Aufgabe 1 Statische Semantik (__ /20 Punkte)

a) Füllen Sie die Lücken, sodass sich **gültige Aussagen der Statischen Semantik** ergeben. Alle vorkommenden Zahlen sind vom Typ `Nat`. Sie müssen **keine Beweisbäume** angeben!

Hinweis: In einigen Teilaufgaben gibt es mehrere richtige Lösungen. Die angegebene Lösung ist nur ein Beispiel.

i) $\emptyset \vdash$ `4711 * 815` : _____ **__ /10**

$\emptyset \vdash$ `4711 * 815 : Nat`

ii) $\emptyset \vdash$ _____ : `Nat * Nat`

$\emptyset \vdash$ `(4711, 815) : Nat * Nat`

iii) $\emptyset \vdash$ **`if`** _____ **`then`** `4711` **`else`** _____ : Nat

$\emptyset \vdash$ **`if true then`** `4711` **`else`** `815 : Nat`

iv) $\emptyset \vdash$ **`let rec`** `f (x: Bool): Nat = f x` **`in`** `f` : _____

$\emptyset \vdash$ **`let rec`** `f (x: Bool): Nat = f x` **`in`** `f` : $\texttt{Bool} \to \texttt{Nat}$

v) $\{\, \texttt{f} \mapsto (\texttt{Nat} \to \texttt{Bool}) \,\} \vdash$ **`let`** `g (x: Nat): Bool = f x` : _____

$\{\, \texttt{f} \mapsto (\texttt{Nat} \to \texttt{Bool}) \,\} \vdash$ **`let`** `g (x: Nat): Bool = f x` : $\{\, \texttt{g} \mapsto \texttt{Nat} \to \texttt{Bool} \,\}$

b) Geben Sie einen **vollständigen Beweisbaum** für folgende **Aussage der Statischen Semantik** an: **__ /10**

$$\{\texttt{f} \mapsto (\text{Nat} \to \text{Nat})\} \vdash \texttt{f (f 4711)} : \text{Nat}$$

$$\dfrac{\dfrac{}{\{\texttt{f} \mapsto \text{Nat} \to \text{Nat}\} \vdash \texttt{f} : \text{Nat} \to \text{Nat}} \qquad \dfrac{\dfrac{}{\{\texttt{f} \mapsto \text{Nat} \to \text{Nat}\} \vdash \texttt{f} : \text{Nat} \to \text{Nat}} \qquad \dfrac{}{\{\texttt{f} \mapsto \text{Nat} \to \text{Nat}\} \vdash \texttt{4711} : \text{Nat}}}{\{\texttt{f} \mapsto \text{Nat} \to \text{Nat}\} \vdash \texttt{f 4711} : \text{Nat}}}{\{\texttt{f} \mapsto \text{Nat} \to \text{Nat}\} \vdash \texttt{f (f 4711)} : \text{Nat}}$$

Aufgabe 2 Dynamische Semantik (__ /20 Punkte)

a) Füllen Sie die Lücken, sodass die Ausdrücke **typkorrekt** sind und sich **gültige Aussagen der Dynamischen Semantik** ergeben. Alle vorkommenden Zahlen sind vom Typ `Nat`. Sie müssen **keine Beweisbäume** angeben!

Hinweis: In einigen Teilaufgaben gibt es mehrere richtige Lösungen. Die angegebene Lösung ist nur ein Beispiel.

i) $\emptyset \vdash \texttt{fun x} \to \texttt{fun y} \to \texttt{fun z} \to \texttt{if x then y else z} \Downarrow$ _____ **__ /10**

$$\frac{}{\emptyset \vdash \texttt{fun x} \to \texttt{fun y} \to \texttt{fun z} \to \texttt{if x then y else z} \Downarrow \langle \emptyset, \texttt{x}, \texttt{fun y} \to \texttt{fun z} \to \texttt{if x then y else z} \rangle}$$

Man kann also einfach einen Strich über die ausgefüllte Aussage malen, um Teilaufgabe b zu lösen.

ii) $\emptyset \vdash \texttt{fst (snd (}$ _____ $\texttt{))} \Downarrow 4711$

$$\frac{\dfrac{\dfrac{}{\emptyset \vdash \texttt{false} \Downarrow \texttt{false}} \quad \dfrac{\dfrac{}{\emptyset \vdash 4711 \Downarrow 4711} \quad \dfrac{}{\emptyset \vdash \texttt{true} \Downarrow \texttt{true}}}{\emptyset \vdash \texttt{(4711, true)} \Downarrow (4711, \texttt{true})}}{\emptyset \vdash \texttt{(false, (4711, true))} \Downarrow (\texttt{false}, (4711, \texttt{true}))}}{\emptyset \vdash \texttt{snd (false, (4711, true))} \Downarrow (4711, \texttt{true})}}{\emptyset \vdash \texttt{fst (snd (}\underline{\texttt{false, (4711, true)}}\texttt{))} \Downarrow 4711}$$

