EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Beispielklausur
zur Vorlesung **Informatik 1**
im Wintersemester 2023/24

Matrikelnummer:

Name:

i001

Diese Beispielklausur besteht aus einer Programmieraufgabe und 9 Fragen. Für die Bearbeitung einer Klausur diesen Umfangs hätten Sie ca. 33 Minuten Zeit.

Jede der 9 Fragen 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.

Wir benutzen die Konvention dass [x] "Antwort ausgewählt" bedeutet, und sowohl [ ] als auch [■] bedeuten "Antwort *nicht* gewählt".

Programmieraufgabe I

Gegeben sind folgende Struktur- und Datendefinitionen:

```
(define-struct branch (left right))
; interp. a node in the middle of a binary tree.

(define Tree (signature (mixed (BranchOf Tree Tree)
                               Number)))
; interp. a binary tree of numbers.
```

Aufgabe: Implementieren Sie eine Funktion, die einen `Tree`-Wert konsumiert und die Summe aller im `Tree` enthaltenen Zahlen produziert.

Verwenden Sie das Entwurfsrezept zur Funktionsdefinition!

Bewertungsschema

Diese Aufgabe zählt so viel wie 10 Fragen:

- ☐ *Signatur angegeben*
- ☐ *Signatur syntaktisch korrekt und passend zur Aufgabenstellung*
- ☐ *Aufgabenbeschreibung angegeben*
- ☐ *Aufgabenbeschreibung syntaktisch korrekt und passend zur Aufgabenstellung*
- ☐ *Funktionskopf syntaktisch korrekt und passend zur Signatur*
- ☐ *Testfälle angegeben und syntaktisch korrekt*
- ☐ *Testfälle inhaltlich sinnvoll*
- ☐ *Implementierung angegeben*
- ☐ *Implementierung benutzt sinnvolles Template*
- ☐ *Implementierung korrekt*

Tragen Sie hier nichts ein. Wir füllen das Bewertungsschema während der Korrektur aus.

Programm A

Gegeben sei folgendes Programm:

```
(define (f x)
  (cond
    [(<= 3 x 5)
     (+ x 10)]
    [else
     (/ 7 x)]))
```

Die folgenden zwei Fragen beziehen sich auf dieses Programm.

Frage 1

Was ergibt die Auswertung des Ausdrucks `(f 7)` bezüglich des oben angegebenen Programmes A?

Das Ergebnis ist . . .

a☐ . . . der Fehler "division by zero"
b☐ . . . der Wert `1`
c☐ . . . der Wert `17`
d☐ . . . der Wert `10`

Frage 2

Was ergibt die Auswertung des Ausdrucks `(f 0)` bezüglich des oben angegebenen Programmes A?

Das Ergebnis ist . . .

a☐ . . . der Wert `1`
b☐ . . . der Fehler "division by zero"
c☐ . . . der Wert `10`
d☐ . . . der Wert `17`

Frage 3

Was ist die Signatur der folgenden Funktion?

```
(define (f x y)
  (+ x (string->number y)))
```

Die Signatur ist . . .

a☐ `Number String -> Number`
b☐ `String Number -> Number`
c☐ `Number String -> String`
d☐ `String String -> Number`

Frage 4

Was ist eine Reduktionssemantik?

Eine Reduktionssemantik beschreibt, wie man . . .

- *a* ☐ . . . ein Programm schrittweise in einem Wert umwandeln kann.
- *b* ☐ . . . eine Grammatik schrittweise in eine Datendefinition umwandeln kann.
- *c* ☐ . . . eine Datendefinition schrittweise in einen Produkt- oder Summentyp umwandeln kann.
- *d* ☐ . . . einen Wert schrittweise in eine S-Expression umwandeln kann.

Frage 5

Wieso müssen die Alternativen in Summentypen immer unterscheidbar sein?

- *a* ☐ Damit man die Teilmenge des Datenuniversums bestimmen kann, die von dem Summentyp definiert wird.
- *b* ☐ Damit man Programme schreiben kann, die alle Alternativen des Summentyps erzeugen.
- *c* ☐ Damit man Tests schreiben kann, die alle Alternativen abdecken.
- *d* ☐ Damit man Funktionen schreiben kann, die die Alternativen unterscheiden.

Frage 6

Betrachten Sie dieses Programm:

```
(define world 42)
```

Was ist der Wert des Ausdrucks `` `(('(hello ,world))) ``?

- *a* ☐ `(list (list (list 'quote (list 'hello 'world))))`
- *b* ☐ `(list (list 'quote (list 'hello 42)))`
- *c* ☐ `(list (list (list 'quote (list 'hello 42))))`
- *d* ☐ `(list (list 'quote (list 'hello 'world)))`

Frage 7

Betrachten Sie dieses Programm:

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

Welcher der folgenden Ausdrücke ergibt den Wert `5` ?

a☐ `(((f +) 2) 3)`
b☐ `((f +) 2 3)`
c☐ `(f + 2 3)`
d☐ `(((f +) 2 4) 3)`

Frage 8

Betrachten Sie folgende Funktionsdefinition:

```
(define (zip-with f xs ys)
  (if (or (empty? xs) (empty? ys))
      empty
      (cons (f (first xs) (first ys))
            (zip-with f (rest xs) (rest ys)))))
```

Beispielsweise ergibt `(zip-with + (list 2 7) (list 3 1))` das Ergebnis `(list 5 8)` und `(zip-with cons (list 1 2) (list empty (list 5)))` ergibt `(list (list 1) (list 2 5))`.

Was ist die allgemeinste Signatur für `zip-with`, die mindestens diese beiden Beispiele abdeckt?

a☐ `; [X Y] (X X -> Y) (list-of X) (list-of X) -> (list-of Y)`
b☐ `; [X Y Z] (X Y -> Z) (list-of Z) (list-of Y) -> (list-of X)`
c☐ `; [X Y Z] (X Y -> Z) (list-of X) (list-of Y) -> (list-of Z)`
d☐ `; [X Y Z] (X Y -> Z) (list-of X) (list-of Y) -> (list-of X)`

Frage 9

Betrachten Sie folgende Strukturdefinition:

```
(define-struct foo (bar baz))
```

Welche Funktion ist für diese Struktur definiert?

a☐ `foo-bar?`
b☐ `bar?`
c☐ `bar-foo`
d☐ `foo?`