Nachname: _____

Vorname: _____

Matrikelnummer: _____

Aufgabe	Points	Score
Programmanalyse mit Modulen und I/O	20	
Programmieren mit Listen	28	
Datentypen und Funktionen Höherer Ordnung	22	
Auswertung und Typen	20	
$\sum$	90	

- Sie haben 90 Minuten Bearbeitungszeit.
- Die Klausur besteht aus 4 Aufgaben, in denen Sie 90 Punkte erreichen können.
- Die erreichbaren Punkte sind am Rand notiert.
- Entfernen Sie nicht die Heftklammer.
- Verwenden Sie keinen roten Stift.
- Antworten können in deutscher oder englischer Sprache verfasst werden.

Aufgabe 1: Programmanalyse mit Modulen und I/O 20

Betrachten Sie folgendes Programm.

```
1  module Main(main) where
2
3  include System.Random  -- this module contains randomIO :: IO Integer
4
5  main = do
6    let num = randomIO
7    putStrLn "Try to guess my number"
8    guessingGame (abs num) 1
9
10 guessingGame :: Integer -> Integer -> IO ()
11 guessingGame num n = do
12   str <- getLine
13   let x = (read str :: Int)
14   if x == num then finish n
15   else do
16     let reason = if num < x then "small" else "large"
17     putStrLn $ "Your guess was too " ++ reason ++ ". Try again"
18     guessingGame num (n + 1)
19
20 finish n = putStrLn $ "You guessed my number using " ++ show n ++ " tries"
```

Das Programm enthält vier Fehler. Davon führen drei zu einem Compile-Fehler und einer zu unerwünschtem Verhalten bei der Programmausführung.

- Identifizieren Sie die Fehler durch Angabe der Zeilennummern,
- durch eine kurze Erklärung, warum ein Fehler vorliegt, und
- geben Sie an, wie man den Fehler beheben kann.

Beachten Sie, dass alle vier Fehler unabhängig von einander sind, also jeweils separate Korrekturen benötigen.

(a) Fehler #1 (5)

(b) Fehler #2 (5)

(c) Fehler #3 (5)

(d) Fehler #4 (5)

Aufgabe 2: Programmieren mit Listen 28

Ein Wort v ist ein Suffix von Wort w, wenn v aus mindestens einem Zeichen besteht und es ein Wort u gibt, so dass $w = uv$. Zum Beispiel hat "ball" die Suffixe "ball", "all", "ll" und "l".

Suffixe können für beliebige Listen definiert werden, z. B. hat $[1, 2, 7]$ die Suffixe $[1, 2, 7]$, $[2, 7]$ und $[7]$.

Für die folgenden Programmieraufgaben dürfen Sie beliebige Prelude Funktionen nutzen, außer in Teil (a).

(a) Definieren Sie eine Haskell-Funktion `suff1`, die die Liste aller Suffixe einer Eingabe-Liste in absteigender Länge berechnet. (4)

Beispiel: `suff1 "ball"` sollte zu `["ball","all","ll","l"]` auswerten und der Ausdruck `suff1 [1,2,7]` sollte zu `[[1,2,7],[2,7],[7]]` auswerten.

Für diesen Teil dürfen Sie keine vordefinierten Funktionen auf Listen verwenden, außer die beiden Listen-Konstruktoren.

(b) Geben Sie den allgemeinsten Typ ihrer Implementierung von `suff1` aus Teil (a) an. (2)

(c) Betrachten Sie eine Variante der `foldr`-Funktion: (4)

```
-- zur Erinnerung
foldr :: (a        -> b -> b) -> b -> [a] -> b

-- Variante
fold  :: (a -> [a] -> b -> b) -> b -> [a] -> b
fold f e [] = e
fold f e (x : xs) = f x xs (fold f e xs)
```

Definieren Sie eine alternative Funktion `suff2` zur Berechnung der Liste aller Suffixe einer Eingabe-Liste. Im Gegensatz zu `suff1` muss `suff2` auf `fold` basieren und darf keine Rekursion verwenden.

(d) Im weiteren finden Sie vier Vorschläge `suff3a`–`suff3d` für die Berechnung aller Suffixe einer Liste. Genau eine davon ist richtig, d.h., liefert die Suffixe in absteigender Länge. (13)

1. `suff3a xs = [ take n xs | n <- [ 0 .. length xs] ]`
2. `suff3b xs = [ drop i xs | (i,_) <- zip [0 .. ] xs]`
3. `suff3c xs = map (drop xs) (reverse [0 .. length xs])`
4. `suff3d xs = map (flip drop xs) [0 .. length xs]`

Geben Sie für jede Variante an, ob diese korrekt ist oder nicht. Für die fehlerhaften Varianten geben Sie zudem den Grund an.

Zur Erinnerung:

- `flip :: (a -> b -> c) -> (b -> a -> c)`
- `take, drop :: Int -> [a] -> [a]`

(e) Wozu wertet der Ausdruck `[ x | x : _ <- suff1 xs]` aus, wenn `xs` eine endliche Liste ist? (5)
Gilt dies auch für unendliche Listen?

Aufgabe 3: Datentypen und Funktionen Höherer Ordnung 22

Betrachten Sie folgende Datentypen.

```
data Date = Date Int Int Integer -- Tag, Monat, Jahr
data Employee = Person
  String  -- Name
  Date    -- Einstellungsdatum
  Float   -- Gehalt
```

(a) Definieren Sie eine Funktion `decreaseSalary :: Integer -> Employee -> Employee`, die einen ganzzahlingen Betrag und einen Angestellten als Eingabe erwartet, und das Gehalt dieses Angestellten um den Betrag verringert. (4)

(b) Definieren Sie eine Funktion `salary29Feb :: [Employee] -> Float`, die die Summe aller Gehälter von Angestellten berechnet, die an einem 29. Februar angestellt wurden. (5)

Für diesen Teil sind Rekursion und Importe verboten.