iii) $\{\texttt{x} \mapsto 2\} \vdash \texttt{x + (let x = x +}$ _____ $\texttt{in x + 1)} \Downarrow 6$

$$\frac{\dfrac{}{\{\texttt{x} \mapsto 2\} \vdash \texttt{x} \Downarrow 2} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\{\texttt{x} \mapsto 2\} \vdash \texttt{x} \Downarrow 2} \quad \dfrac{}{\{\texttt{x} \mapsto 2\} \vdash 1 \Downarrow 1}}{\{\texttt{x} \mapsto 2\} \vdash \texttt{x + 1} \Downarrow 3}}{\{\texttt{x} \mapsto 2\} \vdash \texttt{let x = x + 1} \Downarrow \{\texttt{x} \mapsto 3\}} \quad \dfrac{\dfrac{}{\{\texttt{x} \mapsto 3\} \vdash \texttt{x} \Downarrow 3} \quad \dfrac{}{\{\texttt{x} \mapsto 3\} \vdash 1 \Downarrow 1}}{\{\texttt{x} \mapsto 3\} \vdash \texttt{x + 1} \Downarrow 4}}{\{\texttt{x} \mapsto 2\} \vdash \texttt{let x = x + 1 in x + 1} \Downarrow 4}}{\{\texttt{x} \mapsto 2\} \vdash \texttt{x + (let x = x +} \underline{\texttt{1}} \texttt{ in x + 1)} \Downarrow 6}$$

iv) $\emptyset \vdash$ _____ $\Downarrow \{\texttt{f} \mapsto \langle \emptyset, \texttt{f}, \texttt{x}, \texttt{if x = 0 then 0 else x + f (x - 1)} \rangle\}$

$$\frac{}{\emptyset \vdash \underline{\texttt{let rec f x = if x = 0 then 0 else x + f (x - 1)}} \Downarrow \{\texttt{f} \mapsto \langle \emptyset, \texttt{f}, \texttt{x}, \texttt{if x = 0 then 0 else x + f (x - 1)} \rangle\}}$$

Man kann also einfach einen Strich über die ausgefüllte Aussage malen, um Teilaufgabe b zu lösen.

v) $\{\texttt{x} \mapsto$ _____ $\} \vdash \texttt{x < 4711} \Downarrow \texttt{true}$

$$\frac{\dfrac{}{\{\texttt{x} \mapsto 42\} \vdash \texttt{x} \Downarrow 42} \quad \dfrac{}{\{\texttt{x} \mapsto 42\} \vdash 4711 \Downarrow 4711}}{\{\texttt{x} \mapsto \underline{42}\} \vdash \texttt{x < 4711} \Downarrow \texttt{true}}$$

b) **Wählen Sie** von den fünf Aussagen aus Teilaufgabe a) **eine aus** und geben Sie **nur für diese Aussage** einen **vollständigen Beweisbaum** mit den Regeln der Dynamischen Semantik an.

Tipp: Die Bäume für die fünf Aussagen werden unterschiedlich groß. Machen Sie sich kurz Gedanken darüber, welche Aussage zu einem kleinen Baum führt. **__ /10**

Sie können Ihren Beweisbaum oben in die Zwischenräume schreiben und dabei die eigentliche Aussage als Teil des Baumes verwenden. Falls Ihnen der Platz nicht ausreicht, können Sie die Rückseite benutzen.

Aufgabe 3 Entwurfsmuster (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

a) Schreiben Sie eine Funktion `fib: Nat -> Nat * Nat`, die die `n`- und `n+1`-te Fibonacci-Zahl zurückgibt. Zur Erinnerung, die Fibonacci-Zahlen erfüllen folgende Rekursionsgleichungen:

$$F(0) = 0$$
$$F(1) = 1$$
$$F(n) = F(n-2) + F(n-1)$$

Die ersten paar Fibonacci Zahlen sind also:

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

Gehen Sie **strikt nach Peano Entwurfsmuster** vor! *NB:* Dies heißt insbesondere, dass Sie `fib` im Definitionsrumpf von `fib` nur einmal aufrufen dürfen!

Beispiele:

```
fib 0 = (0,1)          fib 1 = (1,1)          fib 5 = (5,8)
```

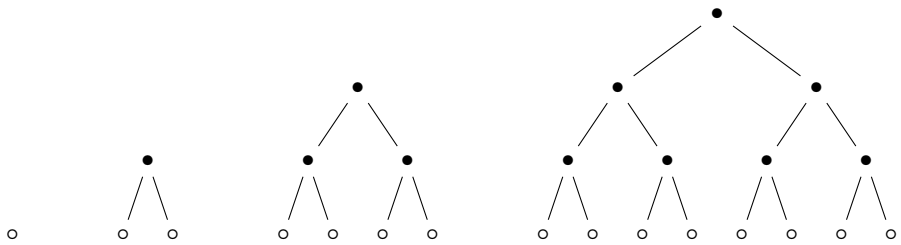
__ /5

```
let rec fib (n : Nat) : Nat * Nat =
    if n = 0N then (0N , 1N)
    else let (fnMin2 , fnMin1) = fib (n - 1N) in (fnMin1, fnMin2 + fnMin1)
```

b) Gegeben sei folgender Typ für Binärbäume:

```
type BTree = Leaf | Branch of BTree * BTree
```

Schreiben Sie eine Funktion `complete: Nat -> BTree`, die den `n`-ten vollständigen Binärbaum zurückgibt. Unten sind graphische Darstellungen der ersten 4, wobei Blätter durch weiß schattierte Knoten dargestellt sind.



