Schreiben Sie eine Scheme Funktion `chebyshev n` und `legendre n`, die zu gegebenm $n \geq 0$ das *Chebychev*-Polynom C_n bzw. das *Legendre*-Polynom P_n als Scheme Ausdruck zurück liefert. Die Polynom-Variable soll jeweils das Symbol `x` sein.
Bei den folgenden Anwendungsbeispielen beachten Sie bitte, dass es durchaus sein kann, dass die von Ihnen geschriebene Scheme Funktionen andere Ergebnisse liefern können; so können etwa Summanden vertauscht sein. Entscheidend ist aber, dass die von den Ausdrücken repräsentierten Polynome übereinstimmen.

```
(chebyshev 0) =>   1
(chebyshev 1) =>   x
(chebyshev 2) =>   (+ (* 2 x x) 1)
(chebyshev 3) =>   (+ (* 2 x (+ (* 2 x x) 1)) x)
```

```
(legendre 0) =>  1
(legendre 1) =>  x
(legendre 2) =>  (/ (- (* 3 x x) (* 1 1)) 2)
```

Die Scheme-Funktion `eval` wertet einen als Parameter gegebenen Scheme-Ausdruck (im momentan aktuellen Environment) aus. So liefert etwa `(eval '(* 3 7))` den Wert `21` oder aber `(eval '(car '(1 2 3)))` oder auch `(eval (list 'car '(list 1 2 3)))` den Wert `1`. Benutzen Sie diese Funktion, um den Wert eines durch die Prozedur `legendre` erzeugten Legendre-Polynoms an einer Stelle `a` zu berechnen. Schreiben Sie dazu eine Prozedur `legendre-wert` mit Parameter `n` und `a`, die den Wert von $P_n(a)$ bestimmt.
Beispiele:

```
(legendre-wert 0 3)  =>  1
(legendre-wert 1 3)  =>  3
(legendre-wert 2 3)  =>  13
(legendre-wert 3 3)  =>  63
```

Hinweis: Schreiben Sie zunächst eine Prozedur `leg-w-help` mit einem Parameter `n`. `leg-w-help` liefert eine Prozedur mit einem Parameter zurück, die den Wert des n-ten Legendre-Polynoms $P_n(x)$ berechnet. Dazu erzeugen Sie die Liste `(lambda (x) <n-tes Legendre-Polynom>)` und wenden Sie darauf `eval` an.
Beispiele:

```
(leg-w-help 2)  => #<prozedure>
((leg-w-help 2) 3)  =>  13
```

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**

Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Programmieren I
WS 2003/2004

Prüfungsaufgabe F vom 29.3.2004

Eine Scheme-Funktion zur Berechnung des größten gemeinsamen Teilers (ggT) zweier Zahlen $m, n \in \mathbb{N}$ ist im folgenden angegeben:

```
(define (ggT m n)
  (if (= 0 n)
    m
    (ggT n (remainder m n))))
```

Dieser Algorithmus greift auf die Scheme-Funktion `remainder` zurück, die den Rest bei der Division des ersten Arguments durch das zweite berechnet. Es besteht jedoch auch die Möglichkeit den ggT ohne Benutzung von `remainder` zu berechnen. Die folgenden (für $m, n \in \mathbb{N}$ gültigen) Formeln liefern hierfür einen Algorithmus:

$$ggT(m,n) = \begin{cases} 2 \cdot ggT(\frac{m}{2}, \frac{n}{2}) & \text{wenn } m \text{ und } n \text{ gerade} \\ ggT(\frac{m}{2}, n) & \text{wenn } m \text{ gerade und } n \text{ ungerade} \\ ggT(m, \frac{n}{2}) & \text{wenn } m \text{ ungerade und } n \text{ gerade} \\ ggT(m, \frac{n-m}{2}) & \text{wenn } m \text{ und } n \text{ ungerade und } n > m \\ ggT(\frac{m-n}{2}, n) & \text{wenn } m \text{ und } n \text{ ungerade und } m > n \\ m & \text{wenn } m = n \end{cases}$$

- Schreiben Sie ein Scheme-Funktion `ggT` zur Berechnung des ggT zweier Zahlen $m, n \in \mathbb{N}$. An arithmetischen Operationen dürfen Sie hierbei lediglich `links-shift` (*verdoppeln*), `rechts-shift` (*halbieren und auf nächste ganze Zahl abrunden*) und `minus` (*subtrahieren*) benutzen. Diese Hilfsfunktionen können Sie wie folgt implementieren:

  ```
  (define (links-shift x) (* x 2))
  (define (rechts-shift x) (quotient x 2))
  (define (minus x y) (- x y))
  ```

- Schreiben Sie eine zweite Version von `ggT` die einen *iterativen* Prozess erzeugt. Hierzu dürfen Sie zusätzlich die arithmetischen Operationen `erhoehe` (*erhöhen um 1*) und `vermindere` (*vermindern um 1*) verwenden.

  Implementieren Sie diese Funktionen wie folgt:

```
(define (erhoehe n) (+ n 1))
(define (vermindere n) (- n 1))
```

  Hinweis: Merken Sie sich in einem zusätzlichen Parameter, wie oft am Ende das Zwischenergebnis noch mit 2 multipliziert werden muss, um das Endergebnis zu erhalten. Diese Multiplikation könnte dann von einer von Ihnen zu schreibenden Hilfsprozedur `links-shift-n` ausgeführt werden, die die Prozedur `links-shift` so oft auf das erste Argument anwendet, wie das zweite Argument angibt und nur die in dieser Aufgabe erlaubten arithmetischen Operationen benutzt.

