Probeklausur zur Mittelklausur

Programmierung 1 (WS 2020/21)

05.12.2020

Name: ______________________________

Matrikelnummer: ______________________

Sitzplatz: ___________________________

Prüfexemplar – nicht für den Druck

Öffnen Sie das Klausurheft erst dann, wenn Sie dazu aufgefordert werden.

Hilfsmittel sind nicht zugelassen. Am Arbeitsplatz dürfen nur Schreibgeräte, Getränke, Speisen und Ausweise mitgeführt werden. Taschen und Jacken müssen an den Wänden des Klausursaals zurückgelassen werden. Mobiltelefone, Smartwatches und andere elektronische Geräte die Kommunikation erlauben sind ebenfalls dort ausgeschaltet aufzubewahren.

Sowohl das Verlassen des Saals ohne Abgabe des Klausurhefts als auch das Öffnen dieser Klausur ohne das Unterschreiben dieser ersten Seite gelten als Täuschungsversuch. Wenn Sie während der Bearbeitung zur Toilette müssen, geben Sie bitte Ihr Klausurheft bei der Aufsicht ab. Es kann zu jeder Zeit nur eine Person zur Toilette.

Alle Lösungen müssen in der Regel auf den bedruckten rechten Seiten des Klausurhefts notiert werden. Die leeren linken Seiten dienen als Platz für Notizen. Als weiteres Notizpapier ist nur von uns ausgegebenes Papier zugelassen. Sollten Sie Lösungen auf den linken Seiten oder dem Notizpapier notieren, so müssen Sie diese deutlich als solche kenntlich machen und auf diese verweisen. Legen Sie zusätzliches Notizpapier bei der Abgabe der Klausur hinter Ihre Klausur, nicht in die Klausur, und beschriften Sie es mit Ihrem Namen und Ihrer Matrikelnummer. Weisen Sie bei der Abgabe darauf hin, dass Sie Notizpapier verwendet haben.

Für die Bearbeitung der Klausur stehen 90 Minuten zur Verfügung. Insgesamt können 90 Punkte und 6 Bonuspunkte erreicht werden.

Bitte legen Sie zur Identifikation Ihren Personalausweis bzw. Reisepass sowie Ihren Studierendenausweis neben sich.

Hiermit bestätige ich, dass ich die obigen Hinweise gelesen habe und insbesondere, dass ich prüfungsfähig bin.

____

Viel Erfolg!

1	2	3	4	5	Bonus	**Summe**
15	13	26	16	20	6	90+6

Note

Sie dürfen folgende Prozeduren verwenden:

- foldl : (α * β $\rightarrow$ β) $\rightarrow$ β $\rightarrow$ α list $\rightarrow$ β
- foldr : (α * β $\rightarrow$ β) $\rightarrow$ β $\rightarrow$ α list $\rightarrow$ β
- iter : int $\rightarrow$ α $\rightarrow$ (α $\rightarrow$ α) $\rightarrow$ α
- iterup : int $\rightarrow$ int $\rightarrow$ α $\rightarrow$ (int * α $\rightarrow$ α) $\rightarrow$ α
- iterdn : int $\rightarrow$ int $\rightarrow$ α $\rightarrow$ (int * α $\rightarrow$ α) $\rightarrow$ α
- first : int $\rightarrow$ (int $\rightarrow$ bool) $\rightarrow$ int
- map : (α $\rightarrow$ β) $\rightarrow$ α list $\rightarrow$ β list
- rev : α list $\rightarrow$ α list
- List.length : α list $\rightarrow$ int
- List.concat : α list list $\rightarrow$ α list
- List.tabulate : int * (int $\rightarrow$ α) $\rightarrow$ α list
- List.filter : (α $\rightarrow$ bool) $\rightarrow$ α list $\rightarrow$ α list
- List.exists : (α $\rightarrow$ bool) $\rightarrow$ α list $\rightarrow$ bool
- List.all : (α $\rightarrow$ bool) $\rightarrow$ α list $\rightarrow$ bool
- List.nth : (α list * int) $\rightarrow$ α
- hd : α list $\rightarrow$ α
- tl : α list $\rightarrow$ α list
- Int.compare : int * int $\rightarrow$ order
- null : α list $\rightarrow$ bool
- explode : string $\rightarrow$ char list
- implode : char list $\rightarrow$ string
- chr : int $\rightarrow$ char
- ord : char $\rightarrow$ int
- Math.sqrt : real $\rightarrow$ real

Aufgabe 1: Grundlagen **(15 Punkte)**

Aufgabe 1.1 (4 Punkte)

Betrachten Sie die folgende Prozedur:

```
fun f (x: int, y:int) = if x = 7 then ~y else ~42
```

(a) Geben Sie Argumentvariablen, Argumentmuster und Rumpf der Prozedur an

Argumentvariablen:

`x, y`

Argumentmuster:

`(x: int, y: int)`

Rumpf:
`if x = 7 then ~y else ~42`

(b) Geben Sie alle Bezeichner, Konstanten, Operatoren und Schlüsselwörter an, die in obigem Ausdruck vorkommen.