Gehen Sie **strikt nach Peano Entwurfsmuster** vor.

Beispiele:

```
complete 0 = Leaf
complete 1 = Branch (Leaf, Leaf)
complete 2 = Branch (Branch (Leaf, Leaf), Branch (Leaf, Leaf))
```

```
let rec complete (n : Nat) : BTree =
  if n = 0N then Leaf
  else let o = complete (n - 1N) in Branch (o , o)
```

__/5

Für die nächsten beiden Teilaufgaben verwenden wir folgende Funktionsdefinition.

```
let fold<'a, 'b> (z: 'b) (f: 'a -> 'b -> 'b) : List<'a> -> 'b =
    let rec worker (t: List<'a>): 'b =
        match t with
        | [] -> z
        | x :: xs -> f x (worker xs)
    in worker
```

Die Funktion `fold` abstrahiert das Struktur-Entwurfsmuster für Listen, analog zu `peano-pattern` aus der Vorlesung. Sie nimmt einen Wert `z` sowie eine Funktion `f`. Der Wert `z` ist der Rückgabewert des Basisfalls (`[]`), während `f` das Listenelement sowie den Rückgabewert des rekursiven Aufrufs auf die Restliste verarbeitet.

c) Schreiben Sie eine Funktion `size<'a>: List<'a> -> Nat`, die die Anzahl der Elemente einer Liste zählt. Verwenden Sie hierfür die Funktion `fold`.

Beispiele:

```
size [] = 0                 size [6] = 1                 size [3;4;5] = 3
```

```
let size<'a> : List<'a> -> Nat = fold 0N (fun _ n -> 1N + n)
```

__/5

d) Schreiben Sie eine Funktion `maximum: List<Nat> -> Nat`, die das Maximum der Elemente einer Liste von natürlichen Zahlen berechnet. Verwenden Sie hierfür die Funktion `fold`. Wir definieren als das Maximum der leeren Liste `0`, das neutrale Element von `max` auf `Nat`.

Beispiele:

```
maximum [] = 0              maximum [6] = 6              maximum [4;3;5] = 5
```

```
let maximum: List<Nat> -> Nat = fold 0N max
```

__/5

Aufgabe 4 Multimengen (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

Wir betrachten den folgenden Typ, um Multimengen zu modellieren:

```
type Bag<'a> = 'a -> Nat
```

Ist `m` vom Typ `Bag<'a>` und `x` vom Typ `'a`, so gibt `m x` an, wie oft das Element `x` in der Multimenge `m` vorkommt.

Zur Erinnerung: Eine Multimenge (engl. multiset oder bag) ist im Prinzip eine Menge, in der Elemente mehrfach vorkommen können. Die Anzahl der Vorkommen eines Elements wird als *Vielfachheit* bezeichnet.

Hinweis: Sie dürfen davon ausgehen, dass Elemente vom Typ `'a` auf Gleichheit getestet werden können.

a) Schreiben Sie eine Funktion `multiplicity<'a>: Bag<'a> -> 'a -> Nat`, die die Vielfachheit eines Elements in einer Multimenge zurückgibt.

```
let multiplicity<'a> (m: Bag<'a>) (x: 'a): Nat =
    m x
```

__ /5

b) Schreiben Sie eine Funktion `singleton<'a>: 'a -> Bag<'a>`, die ein Element vom Typ `'a` nimmt und eine Multimenge zurückgibt, die genau dieses Element einmal enthält.

```
let singleton<'a when 'a: equality> (x: 'a): Bag<'a> =
    fun (y: 'a) -> if x = y then 1N else 0N
```

__ /5

c) Schreiben Sie eine Funktion `union<'a>: Bag<'a> -> Bag<'a> -> Bag<'a>`, die zwei Multimengen nimmt und die Vereinigung der beiden zurückgibt.

```
let union<'a> (m1: Bag<'a>) (m2: Bag<'a>): Bag<'a> =
    fun (x: 'a) -> m1 x + m2 x
```

__ /5

d) Schreiben Sie eine Funktion `insert<'a>: 'a -> Bag<'a> -> Bag<'a>`, die ein Element in eine Multimenge einfügt.

```
let insert<'a when 'a: equality> (x: 'a) (m: Bag<'a>): Bag<'a> =
    union (singleton x) m
```

__ /5

Aufgabe 5 Ternärbäume (__ /20 Punkte)

Sie dürfen zur Lösung dieser Aufgabe Bibliotheksfunktionen verwenden.

