

Praktische Informatik 1: Deklarative Programmierung

Prüfungsprotokoll

Dozent: Prof. Dr. Ostermann

Hauptklausur WS 2021/22

Anmerkung: Die Klausur umfasst zwei Programmieraufgaben und 40 Ankreuzaufgaben. Jede Ankreuzaufgabe hat vier Antwortmöglichkeiten, von denen genau eine richtig ist. Wir konnten uns jedoch nicht an alle Fragen und alle Antworten erinnern. Die richtige Antwort sollte aber immer dabei sein.

Bearbeitungszeit: 90 Minuten

Jede der 40 Ankreuzaufgaben hat genau eine richtige Antwort. Für jede Frage können Sie keine Antwort, eine Antwort, oder mehrere Antworten geben indem sie die entsprechenden Felder ankreuzen. Das bedeutet:

- Wenn Sie die Antwort sicher wissen, kreuzen sie nur diese an.
- Wenn Sie sich nicht sicher sind aber einige Antworten ausschließen können, kreuzen sie die Antworten an, die korrekt sein könnten.
- Wenn Sie überhaupt nicht wissen, was die Antwort sein könnte und nichts ausschließen können, kreuzen Sie nichts an.

Die Antworten werden so bewertet, dass zufälliges Raten null Punkte ergibt, also eine falsche Antwort Minuspunkte gibt.

Programmieraufgabe 1

Eine Firma möchte Teile (parts) produzieren. Ein Teil hat einen Namen (name) und Baukosten (cost), und es kann aus Bestandteilen (components) bestehen. Die Bestandteile sind dabei wiederum Teile und sie können auch selbst wieder aus beliebig vielen Bestandteilen bestehen. Die Gesamtkosten eines Teils ist die Summe aus den Baukosten des Teils und den Baukosten der Bestandteile. Sie können bei der Aufgabe davon ausgehen, dass der Datentyp `List-of X` bereits definiert wurde.

1. Implementieren sie eine Datendefinition für Teile. Definieren sie ein Beispiel für ein Teil mit zwei Bestandteilen als Konstante.

Hinweis: Implementieren sie auch weitere Datendefinitionen, falls sie diese benötigen.

2. Programmieren sie eine Funktion `total-cost`, die die Gesamtkosten eines Teils berechnet. Geben sie die Signatur und eine Funktionsbeschreibung an. Sie müssen keine Tests entwerfen.

Hinweis: Sie können diese Aufgabe ohne Higher-Order-Funktionen lösen. Falls sie Higher-Order-Funktionen verwenden wollen, können sie diese Funktionen, die sie aus der Vorlesung kennen, verwenden:

```
; [X Y] (X Y -> Y) Y (list-of X) -> Y
(define (foldr op z xs) ...)
```

```
; [X Y] (X -> Y) (list-of X) -> (list-of Y)
(define (map f xs) ...)
```

Programmieraufgabe 2

In dieser Aufgabe sollen sie zuerst eine Funktion `contains-smaller-zero?` programmieren, die bestimmt, ob in einer Liste von Zahlen eine Zahl ist, die kleiner als Null ist. Anschließend werden sie eine allgemeinere Higher-Order-Funktion `contains?` programmieren, die bestimmt, ob ein bestimmtes Element in der Liste ist. Sie werden dann die `contains?`-Funktion verwenden, um die Funktion `contains-smaller-zero?` und eine weitere Funktion erneut zu entwerfen.

1. Entwerfen sie eine Funktion `smaller-than-zero?`, die bestimmt, ob in einer Liste von Zahlen eine Zahl enthalten ist, die kleiner als null ist. Sie müssen keine Tests schreiben.

```
; [List-of Number] -> Boolean  
; Determines whether there is a Number smaller than zero in the list
```

2. Entwerfen sie nun eine Higher-Order-Funktion `contains?`, die überprüft, ob ein bestimmtes Element in der übergebenen Liste ist. Die Funktion soll ein Prädikat und eine Liste als Argumente übernehmen. Ein Prädikat ist eine Funktion `X -> Boolean`, die überprüft, ob eine bestimmte Bedingung auf ein Element zutrifft. Schreiben sie auch eine Signatur. Sie müssen keine Funktionsbeschreibung und keine Tests schreiben. Orientieren sie sich beim Funktionsentwurf an ihr Vorgehen bei der vorhergehenden Aufgabe.

3. Schreiben sie erneut eine Funktion `contains-smaller-zero?` sowie eine Funktion `contains-string?`. Verwenden sie keine rekursiven Funktionsaufrufe. Verwenden sie stattdessen die von ihnen entworfene Funktion `contains?`. Sie müssen keine Funktionsbeschreibung, keine Signatur und keine Tests schreiben.