Bezeichner:

`f, x, y`

Konstanten:

`7, ~42`

Operatoren:

`=, ~`

Schlüsselwörter: `fun, ( , ), :, int, , ,=, if, then, else`

Aufgabe 1.2 (5 Punkte)

Die Prozedur `pythagoras : real * real → real` soll aus zwei Seitenlängen a und b eines Dreiecks die dritte Seitenlänge c nach dem Satz des Pythagoras ($a^2 + b^2 = c^2$) berechnen.

Beispiel: `pythagoras (3.0, 4.0) = 5.0`

Hinweis: Verwenden Sie `Math.sqrt`.

Schreiben Sie `pythagoras` auf drei Arten:

(a) Mit einem kartesischen Argumentmuster.

```
fun pythagoras (a : real, b : real) = Math.sqrt(a * a + b * b)
```

(b) Mit einer lokalen Deklaration.

```
fun pythagoras (p : real * real) = let
 val (a, b) = p
in
 Math.sqrt(a * a + b * b)
end
```

(c) Mithilfe von Projektionen.

```
fun pythagoras (p : real * real) = Math.sqrt(#1 p * #1 p + #2 p * #2 p)
```

Aufgabe 1.3 (6 Punkte)

Schreiben Sie eine endrekursive Prozedur `emb : int * int → int`, die semantisch äquivalent zur gegebenen Prozedur `mb` ist.

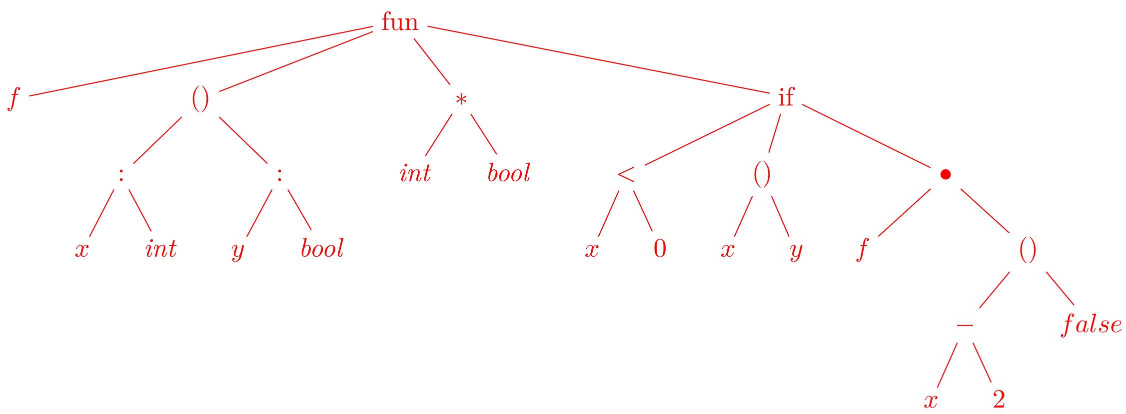
```
fun mb (x, c) = if x < 1 then 0 else mb(x - 1, c) * mb(x - 1, c) + c
```

```
fun emb (x, c) = let
 fun mb'(x, c, a) = if x < 1 then a else mb'(x - 1, c, a * a + c)
in
 mb'(x, c, 0)
end
```

Aufgabe 2: Programmiersprachliches (13 Punkte)

Aufgabe 2.1 *(Bäume für die Umwelt)* (4 Punkte)
Geben Sie die Baumdarstellung der folgenden durch ihre Zeichendarstellung beschriebene Phrase an:

```
fun f (x : int, y:bool) : int*bool = if x < 0 then (x,y) else f(x-2,false)
```



Aufgabe 2.2 *(Umgebungen)* (5 Punkte)
Welche Umgebung erhalten Sie nach einer Ausführung des folgenden Teilprogramms in der Umgebung `[y:=8]`?

```
val x = 4
val z = 10
fun p (a:int) = if a > y then 3 + p(a-4) else 0
val y = 5
val r = p(z)
```

```
[x:= 4, z:= 10,
 p:= (fun p a = if a > y then 3 + p(a-4) else 0, int → int, [y:=8]),
 y:= 5, r:= 3]
```