Gegeben ist der folgende Typ für ternäre Bäume:

```
type Tree<'a> = | Leaf | Node of 'a * Tree<'a> * Tree<'a> * Tree<'a>
```

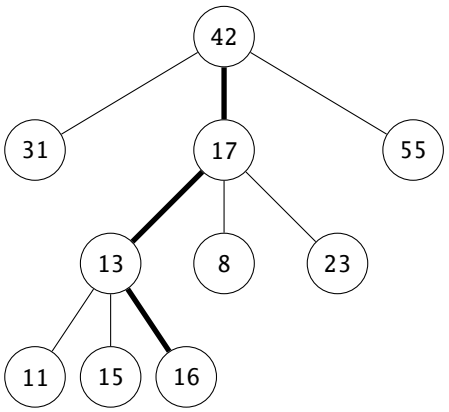
Darüber hinaus ist der folgende Typ für Pfade gegeben:

```
type Dir = | L | M | R

type Path = List<Dir>
```

Ein Pfad ist eine Liste von Richtungen (`Dir`). Da es sich um Ternärbäume handelt, gibt es drei Richtungen: Links (`L`), Mitte (`M`) und Rechts (`R`).

Beispiel: Rechts sehen Sie den Code zu dem hier dargestellten Baum und dem hervorgehobenen Pfad mit den Knoten 42, 17, 13, 16. Zur bessern Übersichtlichkeit sind die Blätter (`Leaf`s) nicht eingezeichnet.




```
let exTree: Tree<Nat> =
    Node(42,
        Node(31, Leaf, Leaf, Leaf),
        Node(17,
            Node(13,
                Node(11, Leaf, Leaf, Leaf),
                Node(15, Leaf, Leaf, Leaf),
                Node(16, Leaf, Leaf, Leaf)
            ),
            Node(8, Leaf, Leaf, Leaf),
            Node(23, Leaf, Leaf, Leaf)
        ),
        Node(55, Leaf, Leaf, Leaf)
    )
let exPath: Path = [M; L; R]
```

a) Schreiben Sie eine Funktion `preorder<'a>: Tree<'a> -> List<'a>`, die die Elemente eines Baums in der Preorder-Reihenfolge zurückgibt. Ihre Funktion soll in *linearer Laufzeit* arbeiten. (Für langsamere Implementierungen gibt es nur Teilpunkte.)

Zur Erinnerung: Bei der Preorder-Traversierung wird der Wurzelknoten zuerst besucht, dann der linke Teilbaum, dann der mittlere Teilbaum und zuletzt der rechte Teilbaum.

Hinweis: Schreiben Sie eine Hilfsfunktion, die allgemeiner ist als `preorder`.

Beispiele:

```
preorder Leaf = []
preorder (Node (2, Leaf, Node (1, Leaf, Leaf, Leaf), Leaf)) = [2; 1]
preorder exTree = [42; 31; 17; 13; 11; 15; 16; 8; 23; 55]
```

__/7

```
let preorder<'a> (t: Tree<'a>): List<'a> =
    let rec go (t: Tree<'a>) (acc: List<'a>) =
        match t with
        | Leaf -> acc
        | Node (x, l, m, r) -> x :: go l (go m (go r acc))
    go t []
```

b) Schreiben Sie eine Funktion `lookup<'a>: Tree<'a> -> Path -> Option<'a>`, die zu einem Pfad das Element zurückgibt, auf das der Pfad verweist. Ist der Pfad ungültig, soll `None` zurückgegeben werden.

Beispiele:

```
lookup exTree [M; L; R] = Some 16
lookup exTree [M; R]    = Some 23
lookup exTree [L; M]    = None
```

__/6

```
let rec lookup<'a> (t: Tree<'a>) (p: Path): Option<'a> =
    match (t, p) with
    | (Leaf, _) -> None
    | (Node (x, _, _, _), []) -> Some x
    | (Node (_, t', _, _), L :: p') // -> lookup t' p'
    | (Node (_, _, t', _), M :: p') // -> lookup t' p'
    | (Node (_, _, _, t'), R :: p') -> lookup t' p'
```

c) Schreiben Sie eine Funktion `search<'a>: ('a -> Bool) -> Tree<'a> -> List<Path>`, die alle Pfade zurückgibt, die zu einem Knoten führen, für den das gegebene Prädikat gilt.

Hinweis: Die Reihenfolge der Pfade spielt keine Rolle.