Ankreuzaufgaben

1. Betrachten sie folgendes Programm. Was ist die Bedeutung des Programms?

```
(local [(define X 5)]  
  (local [(define X 8)]))  
(+ X 2)
```

- ☐ 7
- ☐ 10
- ☐ 15
- ☐ Es tritt ein Fehler auf.

2. Was besagt das DRY-Prinzip?

- ☐ Rekursive Funktionsaufrufe sollten vermieden werden.
- ☐ Codewiederholungen und redundante Implementierungen müssen vermieden werden.
- ☐ Dass man von der Copy-Paste-Funktion regen Gebrauch machen sollte.
- ☐ Wenn im Programmcode der gleiche Textausschnitt mehrfach vorkommt, kann das Programm langsam werden.

3. Was ist syntaktischer Zucker?

- ☐ Ausdrücke, die nur dazu da sind, um bereits bestehende Ausdrücke anders auszudrücken. Dazu zählt z.B. `cond` und `if`.
- ☐ Zusatzfunktionen von Entwicklungsumgebungen, die zum Programmieren nicht unbedingt gebraucht werden (z.B. automatische Code-Einfärbung).
- ☐
- ☐

4. Was kann mittels der KONG-Regel geschehen?

- ☐ Falls `(f e-1 ... e-n)` in der Umgebung definiert ist, kann `(f v-1 ... v-n)` durch den Funktionsbody von `f` ersetzt werden, indem alle vorkommen von `e-n` durch `v-n` ersetzt werden.
- ☐ Falls `(define X 7)` Teil der Umgebung ist, so kann `(+ X 35)` durch `(+ 7 35)` ersetzt werden.
- ☐ `(+ 7 35)` kann durch 42 ersetzt werden.
- ☐ Falls `e-1 -> e-2` gilt, so kann `(cons (e-1) empty)` durch `(cons (e-2) empty)` ersetzt werden.

5. Was sind magic numbers?

- ☐ Zahlenwerte, die an Konstanten gebunden sind.
- ☐ Zahlen im Programmcode, deren Bedeutung nicht unmittelbar ersichtlich ist.
- ☐
- ☐

6. Sie möchten eine Charakterklasse für ein Videospiel in BSL modellieren. Eine Charakterklasse kann entweder ein Magier, ein Krieger oder ein Heiler sein. Welche Art von Datentyp verwenden sie hierfür?

☐ Summentyp
☐ Produkttyp
☐ Rekursiver Datentyp
☐ Einen bereits in BSL eingebauten Datentyp.

7. Sie möchten einen Charakter für ein Videospiel in BSL modellieren. Ein Charakter besitzt einen Namen, ein Level, einen Lebensstand und die Attribute Strength, Perception, Endurance, Charisma, Intelligence, Agility und Luck. Welche Art von Datentyp verwenden sie hierfür?

☐ Summentyp
☐ Produkttyp
☐ Rekursiver Datentyp
☐ Einen bereits in BSL eingebauten Datentyp.

8. Betrachten sie die `compose`-Funktion, die sie bereits aus der Vorlesung kennen:

```
(define (compose f g) (lambda (x) (f (g x))))
```

Welche ist die korrekte Signatur für `compose`?

☐ $[X\ Y\ Z]\ (Y \rightarrow Z)\ (X \rightarrow Y) \rightarrow (X \rightarrow Z)$
☐ $[X\ Y\ Z]\ (X \rightarrow Z)\ (X \rightarrow Y) \rightarrow (X \rightarrow Y)$
☐ $[X\ Y\ Z]\ (Y \rightarrow Z)\ (X \rightarrow Y) \rightarrow (Y \rightarrow Z)$
☐ $[X\ Y\ Z]\ (Y \rightarrow Z)\ (Z \rightarrow Y) \rightarrow (Z \rightarrow Z)$

9. Betrachten sie die `unfold`-Funktion, die sie bereits aus der Vorlesung kennen:

```
; [X Y] (Y -> Bool) (Y -> Y) (Y -> X) Y -> (list-of X)  
(define (unfold stop next emit seed) ...)
```

Was macht die dritte Funktion?

☐ Die Funktion wird am Ende einmal angewendet, um das letzte Listenelement zu bestimmen.
☐ Die Funktion verwandelt das momentane Element in das nächste, das bearbeitet werden soll.
☐ Die Funktion gibt an, wann `unfold` beendet werden soll.
☐ Die Funktion wird auf die Elemente der Ausgabeliste angewendet, um die Elemente der Ausgabeliste zu erzeugen

10. Betrachten sie die `flatmap`-Funktion, die sie bereits aus der Vorlesung kennen:

```
(define (flatmap f xs)  
  (foldr  
    append  
    empty  
    (map f xs)))
```

Welche Signatur hat `flatmap`?