Aufgabe 2.3 *(Bereinigung)* (4 Punkte)
Bereinigen Sie folgendes Teilprogramm, indem Sie

- alle definierenden Bezeichnerauftreten überstreichen ($\overline{\texttt{a}}$)
- mit Pfeilen alle lexikalische Bindungen darstellen
- alle freien Bezeichner unterstreichen ($\underline{\texttt{a}}$)
- alle Bezeichner durch Indizieren konsistent umbenennen ($\texttt{a}_1$)

```
val x = fn y => x y (fn x => z x (fn z => z + y) z)
```

$$\texttt{val}\ \overline{x_1} = \texttt{fn}\ \overline{y_1} \Rightarrow \underline{x}\ y_1\ (\texttt{fn}\ \overline{x_2} \Rightarrow \underline{z}\ x_2\ (\texttt{fn}\ \overline{z_1} \Rightarrow z_1 + y_1)\ \underline{z})$$

Lexikalische Bindungen: Ziehe einen Pfeil von jedem indizierten, benutzenden Bezeichnerauftreten zu dem gleich indizierten, überstrichenen Bezeichnerauftreten.

Aufgabe 3: Höherstufige Prozeduren (26 Punkte)

Aufgabe 3.1 *(Primzahl Wetten)* (4+7+7 Punkte)
Dieter Schlau grübelt schon wieder über eine außergewöhnliche Frage. Nachdem er eine Prozedur `nextPrime : int → int` geschrieben hat, die ihm für jede Primzahl die nächstgrößere Primzahl ausgibt, möchte er folgendes wissen: Was ist die häufigste letzte Ziffer bei Primzahlen? Dieter hat sich bereits überlegt, dass nur die Ziffern $1, 3, 7, 9$ überhaupt infrage kommen (2 und 5 werden nicht betrachtet, da sie jeweils nur einmal auftreten). Nun möchte er die Vorkommen für die ersten n Primzahlen zählen.

(a) Schreiben Sie zunächst eine Prozedur `hasUnitsDigit : int * int → int`, welche eine positive Zahl und eine Ziffer als Argumente bekommt. Es soll 1 ausgegeben werden, falls die letzte Ziffer der Zahl der übergebenen Ziffer entspricht, und sonst 0.

> **Beispiel:** `hasUnitsDigit (356, 6)` soll 1 ausgeben, da 6 die letzte Ziffer von 356 ist.

```
fun hasUnitsDigit number digit = if number mod 10 = digit then 1 else 0
```

(b) Schreiben Sie eine Prozedur `countUnitsDigit : int → int * int * int * int`, welche das Zählen der Kandidaten $1, 3, 7, 9$ (in dieser Reihenfolge) für Dieter übernimmt. Also soll z.B. die erste Komponente des ausgegebenen Tupels der Anzahl der Einerziffer 1 in den ersten n Primzahlen entsprechen, wobei n der Prozedur übergeben wird.
Sie müssen dabei `iter` genau einmal verwenden. Außerdem dürfen Sie die Prozeduren `nextPrime` und `hasUnitsDigit` verwenden (ohne diese erneut zu deklarieren), jedoch dürfen Sie keine anderen Prozeduren aufrufen.

> **Beispiel:** Die ersten 9 Primzahlen sind: $2, 3, 5, 7, 11, 13, 17, 19, 23$
> `countRightDigit 9` soll also das Tupel `(1,3,2,1)` ausgeben.

```
fun countUnitsDigit n =
        let
                val (p, one, three, seven, nine) = iter n (2, 0, 0, 0, 0)
                        (fn (p, a, b, c, d) =>
                                (nextPrime p,
                                 a + hasUnitsDigit (p, 1),
                                 b + hasUnitsDigit (p, 3),
                                 c + hasUnitsDigit (p, 7),
                                 d + hasUnitsDigit (p, 9)))
        in
                (one, three, seven, nine)
        end
```

(c) Dieter möchte nun mit seinen Freunden Wetten abschließen, welche letzte Ziffer ($1, 3, 7$ oder 9) in den ersten Primzahlen am häufigsten auftritt.
Schreiben Sie nun eine Prozedur `winner : int → int`, welche diejenige letzte Ziffer ausgibt, die als Erste mindestens n mal in den ersten Primzahlen auftritt. Verwenden Sie dazu `first` und die in Teil (b) deklarierte Prozedur.