Beispiele:

```
search (fun n -> n = 4711)  exTree = []
search (fun n -> n > 50)    exTree = [[R]]
search (fun n -> n % 2 = 0) exTree = [[]; [M; M]; [M; L; R]]
```

__/7

```
// Top-down: Pfade beim rekursiven Abstieg konstruieren
let search<'a> (p: 'a -> Bool) (t: Tree<'a>): List<Path> =
    let rec go (t: Tree<'a>) (pth: Path) (acc: List<Path>) =
        match t with
        | Leaf -> acc
        | Node (x, l, m, r) ->
            (if p x then [pth] else [])
            @ go l (pth @ [L]) (go m (pth @ [M]) (go r (pth @ [R]) acc))
    go t [] []

// Bottom-up: Pfade beim rekursiven Aufstieg korrigieren
let rec search'<'a> (p: 'a -> Bool) (t: Tree<'a>): List<Path> =
    match t with
    | Leaf -> []
    | Node (x, l, m, r) ->
        (if p x then [[]] else [])
        @ List.map (fun p -> L :: p) (search' p l)
        @ List.map (fun p -> M :: p) (search' p m)
        @ List.map (fun p -> R :: p) (search' p r)
```

Aufgabe 6 Reguläre Ausdrücke (__ /20 Punkte)

a) Wir betrachten den regulären Ausdruck

$$\left((\mathsf{a} \cdot \mathsf{b})^* \cdot \mathsf{c}\right)^* \mid \left((\mathsf{b} \cdot \mathsf{a})^* \cdot \mathsf{c}\right)^*$$

über dem Alphabet $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$.

Kreuzen Sie an, ob die folgenden Wörter in der von dem Ausdruck beschriebenen Sprache enthalten sind oder nicht. Für richtige Antworten erhalten Sie einen Punkt, für falsche Antworten wird ein Punkt abgezogen. Nicht markierte Zeilen wirken sich nicht auf die Punktzahl aus. Diese Teilaufgabe wird mit mindestens 0 Punkten bewertet.

__ /6

Wort	enthalten	nicht enthalten
abcabcabcabca		X
ababcababcabc	X	
bababaccbabac	X	
abcbacabcbacc		X
ccccccccccccc	X	
ababababababa		X

b) Wir erweitern die Syntax für reguläre Ausdrücke:

$$\begin{array}{rcll} r \in \mathrm{Reg} & ::= & \ldots & \text{bisherige Definitionen} \\ & \mid & r? & \text{optionales Vorkommen} \end{array}$$

Die Denotationelle Semantik ist wie folgt definiert:

$$[\![r?]\!] = [\![r]\!] \cup \{\epsilon\}$$

Geben Sie die zusätzlich benötigte(n) Regel(n) der Reduktionssemantik an.

__ /4

Lösung mit zwei Regeln: $\overline{r? \to r}$ $\overline{r? \to \epsilon}$

Lösung mit einer Regel: $\overline{r? \to r \mid \epsilon}$

c) Bestimmen Sie die folgenden Rechtsfaktoren. Geben Sie in der Rechnung **jeweils den ersten Schritt explizit** an, nachfolgende Zwischenschritte dürfen Sie zusammenfassen.

Alphabet: $\{x, y, z\}$

__ / 10

$$x \setminus \left((x \cdot y)^* \cdot (y \cdot x)^*\right) = \left(x \setminus (x \cdot y)^*\right) \cdot (y \cdot x)^* \mid x \setminus (y \cdot x)^*$$
$$= y \cdot (x \cdot y)^* \cdot (y \cdot x)^*$$

$$y \setminus \left((x \cdot y)^* \cdot (y \cdot x)^*\right) = \left(y \setminus (x \cdot y)^*\right) \cdot (y \cdot x)^* \mid y \setminus (y \cdot x)^*$$
$$= x \cdot (y \cdot x)^*$$

$$x \setminus \left(\left((x \mid y)^* \cdot z\right)^*\right) = \left(x \setminus \left((x \mid y)^* \cdot z\right)\right) \cdot \left((x \mid y)^* \cdot z\right)^*$$
$$= (x \mid y)^* \cdot z \cdot \left((x \mid y)^* \cdot z\right)^*$$

$$y \setminus \left((y \cdot x) \mid \left((y \cdot y) \mid (y \cdot z)\right)\right) = \left(y \setminus (y \cdot x)\right) \mid \left(y \setminus \left((y \cdot y) \mid (y \cdot z)\right)\right)$$
$$= x \mid y \mid z$$

$$z \setminus \left(z \mid (z \cdot y)\right) = (z \setminus z) \mid \left(z \setminus (z \cdot y)\right)$$
$$= \epsilon \mid y$$

Aufgabe 7 Ringpuffer (__ /20 Punkte)

Sie dürfen zur Lösung dieser Aufgabe Bibliotheksfunktionen verwenden.

Ein Ringpuffer ist ein Speicher mit fester Kapazität, in den kontinuierlich Daten gespeichert und wieder herausgenommen werden können. Er funktioniert wie eine Warteschlange: Es wird immer dasjenige Element herausgenommen, das sich schon am längsten im Ringpuffer befindet.