- ☐ `[X Y] (X -> (list-of Y)) (list-of X) -> (list-of Y)`
- ☐ `[X Y Z] (X -> (Y -> Z)) (list-of X) -> (list-of Z)`
- ☐ `[X] (X -> (list-of X)) (list-of X) -> (list-of X)`
- ☐ `[X Y] (X -> list-of Y) X -> (list-of Y)`

11. Was ist ein gültiger Wert in BSL?

- ☐ `(define-struct my-posn (x y))`
- ☐ `#false`
- ☐ `(f 2)`
- ☐ `(+ 21 21)`

12. Wozu dienen Tests?

- ☐ Durch ausreichend Tests kann sichergestellt werden, dass das Programm terminiert.
- ☐ Die Tests sorgen dafür, dass die Funktion im Programm nur mit gültigen Argumenten aufgerufen wird.
- ☐ Durch Tests kann man sich sicher sein, dass die Funktion richtig funktioniert.
- ☐ Sie formalisieren die Spezifikation einer Funktion bzw. verdeutlichen diese.

13. Wozu wird folgendes Programm im nächsten Schritt ausgewertet?

`(cond [#true (+ 2 2)] [else (+ 3 2)])`

- ☐ 4
- ☐ `(+ 3 2)`
- ☐ `(+ 2 2)`
- ☐ `(#true (+ 2 2))`

14. Was ist die Bedeutung des folgenden Programms?

`(quasiquote (+ 3 (unquote (+ 5 7)) 8))`

- ☐ `(list '+ 3 12 8)`
- ☐ `(list '+ 3 (list '+ 5 7) 8)`
- ☐ 23
- ☐ `(quasiquote (+ 3 (unquote (+ 5 7)) 8))`

15. Welche der folgenden Aussagen über Funktionen, die generative Rekursion verwenden, ist korrekt?

- ☐ Ihr Entwurf fordert viel Kreativität.
- ☐ Man muss sich keine Gedanken über deren Terminierung machen, da Funktionen mit generativer Rekursion immer terminieren.
- ☐ Sie werden immer genau zweimal aufgerufen: Einmal am Anfang eines Algorithmus und einmal am Ende, um das Ergebnis zu erzeugen.
- ☐

16. Betrachten sie folgende kontextfreie Grammatik:

```
<TLD> ::= de | com | co.uk  
<PREFIX> ::= alice | bob | mallory | <PREFIX>-<PREFIX>  
<NAME> ::= amazon | facebook | google | webshop | <NAME>.<NAME>  
<URL> ::= <PREFIX>.<NAME>.<TLD>
```

Welcher Ausdruck ist **keine** gültige URL?

- ☐ alice-bob.amazon-webshop.co.uk
- ☐ mallory.google.de
- ☐ alice-alice-alice.google.google.com
- ☐ bob.facebook.de

Für die nächsten Aufgaben gelte folgendes Programm:

```
(define-struct collision (top bottom left right))  
; A Collision is a struct: (make-collision bool bool bool bool)  
; interp. the sides on which an object is colliding
```

17. Was ist die erste Zeile des Programms? (define-struct ...)

- ☐ Ein Summentyp
- ☐ Ein Produkttyp
- ☐ Ein Boolean
- ☐ Eine Strukturdefinition

18. Was ist eine gültige collision?

- ☐ "collision"
- ☐ "(make-collision #true #true #true #false)"
- ☐ "(make-collision #true #true #false #false #false)"
- ☐ "#false"

19. Wieviele Werte des Datenuniversums werden durch collision beschrieben?

- ☐ 2
- ☐ 4
- ☐ 16
- ☐ Keine der Antworten ist richtig.

Für die nächsten Aufgaben gelte folgendes Programm:

```
(define (f xs)  
  (cond [(empty? xs) empty]  
        [else (cons (first xs) (g (rest xs)))]))
```

```
(define (g xs)  
  (cond [(empty? xs) empty]  
        [(empty? (rest xs)) empty]  
        [else (f (rest (rest xs)))]))
```

20. Welche Signatur hat die Funktion f?

- ☐ [X] X -> X
- ☐ [X Y] (list-of X) -> (List-of Y)
- ☐ [X] (List-of X) -> (List-of X)
- ☐ (list-of Number) -> Number

21. Was ist die Bedeutung des folgenden Ausdrucks?

```
(f (list #true 42 70 55 "nein"))
```

- ☐ empty
- ☐ (list #true 55)
- ☐ (list #true 42 70 55 "nein")
- ☐ (list 42 55 "nein")

