Regeln

Einführung in die Programmierung (I167)

1. **Funktionskopf**
   Verwende aussagekräftige Namen für Funktionen und Variablen („sprechende Bezeichner“).

2. **Zweckbestimmung**
   Beschreibe den Zweck der Funktion durch einen Satz, der das Resultat der Funktion in Abhängigkeit von ihren Argumenten beschreibt.

3. **Tests**
   Schreibe Tests für Funktionen unter Verwendung einer der check-...-Prozeduren von Racket auf.
   Es ist üblich, die Tests zwischen die Zweckbestimmung und die Funktion zu schreiben.

4. **Hilfsfunktionen**
   Definiere für jeden Zusammenhang zwischen Größen, die sich aus der Problembeschreibung ergeben, eine Funktion.

5. **Konstantendefinition**
   Ersetze jede Konstante, deren Bedeutung sich nicht aus dem Kontext ergibt, durch einen sprechenden Variablennamen.
   Ziel: Verbesserung der Lesbarkeit

6. **Entwurf bedingter Funktionen**
   Für den Funktionsrumpf ist

   - ein `cond`-Skelett mit je einer Frage-Antwort-Kombination für jeden Fall zu formulieren,
   - für jeden Fall die Frage (Bedingung) zu formulieren,
   - für jeden Fall der Racket-Ausdruck für die Berechnung der Antwort zu ermitteln.

7. **Definition von Datenstrukturen**

   (a) Datenanalyse: Identifikation der Komponenten, aus denen die Datenstruktur besteht

   (b) Datendefinition

   ```
   ;; Ein s ist ein Wert
   ;;    (make-s s1 ... sn)
   ;; wobei s1 ...
   ```

   (c) Definition der Datenstruktur

   ```
   (define-struct s [s1 ... sn])
   ```

8. **Strukturverarbeitende Funktionen**

   (a) Bestimme die für die Berechnung des Ergebnisses erforderlichen Komponenten der Datenstruktur!

   (b) Schreibe den Selektor jeder Komponente der Datenstruktur in die Funktionsschablone, von deren Wert das Ergebnis der Funktion abhängt.

   (c) Nimm den Aufruf des Konstruktors in die Schablone mit auf, falls die Funktion eine Datenstruktur als Resultat liefern muss.

9. **Funktion zur Verarbeitung gemischter Daten**

   (a) Ergibt die Datenanalyse, dass gemischte Daten zu verarbeiten sind, schreiben Sie zunächst eine Datendefinition der folgenden Form auf:

   ```
   ;; Ein s ist entweder
   ;; - ein t1 oder
   ;; - ...
   ;; - ein tn
   ;; Name: x
   ```

   (b) Für eine s-verarbeitende Funktion ist mindestens je ein Testfall für jede Variante von s aufzuschreiben.

   (c) Seien t1?, t2?, ..., tn? Die Prädikate zu den in s vorkommenden Varianten t1, t2, ..., tn. Dann ergibt sich für die Funktion die Funktionsschablone:

   ```
   (check-... )
   ...
   (check-... )
   (define f
      (lambda [x]
         (cond
            [(t1? x) ...]
            [(t2? x) ...]
            ...
            [(tn? x) ...])))
   ```

10. **Verträge für Funktionen**

    - Formuliere einen Vertrag für jede Funktion! (Schreibe ihn als Kommentar vor dem Funktionskopf!)
    - Der Vertrag beschreibt:
      - Arten von Daten (Datentypen) der Argumente
      - Art von Daten des Resultats

    ```
    <Name der Funktion> : <Typ von Argument 1> ... <Typ von Argument n>
                          -> <Typ des Resultats>
    ```

11. **Definition rekursiver Datenstrukturen**
    Schreibe eine präzise Datendefinition für die zu verarbeitende rekursive Datenstruktur auf. Folge dabei dem im Abschnitt Datendefinition angegebenen Muster.

12. **Anzahl der Testfälle fur rekursive Datenstrukturen**
    Formuliere mindestens einen Test für den Fall, dass die rekursive Datenstruktur

    - keine Elemente,
    - genau ein Element,
    - mehr als ein Element

    enthält!

13. **Funktionsschablone fur die Verarbeitung rekursiver Datenstrukturen**
    Entwickle eine Funktionsschablone, die der rekursiven Struktur der Daten folgt:

    - Schreibe das Skelett eines cond-Ausdrucks auf mit je einer Frage-Antwort-Klausel für jeden nicht-rekursiven und jeden rekursiven Fall!
    - Notiere dabei für die rekursiven Fälle die passenden Selektionsausdrücke ((first ...), (rest ...))!
    - Dabei ist zu beachten, dass auf den Ausdruck (rest ...) die zu entwickelnde Funktion rekursiv anzuwenden ist.