Der Ringpuffer-Typ ist wie folgt definiert:

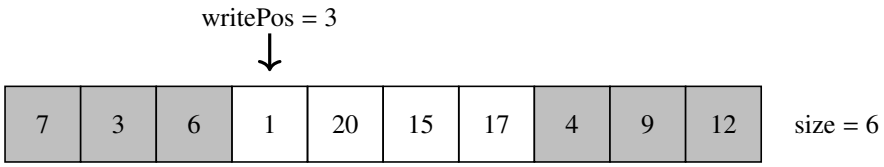
```
type RingBuffer<'a> =
    { buffer: Array<'a>
      size: Ref<Int>
      writePos: Ref<Int> }
```

Dabei speichert das Array `buffer` die aktuellen Elemente. Die Gesamtkapazität des Ringpuffers ist durch die Länge des Arrays gegeben. Die Anzahl der aktuell gespeicherten Elemente ist durch `size` gegeben. Wir speichern die Schreibposition mit `writePos`.

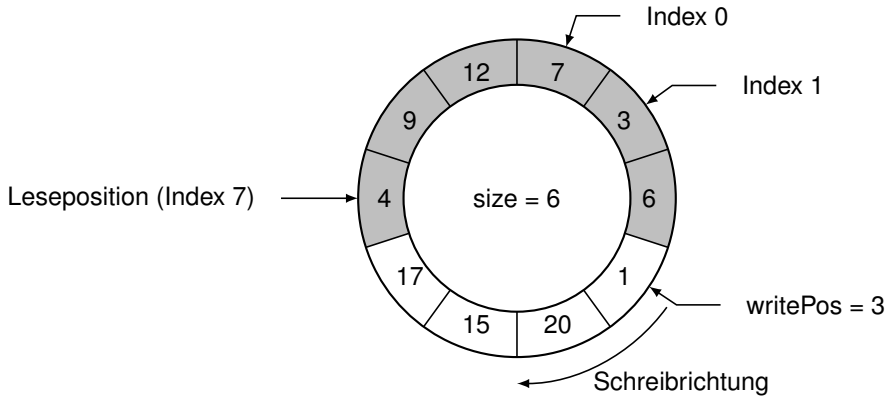
Die Elemente im Ringpuffer sind die `size` Elemente links von `writePos`. Nach dem letzten Index des Arrays wird zyklisch bei Index `0` fortgesetzt – sowohl beim Lesen als auch beim Schreiben.

Die folgende Abbildung veranschaulicht einen `RingBuffer<Int>` mit Kapazität 10 und Größe 6.

Der Ringpuffer enthält sechs Elemente (graue Felder, gelesen wird in der Reihenfolge 4, 9, 12, 7, 3, 6) und es können noch vier weitere Elemente gespeichert werden (dabei werden die weißen Felder 1, 20, 15 und 17 überschrieben).



Als Ring dargestellt wird der Übergang zwischen Anfang und Ende des Arrays deutlicher:



Zur Fehlerbehandlung führen wir außerdem folgende Ausnahmen ein:

```
exception BufferEmpty
exception BufferFull
```

a) Schreiben Sie eine Funktion `create<'a>: Int -> RingBuffer<'a>`, die einen leeren Ringpuffer der gegebenen Kapazität erstellt.

Hinweis: Verwenden Sie die Funktion `Array.zeroCreate<'a>: Int -> Array<'a>`, um ein Array der gegebenen Länge zu erstellen.

__/6

```
let create<'a> (capacity: Int): RingBuffer<'a> =
    { buffer = Array.zeroCreate<'a> capacity
      size = ref 0
      writePos = ref 0 }
```

b) Schreiben Sie eine Funktion `get<'a>: RingBuffer<'a> -> 'a`, welche das jeweils nächste im Puffer enthaltene Element zurückgibt. Dabei soll `size` um 1 verringert werden soll.

Falls keine zu lesenden Elemente (graue Felder) mehr vorhanden sind, soll der Ringpuffer nicht verändert und eine `BufferEmpty` Ausnahme geworfen werden.

__/7

```
let get<'a> (r: RingBuffer<'a>): 'a =
    if !r.size > 0 then
        let capacity = r.buffer.Length
        let readPos = (capacity + !r.writePos - !r.size) % capacity
        r.size := !r.size - 1
        r.buffer.[readPos]
    else
        raise BufferEmpty
```

c) Schreiben Sie eine Funktion `put<'a>: RingBuffer<'a> -> 'a -> Unit`, welche ein Element `elem` auf den Ringpuffer schreibt und den Zustand des Ringpuffers entsprechend anpasst. Falls der Ringpuffer voll ist (keine weißen Felder), soll eine `BufferFull` Ausnahme geworfen und der Ringpuffer nicht verändert werden.