  Beispiele:

```
(links-shift-n 3 0)  =>  3
(links-shift-n 3 2)  =>  12

(ggT 64 1)  =>  1
(ggT 1 64)  =>  1
(ggT 64 32)  =>  32
(ggT 123456 789)  =>  3
```

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**

Universität Hannover
Institut für Informationssysteme
Fachgebiet Programmiersprachen
und Übersetzer
Prof. Dr. R. Parchmann

Programmieren I
WS 2003/2004

Prüfungsaufgabe G vom 29.3.2004

Eine naheliegende Möglichkeit zur Darstellung von Mengen in Scheme ist eine Liste der Elemente der Menge. Eine Menge wie z.B. {2 3 5} wird hierbei etwa als eine Liste `(2 3 5)` dargestellt. Die Reihenfolge der Listenelemente ist dabei ohne Bedeutung, aber es darf natürlich kein Element in der Liste mehrfach auftreten. Analog ist eine Menge von Mengen eine Liste von Listen. Diese Darstellung einer Menge `M` impliziert:

- Mit `(car M)` erhält man ein Element der Menge `M` und `(cdr M)` liefert dementsprechend die Menge `M` ohne dieses Element.
- Die Vereinigung zweier disjunkter Mengen kann mit `append` durchgeführt werden.
- Das Hinzufügen eines Elementes `x` zu einer Menge `M`, die dieses Element noch nicht enthält, kann mit `cons` erfolgen.

Ist M eine Menge, so wird die Potenzmenge von M („Menge aller Teilmengen von M“) mit $\mathcal{P}(M)$ bezeichnet. Im Falle $M = \emptyset$ ist $\mathcal{P}(M) = \{\emptyset\}$. Anderenfalls gilt für jedes $x \in M$:

$$\mathcal{P}(M) = \mathcal{P}(M \setminus \{x\}) \cup \{(\{x\} \cup A) : A \in \mathcal{P}(M \setminus \{x\})\}$$

Sie können also die Potenzmenge einer nichtleeren Menge M dadurch erzeugen, dass Sie ein beliebiges Element x aus der Menge M entfernen und aus der Restmenge die Potenzmenge M' bilden. Danach vereinigen Sie M' mit einer Kopie von M', bei der jede Teilmenge um x erweitert wird.
Benutzen Sie diese Formel um eine Scheme Prozedur `potenzmenge` zu erstellen, die die Potenzmenge einer als Liste dargestellten Menge bestimmt.
Hinweise zur Implementierung von `potenzmenge`:

- Sie werden mit einem rekursiven Aufruf von `potenzmenge` zunächst die Menge $\mathcal{P}(M \setminus \{x\})$ bestimmen müssen. Das Ergebnis merken Sie sich am besten in einer lokalen Variablen (Stichwort: `let`) da Sie es an zwei Stellen benötigen. (Ein zweiter rekursiver Aufruf würde das Laufzeitverhalten drastisch verschlechtern!)
- Um aus der Menge $\mathcal{P}(M \setminus \{x\})$ die Menge $\{(\{x\} \cup A) : A \in \mathcal{P}(M\ \{x\})\}$ zu erzeugen benutzen Sie am besten einen `map`-Ausdruck. (Ansonsten wird es wohl nötig sein eine zusätzliche Hilfsfunktion zu schreiben.) Die als Ergebnis gelieferte Liste enthält offenbar keine Mehrfachvorkommen und ist bereits die gewünschte Menge.

Bei den folgenden Anwendungsbeispielen beachten Sie bitte, dass die von Ihnen geschriebene Scheme-Funktion höchstwahrscheinlich die Listenelemente in einer anderen Reihenfolge liefert. Für die hier benutzten Darstellung von Mengen ist diese Reihenfolge jedoch unerheblich:

```
(potenzmenge '())       => (())
(potenzmenge '(2))      => (() (2))
(potenzmenge '(2 3))    => (() (2) (3) (2 3))
(potenzmenge '(2 3 5))  => (() (2) (3) (5) (2 3) (2 5) (3 5) (2 3 5))
```

Ist M eine Menge und $n \geq 0$, so definiert man $\mathcal{P}_n(M) := \{A \in \mathcal{P}(M) : |A| = n\}$, d.h. $\mathcal{P}_n(M)$ ist die Menge aller n-elementigen Teilmengen von M. Im Falle $n = 0$ ist also $\mathcal{P}_n(M) = \{\emptyset\}$. Ist hingegen $n \neq 0$ und $M = \emptyset$ so ist $\mathcal{P}_n(M) = \emptyset$. Anderenfalls gilt für jedes $x \in M$:

$$\mathcal{P}_n(M) = \mathcal{P}_n(M \setminus \{x\}) \cup \{(\{x\} \cup A) : A \in P_{n-1}(M \setminus \{x\})\}$$

Schreiben Sie in Analogie zu `potenzmenge` eine Scheme-Funktion `potenzmenge-n`.

Auch bei den folgenden Anwendungsbeispielen beachten Sie bitte, dass die von Ihnen geschriebene Scheme-Funktion höchstwahrscheinlich die Listenelemente in einer anderen Reihenfolge liefert.

```
(potenzmenge-n '(2 3 5) 0)  => (())
(potenzmenge-n '(2 3 5) 1)  => ((2) (3) (5))
(potenzmenge-n '(2 3 5) 2)  => ((2 3) (2 5) (3 5))
(potenzmenge-n '(2 3 5) 3)  => ((2 3 5))
(potenzmenge-n '(2 3 5) 4)  => ()
```

- **Die Bearbeitungszeit beträgt 45 Minuten.**
- **Bei Fragen wenden Sie sich bitte an einen Betreuer.**
- **Viel Erfolg!**