Für die nächsten Aufgaben gelte folgendes Programm:

```
(define (myfun x)
  (cond [(> x 10) (- x 10)]
        [else #false]))
```

22. Was ist die Bedeutung des folgenden Ausdrucks? (myfun (myfun 2))

- ☐ Error: >: expects a real, given #false
- ☐ -18
- ☐ 20
- ☐ #false

23. Was ist die Signatur von myfun?

- ☐ Number -> Number
- ☐ Number -> MaybeNumber
- ☐ MaybeNumber -> Boolean
- ☐ MaybeNumber -> MaybeNumber

24. Was ergibt die Auswertung des folgenden Programms?

```
(or #true (+ 2 2))
```

- ☐ #true
- ☐ Error: or: question result is not true or false: 4
- ☐ #false
- ☐ 4

25. Betrachten sie das folgende Programm: (and e-1 e-2) Wann wird e-2 ausgewertet?

- ☐ Nur wenn e-1 #true ist.
- ☐ Nur wenn e-1 #false ist.
- ☐ Niemals.

- ☐ Bevor e-1 ausgewertet wird.
26. Wozu kann (or (and (not b2) #true) b1) reduziert werden?
- ☐ (or (not b2) b1)
 - ☐ #true
 - ☐ (or and (not b2 #true))
 - ☐ (and (not b2) b1)
27. Was ergibt die Auswertung von (+ 1 "1")?
- ☐ 2
 - ☐ "11"
 - ☐ "2"
 - ☐ Error: +: expects a number, given "1"
28. Was ergibt die Auswertung von (#true + 2)?
- ☐ Error: function call: expected a function after the open parenthesis, but found something else
 - ☐ #true
 - ☐ (list #true #true)
 - ☐ 2
29. Welche dieser Funktionen ist eine Funktion erster Ordnung?
- ☐ (lambda (x y) (x y))
 - ☐ (lambda (x) ((lambda (y) y) x))
 - ☐ (lambda (x) (lambda (y) (+ x y)))
 - ☐ (lambda (x) (lambda (y) (lambda (z) (+ x y z))))

Für die nächsten Aufgaben gelte folgendes Programm:

```
(define (my-animate f)
  (big-bang 500
    (to-draw f)
    (on-tick ...)))
```

30. Welche Funktion passt in die Lücke?
- ☐ (lambda (x) (overlay (circle x "solid" "blue")))
 - ☐ (lambda (x) (make-state (+ x 1)))
 - ☐ (lambda (x) (make-state (state-num x)))
 - ☐ (lambda (x) (+ x 1))
31. Welche Signatur passt zu my-animate?
- ☐ (Number -> Image) -> Number

- ☐ Number -> Image
- ☐ Image -> Number
- ☐ Number -> String

32. Warum ist das folgende Programm falsch?

```
; Number -> Number
; Bestimmt den Vorgänger einer Positiven Zahl
(check-expect (pre 5) 4)
(check-expect (pre 43) 42)
(check-expect (pre 0) 0)
(check-expect (pre -1) #false)
(define (pre n)
  (cond
    [(> 0 n) (- n 1)]
    [(= 0 n) 0]
    [(< 0 n) #false]))
```

- ☐ Gemäß des Entwurfsrezept müssen alle `cond`-Ausdrücke beim Refactoring durch verschachtelte `if`-Ausdrücke ersetzt werden.
- ☐ Weil die angegebene Signatur nicht stimmt.
- ☐
- ☐

33. Was ergibt die Auswertung des folgenden Programms?

```
(define X (+ 2 Y))
(define Y 2)
(+ 2 X)
```

- ☐ 2
- ☐ 4
- ☐ Error: Y: this variable is not defined
- ☐ 6

34. Was ist eine gültige Umgebung?

- ☐ (define X 3)
(f 2)
- ☐ (define X (+ 2 2))
- ☐ (define X 5)
- ☐

35. Was ist ein gültiger Auswertungskontext?

- ☐ (cond [[] e-1] [else e-2])
- ☐ (cond [#true e-1] [[] e-3])

- ☐ (cond [e-1 e-2] [[] e-4])
- ☐ (cond [[] []] [[] e-2])

36. Welcher Ausdruck wird durch folgendes Programm zur Umgebung hinzugefügt?
 (define X (lambda (y) (+ 2 2)))

- ☐ (+ 2 2)
- ☐ (define X (lambda (y) 4))
- ☐ (define X (lambda (y) (+ 2 2)))
- ☐ (define X 4)

37. ☐
☐
☐
☐

38. ☐
☐
☐
☐

39. ☐
☐
☐
☐

40. ☐
☐
☐
☐