__/7

```
let put<'a> (r: RingBuffer<'a>) (elem: 'a): Unit =
    if !r.size < r.buffer.Length then
        r.buffer.[!r.writePos] <- elem
        r.size := !r.size + 1
        r.writePos := (!r.writePos + 1) % r.buffer.Length
    else
        raise BufferFull
```

Aufgabe 8 Ausnahmen (__ /20 Punkte)

a) Unter Berücksichtigung dieser Typ- und Ausnahmedefinitionen

```
type BA = | B of Nat | A

exception E of Nat
exception F of Bool
```

betrachten wir den folgenden Ausdruck. Dabei ist `f` eine Funktion vom Typ `Unit -> BA`.

```
try
    match f() with
    | B x -> if x >= 10 then raise (E x) else x
    | A   -> raise (F true)
with
| E x -> if x = 0 then x else raise (F false)
| F x -> if not x then 4711 else raise (E 7)
```

Bestimmen Sie für die folgenden fünf Implementierungen der Funktion `f` jeweils, zu welchem Wert obiger Ausdruck auswertet. Notieren Sie geworfene Ausnahmen dabei, wie in der Vorlesung eingeführt, mit einem Kästchen, die durch `raise (E 123)` geworfene Ausnahme also durch `E 123`.

1. `let f() = B 0` **__ /10**

   `0`

2. `let f() = B 4711`

   `F false`

3. `let f() = A`

   `E 7`

4. `let f() = raise (E 99)`

   `F false`

5. `let f() = raise (F false)`

   `4711`

b) Wir betrachten die Programmierschnittstelle eines Online-Shops. Diese stellt folgende Typen bereit. **__/10**

```
type Item  = {gid : Nat; name : string}
type Query = {item : Item; quantity : Nat}

exception NoSuchItem
exception InsufficientStock of Nat
```

Sie stellt des weiteren folgende Funktion zur Verfügung.

`putQuery : Query -> Unit`	Führt die übergebene Nachfrage aus. Wirft `NoSuchItem` falls der angefragte Artikel nicht existiert, `InsufficientStock n` falls der Lagerbestand des Artikels geringer ist als die angefragte Menge, wobei `n` der Restbestand ist.

Im folgenden sollen Sie zwei unterschiedliche *Ausnahmebehandler* schreiben. Diese Ausnahmebehandler bekommen eine Funktion des Typs `Query -> Unit` wie z.B. `putQuery` als Argument übergeben, und gehen auf verschiedene Weise mit den möglicherweise bei ihrer Auswertung geworfenen Ausnahmen um.

1. Definieren Sie einen Ausnahmebehandler der die übergebene Anfrage ausführt, und in dem Fall, dass der Artikel nicht existiert, oder der Restbestand zu gering ist, eine entsprechende Fehlermeldung auf der Konsole ausgibt. In dieser Fehlermeldung soll der Name des Artikels erwähnt werden.

```
let handleQueryPrintErrs (op : Query -> Unit) (q : Query) : Unit =
 try op q with
 | NoSuchItem ->
     putline ("Der Artikel "^ q.item.name ^" existiert nicht")
 | InsufficientStock n ->
     putline ("Unzureicheinder Lagerbestand von "^ q.item.name)
```

2. Definieren Sie einen Ausnahmebehandler der die übergebene Anfrage ausführt, und, *nur* in dem Fall, dass der Restbestand zu gering ist, auf der Konsole fragt, ob die Anfrage erneut durchgeführt werden soll. Dabei sollen sowohl die ursprünglich angefragte Menge als auch der Restbestand auf der Konsole ausgegeben werden.

   Sie können die bereitgestellte Funktion `responseYes : Unit -> Bool` verwenden um eine Ja/Nein Antwort auf der Konsole einzuholen. Führen Sie im positiven Falle die Anfrage mit als angepasster Menge den Restbestand durch.

```
let handleQueryInsufficient (op : Query -> Unit) (q : Query) : Unit =
    try op q with
    | InsufficientStock n -> do
        putline ("Es gibt nur noch "^show n^" "^q.item.name^" auf Lager.
    Wollen Sie "^show n^" statt "^show q.quantity^" bestellen?")
        if responseYes () then op ({item = q.item; quantity = n})
```

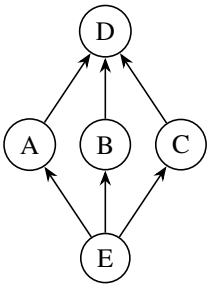
3. Kombinieren Sie die Ausnahmebehandler aus den zwei vorigen Teilaufgaben zu einem Ausnahmebehandler der die Ausnahmen `InsufficientStock` so wie `handleQueryInsufficient`, und `NoSuchItem` so wie `handleQueryPrintErrs` behandelt.