> **Beispiel:** `winner 3` soll also 3 ausgeben.

```
fun winner n =
        let
                val m = first 1 (fn x =>
                        let
                                val (a, b, c, d) = countUnitsDigit x
                        in
                                if a >= n orelse b >= n orelse c >= n orelse d >= n then
                                 ↳ true else false
                        end)
                val (a, b, c, d) = countUnitsDigit m
        in
                if a>=n then 1 else if b>=n then 3 else if c>=n then 7 else 9
        end
```

Aufgabe 3.2 (8 Punkte)

Die Prozedur `iterlist` : $\forall\alpha.$ `int` $\to \alpha \to (\alpha \to \alpha) \to \alpha$ `list` berechnet die Liste der Zustände während des iterativen Anwenden einer Prozedur. Das heißt konkret:

`iterlist 3 x f = [x, f x, f (f x), f (f (f x))]`

Implementieren Sie `iterlist` mit Hilfe von `iter`. `iterlist` darf selbst nicht rekursiv sein.

`iterup` Lösung:

```
fun iterlist n s f = iterup 0 n [] (fn (m, fs) => fs @ [iter m s f])
```

`iterdn` Lösung:

```
fun iterlist n s f = iterdn n 0 [] (fn (m, fs) => iter m s f :: fs)
```

`iter` Lösung:

```
fun iterlist n s f = iter n [s] (fn fs => s :: map f fs)
```

Effiziente Lösung:

```
fun iterlist n s f =
        rev (#2(iter n (s, [s])
            (fn (x, fs) =>
                let val x' = f x
                in (x', x' :: fs)
                end)))
```

Aufgabe 4: Listen **(16 Punkte)**

Aufgabe 4.1 *(BinKos)* (8+8 Punkte)
In dieser Aufgabe sollen Sie Binomialkoeffizienten mithilfe des *Pascal'schen Dreiecks* bestimmen.

Das Pascal'sche Dreieck besteht aus unendlich vielen Zeilen, die von 0 ab nummeriert sind. In der 0-ten Zeile steht eine einzelne Eins. In jeder weiteren Zeile stehen links und rechts Einsen; jeder weitere Eintrag wird als Summe der beiden Einträge schräg über ihm berechnet. In Abbildung 1 sehen Sie die ersten Zeilen des Dreiecks.

Wir stellen die einzelnen Zeilen als Listen ganzer Zahlen dar.

Zeile 0:						1					
Zeile 1:					1		1				
Zeile 2:				1		2		1			
Zeile 3:			1		3		3		1		
Zeile 4:		1		4		6		4		1	
Zeile 5:	1		5		10		10		5		1

Abbildung 1: Die ersten Zeilen des Pascal'schen Dreiecks.

(a) Schreiben Sie eine Prozedur `nextLine : int list` $\to$ `int list`, die zu einer gegebenen Zeile des Dreiecks die darauf folgende berechnet.

Beispiel: `nextLine [1] = [1, 1]` und `nextLine [1, 3, 3, 1] = [1, 4, 6, 4, 1]`

```
fun nextLine' (x::y::ys) = (x + y) :: nextLine' (y::ys)
  | nextLine' _ = [1]
fun nextLine xs = 1 :: nextLine' xs
```

(b) Der Binomialkoeffizient $\binom{n}{k}$ mit $n, k \in \mathbb{N}$ kann als der $(k+1)$-te Eintrag der n-ten Zeile bestimmt werden. Schreiben Sie mithilfe von `iter` eine *nicht-rekursive* Prozedur `binomial : int` $\to$ `int` $\to$ `int`, die $\binom{n}{k}$ für ein gegebenes n und k mit $k \leq n$ berechnet.

```
fun binomial n k = List.nth (iter n nextLine [1], k)
```

Aufgabe 5: Sortieren (20 Punkte)

Lesen Sie zunächst die gesamte Aufgabenstellung, bevor Sie mit der Bearbeitung beginnen.

In dieser Aufgabe sollen Sie natürliche Zahlen mit **maximal n Stellen** sortieren. Dabei werden Sie schrittweise das Sortierverfahren "Sortieren durch Partitionieren" entwickeln.

Vorgehen des Verfahrens:
Sortieren durch Partitionieren arbeitet in mehreren Schritten. Wir zeigen die Arbeitsweise hier exemplarisch an einem Beispiel:

Zu sortierende Liste:

```
[233, 321, 223]
```

Man betrachtet nun zunächst die erste Ziffer (Ziffer **0**) jeder Zahl und teilt (*partitioniert*) diese entsprechend in eine von 10 (für die Ziffern 0-9) Teillisten ein.

```
[[], [321], [], [233, 223], [], [], [], [], []]
```

Diese werden wieder zu einer Liste "*eingesammelt*", dabei ist es essenziell für ein korrektes Ergebnis, dass die Reihenfolge der Zahlen **nicht verändert wird**. Es ergibt sich:

```
[321, 233, 223]
```

Diese beiden Schritte "**Partitionieren**" und "**Einsammeln**" werden nun wiederholt, bis alle Stellen betrachtet wurden.

