Container füllen

Die Schritte der Entwicklung im Kurs

mit zusätzlichen Erläuterungen

Container füllen

- Modellierungsvorgaben
  - *stuecke* ist eine Liste von Zahlen, welche die Größe der einzelnen Stücke angeben
  - vorgegeben ist '(30 30 30 30 20 20 20 20)
  - *container* ist eine Liste, die als erstes Element (*am Kopf*) die Zahl für seine Größe enthält
  - als Beispiel wird eine Größe von 80 gewählt
  - der Container soll vollständig (*alternativ wäre: optimal*) gefüllt werden

Container füllen

- Schritt 1:
  - Funktionskopf definieren
  - zur Kennzeichnung der Unvollständigkeit als Funktionsrumpf die Rückgabe *'undefiniert* einfügen

```
(define (fuelle stuecke container)
  'undefiniert)
```

Container füllen

- Schritt 1
  - Testen

```
(fuelle
  '(30 30 30 30 20 20 20 20)
  '(80)
)

   →   'undefiniert)
```

Container füllen

- Schritt 2
  Es gibt verschiedene Möglichkeiten, auf die das Programm reagieren können muss, daher muss eine Verzweigung eingebaut werden
  - *cond* stellt eine Mehrfachverzweigung zur Verfügung
  - alternativ kann man bei nur zwei Alternativen *if* einsetzen, was hier aber nicht sinnvoll ist, da wir tatsächlich auf mehrere Alternativen reagieren müssen

Container füllen

- Schritt 2
  - cond einbauen
  - Hinter cond kommen prinzipiell immer Paare aus einer Bedingung und dem Wert, den die Funktion zurückgeben soll, wenn die Bedingung zutrifft
  - else kennzeichnet den sonst-Fall

Container füllen

- Schritt 2: Container exakt voll?
  - Im Erfolgsfall wird der Container zurück gegeben
  - wir gliedern die Prüfung in eine Funktion exakt-voll? aus

```
(define (fuelle stuecke container)
  (cond
    ((exakt-voll? container)
     container)
   (else  'undefiniert)
))
```

Container füllen

- Schritt 1 von exakt-voll? : (Funktionskopf)
  - Die Funktion benötigt allein den Container
  - Der Name kennzeichnet sie als Prädikat

```
(define (exakt-voll? container)
  'undefiniert
  )
```

Container füllen

- Schritt 2 von exakt-voll? :
  - Wir verwenden eine weitere Hilfsfunktion fuellung
  - Die Auswertung enthält eine Schachtelung: erst fuellung und first anwenden, danach deren Werte vergleichen

```
(define (exakt-voll? container)
  (= (fuellung container) (first container))
  )
```

Container füllen

- Schritt 1 und 2 von fuellung :
  - Die Funktion benötigt allein den Container
  - und verwendet wiederum eine Hilfsfunktion zur Berechnung des Inhalts

```
(define (fuellung container)
  (summe (rest container))
  )
```

Container füllen

- Schritt 1 von summe :
  - Die Funktion benötigt allein den Inhalt des Containers, der im Rest der Containerliste steht
  - Parameternamen
    - haben im Rumpf Gültigkeit
    - werden durch den Aufruf mit Werten besetzt

```
(define (summe inhalt)
  'undefiniert
  )
```

Container füllen

- Schritt 2 von summe : (Elementarfall)
  - Wenn (→ Verzweigung mit cond) die Liste leer ist, kennt man den Wert der Summe, er ist 0

```
(define (summe inhalt)
  (cond
    ((null? inhalt)
     0)
    (else
     'undefiniert)
  )
```

Container füllen

- Schritt 3 von summe :
  - Sonst (→ else) addiert man die erste Zahl der Liste zu der Summe für die Restliste

```
(define (summe inhalt)
  (cond
    ((null? inhalt)
     0)
    (else
     (+ (first inhalt) (summe (rest inhalt)))))
  )
```

Container füllen

- Schritt 2 (von fuelle)
  - Testen geht erst jetzt, da nun die Hilfsfunktionen zur Verfügung stehen

```
(fuelle  '(<beliebig>)  '(80 30 30 20) )
(fuelle  '(30 30 30 30 20 20 20 20)  '(80) )
  →  '(80 30 30 20)
  →  'undefiniert)
```

Container füllen

- Schritt 3: Container würde zu voll werden
  - Es wird ohne das Stueck einzufügen weiter gearbeitet

```
(define (fuelle stuecke container)
  (cond
     ...
     ((zu-voll? (first stuecke) container)
      (fuelle (rest stuecke) container))
    (else  'undefiniert)
))
```

Container füllen

- Schritt 1 von zu-voll? : (Funktionskopf)
  - Die Funktion benötigt das aktuelle Stueck und den Container
  - Der Name kennzeichnet sie durch das Fragezeichen als Prädikat

```
(define (zu-voll? stueck container)
  'undefiniert
  )
```

Container füllen

- Schritt 2 von zu-voll? :
  - Die Summe aus dem aktuellen Stueck und der Füllung des Containers wird mit der Größe des Containers verglichen

```
(define (zu-voll? stueck container)
  (>  (+ stueck (fuellung container))  (first container))
  )
```

Container füllen

- Schritt 3: zu-voll? testen
  - Der Test führt zu einem Fehler. Warum?
  - Das Testen des Auswertungsablaufs mit dem Debugger zeigt, dass die Funktion sich selbst ohne das Stueck mit der Groesse 30 aufruft, die Stueckeliste dabei also leer ist und first fehlschlägt.

```
(fuelle '(30) '(80 30 30))
  → Fehlermeldung
```

Container füllen

- Schritt 4
  - Abfrage ins cond einbauen
  - nach dem Erfolgsfall

```
(define (fuelle stuecke container)
  (cond
    ((exakt-voll? ...)
    ((null? stuecke)
     #f)
    ((zu-voll? ...)
    (else
     'undefiniert)
    ))
```

Container füllen

- `null?` hat ein Fragezeichen am Ende, durch das man kennzeichnet, dass es eine Funktion ist, die nur die beiden Werte `#t` oder `#f` zurückliefern kann
  - Solche Funktionen bezeichnet man als Prädikate (s.o.)

Container füllen

- Schritt 5: else – Fall
  - Es gibt keine Sonderfälle mehr, im Normalfall wird das Stueck eingefüllt

```
(define (fuelle stuecke container)
  (cond
    ...
    (else
     (fuelle
      (rest stuecke)
      (fuelle-ein (first stuecke) container)))
))
```

Container füllen

- Schritt 1 von fuelle-ein : (Funktionskopf)
  - Wieder eine Hilfsfunktion
  - Die Funktion benötigt das aktuelle Stueck und den Container

```
(define (fuelle-ein stueck container)
  'undefiniert
  )
```

Container füllen

- Schritt 2 von fuelle-ein :
  - Die Groesse muss wieder an den Kopf der Containerliste gesetzt werden, das geht mit cons
  - Das neue Stueck kommt an den Kopf des Inhalts

```
(define (fuelle-ein stueck container)
  (cons
      (first container)
      (cons stueck (rest container))))
  )
```

Container füllen

- Schritt 5: alle Fälle testen

```
(fuelle '(30) '(80 30 30))
(fuelle *stuecke* '(80 30 30 20))
(fuelle '() *container*)
(fuelle *stuecke* *container*)

→ #f
→ (80 30 30 20)
→ #f
→ (80 20 30 30)
```