Betrachten Sie nun folgendes Haskell Skript.

```
data Expr = Num Integer | Var String | Sum Expr Expr

eval :: (String -> Integer) -> Expr -> Integer
eval _ (Num n) = n
eval a (Var x) = a x
eval a (Sum e1 e2) = eval a e1 + eval a e2
```

(c) Wir möchten einen Ausdruck `e :: Expr` vereinfachen. Dabei soll jedes Vorkommen von `Sum (Num n1) (Num n2)` durch `e` durch `Num (n1 + n2)` ersetzt werden, bis der resultierende Ausdruck das Muster `Sum (Num n1) (Num n2)` nicht mehr enthält. (6)

Dazu wurde die Funktion `normalize :: Expr -> Expr` implementiert.

```
normalize (Num n) = Num n
normalize (Var x) = Var x
normalize (Sum (Num n1) (Num n2)) = Num (n1 + n2)
normalize (Sum e1 e2) = Sum (normalize e1) (normalize e2)
```

Bearbeiten Sie folgende Aufgaben:

- Erfüllt `normalize` das Ziel? Geben Sie eine ja/nein Antwort.
- Falls die Antwort ja ist, begründen Sie kurz, warum das Resultat von `normalize e` kein Vorkommen von `Sum (Num n1) (Num n2)` enthalten kann.
- Falls die Antwort nein ist, geben Sie einen Beispiel-Ausdruck `e` vom Typ `Expr` an, so dass das Ergebnis von `normalize e` ein Vorkommen von `Sum (Num n1) (Num n2)` enthält.

(d) Wir möchten einige Funktionen mit Hilfe einer fold-Funktion für den Datentyp `Expr` (namens `foldExpr`) definieren. (7)

```
eval a = foldExpr id a (+)
vars = foldExpr (\ _ -> []) ( \ x -> [x] ) (++)   -- Variablen einer Expr als Liste
renameVars r = foldExpr Num (Var . r) Sum         -- Umbenennung der Variablen
```

Ihre Aufgabe ist es, `foldExpr` in geeigneter Weise zu definieren, so dass die obigen drei Funktionsdefinitionen korrekt sind.

Aufgabe 4: Auswertung und Typen 20

Zu jeder Frage gibt es genau eine richtige Antwort. Das Ankreuzen dieser Antwort ergibt 4 Punkte; kein Kreuz zu machen ergibt 1 Punkt; das Ankreuzen einer falschen oder mehrerer Antworten ergibt 0 Punkte.

Setzen Sie Ihre Kreuze wie abgebildet: ☒.
Nutzen Sie ■, um ein Kreuz zu entfernen, d.h. ■ ist gleichbedeutend zu ☐.

Betrachten Sie folgendes Programm, was zum Teil aus Definitionen von vordefinierten Funktionen besteht.

```
-- Prelude
take n (x : xs) | n > 0 = x : take (n - 1) xs
take _ _ = []

takeWhile p (x : xs) | p x = x : takeWhile p xs
takeWhile _ _ = []

head (x : _) = x

-- neue Definition
times3 x = x : times3 (3 * x)
```

(a) Was ist der allgemeinste Typ von `times3`? (4)

- ☐ `Num a => [a]`
- ☐ `Int -> [Int]`
- ☐ `Num a => Int -> [a]`
- ☐ `Num a => a -> [a]`

(b) Welche Form der Rekursion liegt bei `times3` vor? (4)

- ☐ end-rekursiv
- ☐ guarded
- ☐ nicht-linear
- ☐ verschränkt rekursiv

(c) Was sind die ersten Schritte der Auswertung von `take 2 (times3 3)` bei lazy Evaluation? (4)

- ☐ `= take 2 (3 : times3 (3 * 3)) = 3 : take (2 - 1) (times3 (3 * 3))`
- ☐ `= take 2 (3 : times3 (3 * 3)) = take 2 (3 : times3 9)`
- ☐ `= take 2 (3 : times3 (3 * 3)) = take 2 (3 : 3 * 3 : times3 ((3 * 3) * (3 * 3)))`
- ☐ `= head (times3 3) : take 1 (tail (times3 3))`

(d) Was passiert, wenn Sie `takeWhile (> 0) (times3 (3 :: Int))` in `ghci` eingeben? (4)

- ☐ Es wird ein Typfehler angezeigt.
- ☐ Es gibt keine Rückmeldung, und Sie müssen eine unendliche Berechnung mittels CTRL-C abbrechen.
- ☐ Sie sehen, wie nach und nach eine Liste mit unendlich vielen Zahlen ausgeben wird, die Sie mittels CTRL-C abbrechen können.
- ☐ Es wird eine endliche Liste erzeugt und angezeigt, da es irgendwann zu einem Overflow bei den Zahlen kommt.

(e) Was ist das Resultat von `head (filter (< 0) (times3 (3 :: Integer)))`? (4)

- ☐ Es wird ein Typfehler angezeigt.
- ☐ Es gibt keine Rückmeldung, und Sie müssen eine unendliche Berechnung mittels CTRL-C abbrechen.
- ☐ Es gibt eine Fehlermeldung, weil `head []` nicht definiert ist.
- ☐ Es wird eine negative Zahl ausgegeben.