Zweite Ziffer:

```
Partitionieren: [[], [], [321, 223], [233], [], [], [], [], []]
Einsammeln: [321, 223, 233]
```

Dritte Ziffer:

```
Partitionieren: [[], [], [223, 233], [321], [], [], [], [], []]
Einsammeln: [223, 233, 321]
```

Alle Ziffern sind betrachtet, der Algorithmus ist abgeschlossen und die Liste sortiert.

Aufgabe 5.1 (4+8+8 Punkte)

(a) Deklarieren Sie eine Prozedur `getNthDigit : int → int → int`, welche gegeben zwei natürliche Zahlen m und n, die m-te Ziffer von n beginnend bei 0 zurückgibt. Sie dürfen eine Prozedur `pow : int * int → int`, welche die Potenzfunktion $(x, y) \mapsto x^y$ berechnet, als gegeben annehmen.
Beispielsweise soll `getNthDigit 1 4532` zu 3 auswerten.

Vordeklariert:

```
fun iterup m n s f = if m>n then s else iterup (m+1) n (f(m,s)) f
fun pow (x, y) = Real.floor(Math.pow(Real.fromInt(x), Real.fromInt(y)))
```

```
fun getNthDigit n x = (x div pow(10, n)) mod 10
```

(b) Deklarieren Sie nun mithilfe Ihrer Prozedur aus (a) eine Prozedur `partition : int → int list → int list list`, welche eine Liste L anhand der n-ten Ziffer der Zahlen in Teillisten einteilt (*partitioniert*). Beispielweise soll `partiton 2 [4832, 5624, 8134, 1145]` zu `[[], [8134, 1145], [], [], [], [], [5624], [], [4832], []]` auswerten.

```
fun partition n list =
        iterup 0 9 nil
            (fn (m, s) => s @
                [foldl (fn (x, s) => if getNthDigit n x = m then s @ [x] else s)
                       nil list]
            )
```

(c) Deklarieren Sie zuletzt mithilfe der Prozedur aus (b) die Prozedur `sort : int → int list → int list`, welche eine Liste L mit natürlichen Zahlen mit maximal n Ziffern sortiert. Beispielweise soll `sort 4 [4832, 5624, 8134, 1145]` zu `[1145, 4832, 5624, 8134]` auswerten.

```
fun sort n list = iterup 0 (n-1) list (fn (m, s) => List.concat (partition m s))
```

Aufgabe 6: Bonus (6 Punkte)

Die Punkte dieser Aufgabe sind Bonuspunkte und gehören also nicht zur Gesamtpunktzahl, die über die Notengrenzen entscheidet. Erreichte Punkte werden zu der von Ihnen erreichten Punktzahl addiert.

Aufgabe 6.1 *(Zahlensysteme)* (6 Punkte)

Dieter Schlau jongliert gerne mit Zahlen in verschiedenen Zahlensystemen (z.B. Binär-, Dezimal- und Hexadezimalsystem). Allgemein können Zahlen zu einer beliebigen Basis b (z.B. $b = 2$, $b = 10$ oder $b = 16$) wie folgt dargestellt werden:

$$X_b = (x_n x_{n-1} \dots x_1 x_0)_b = \sum_{i=0}^{n} x_i \cdot b^i$$

z.B.

$$1893 = 1 \cdot 10^3 + 8 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0$$

Deklarieren Sie **mit Hilfe von Faltung** eine Prozedur `toInt : int → string → int`, welche eine Zahl, dargestellt als `string` in gegebener Basis, zu einem `int` umrechnet.
Hierfür werden Ziffern die Größer als 9 sind durch die Buchstaben A-Z dargestellt.

> **Beispiel:**
> `toInt 10 "12" = 12`
> `toInt 16 "2A" = 42`
> `toInt 2 "00101000" = 40`

Dieter Schlau hat für Sie bereits die Prozedur `charToInt : char → int` deklariert, welche einen `char` in einen `int` umrechnet.

> **Beispiel:**
> `charToInt #"5" = 5`
> `charToInt #"A" = 10`
> `charToInt #"C" = 12`

Ihre Prozedur soll nicht rekursiv sein und darf nur `charToInt`, `foldl` und `foldr` als Hilfsprozeduren verwenden.

```
fun toInt b s = foldl (fn (c, n) => n * b + charToInt c) 0 (explode s)
```