   Verwenden Sie in Ihrer Definition kein zusätzliches **`try ... with`**-Konstrukt!

```
let handleQueryCombined (op : Query -> Unit) (q : Query) : Unit =
    handleQueryPrintErrs (handleQueryInsufficient op) q
```

Aufgabe 9 Untertypen (__ /20 Punkte)

Es gelten die Untertypbeziehungen E $\preccurlyeq$ A, E $\preccurlyeq$ B, E $\preccurlyeq$ C, A $\preccurlyeq$ D, B $\preccurlyeq$ D und C $\preccurlyeq$ D, die in der nebenstehenden Abbildung visualisiert sind.



In der folgenden Tabelle werden je zwei Typen in Relation gesetzt:

- $t_1 \prec t_2$ bedeutet, dass t_1 ein Untertyp von t_2 ist ($t_1 \preccurlyeq t_2$), nicht jedoch t_2 ein Untertyp von t_1.
- $t_1 \succ t_2$ bedeutet, dass t_2 ein Untertyp von t_1 ist ($t_2 \preccurlyeq t_1$), nicht jedoch t_1 ein Untertyp von t_2.
- $t_1 \parallel t_2$ bedeutet, dass t_1 und t_2 unvergleichbar sind, das heißt es gilt weder $t_1 \preccurlyeq t_2$ noch $t_2 \preccurlyeq t_1$.

Füllen Sie die Lücken in der Tabelle aus. In der ersten und dritten Spalte müssen Sie **einen** Typ eintragen, in der zweiten Spalte eine Relation ($\prec$ oder $\succ$ oder $\parallel$). In Lücken, die Kästen ( ) enthalten, müssen Sie in jeden Kasten einen der Typen A, B, C, D oder E eintragen.

Für richtige Antworten (ganze Lücke) erhalten Sie zwei Punkte, für falsche Antworten werden zwei Punkte abgezogen. Nicht ausgefüllte Zeilen wirken sich nicht auf die Punktzahl aus. Diese Aufgabe wird mit mindestens 0 Punkten bewertet.

Zur Erinnerung: Die folgenden Deduktionsregeln gelten für die Relation »$\preccurlyeq$«:

$$\frac{t_1 \preccurlyeq t_2 \qquad t_2 \preccurlyeq t_3}{t_1 \preccurlyeq t_3} \qquad \frac{t_1 \preccurlyeq t_1' \qquad t_2 \preccurlyeq t_2'}{t_1 * t_2 \preccurlyeq t_1' * t_2'} \qquad \frac{t_1' \preccurlyeq t_1 \qquad t_2 \preccurlyeq t_2'}{t_1 \to t_2 \preccurlyeq t_1' \to t_2'}$$

t_1	Relation	t_2
D	$\boxed{\succ}$	E
A $*$ B	$\prec$	$\boxed{\text{D} * \text{B}}$, $\boxed{\text{A} * \text{D}}$, $\boxed{\text{D} * \text{D}}$
A $*$ C	$\boxed{\parallel}$	B $*$ B
C $\to$ D	$\prec$	$\boxed{\text{E} \to \text{D}}$
E $\to$ E	$\prec$	$\boxed{\text{E} \to \text{A}}$, $\boxed{\text{E} \to \text{B}}$, $\boxed{\text{E} \to \text{C}}$, $\boxed{\text{E} \to \text{D}}$
A $\to$ C	$\boxed{\parallel}$	C $\to$ A
E $\to$ D	$\boxed{\succ}$	D $\to$ E
B $\to$ (C $\to$ D)	$\prec$	$\boxed{\text{B} \to \text{E} \to \text{D}}$, $\boxed{\text{E} \to \text{C} \to \text{D}}$, $\boxed{\text{E} \to \text{E} \to \text{D}}$
B $\to$ C $\to$ D	$\parallel$	Es gibt **102 richtige Lösungen**, eine davon ist: $\boxed{\text{A} \to \text{A} \to \text{A}}$ **Falsch sind** die in der vorherigen Zeile genannten ($\prec$) sowie die folgenden ($\succeq$): $\boxed{\text{B} \to \text{C} \to \text{A}}$, $\boxed{\text{B} \to \text{C} \to \text{B}}$, $\boxed{\text{B} \to \text{C} \to \text{C}}$, $\boxed{\text{B} \to \text{C} \to \text{D}}$, $\boxed{\text{B} \to \text{C} \to \text{E}}$ $\boxed{\text{B} \to \text{D} \to \text{A}}$, $\boxed{\text{B} \to \text{D} \to \text{B}}$, $\boxed{\text{B} \to \text{D} \to \text{C}}$, $\boxed{\text{B} \to \text{D} \to \text{D}}$, $\boxed{\text{B} \to \text{D} \to \text{E}}$ $\boxed{\text{D} \to \text{C} \to \text{A}}$, $\boxed{\text{D} \to \text{C} \to \text{B}}$, $\boxed{\text{D} \to \text{C} \to \text{C}}$, $\boxed{\text{D} \to \text{C} \to \text{D}}$, $\boxed{\text{D} \to \text{C} \to \text{E}}$ $\boxed{\text{D} \to \text{D} \to \text{A}}$, $\boxed{\text{D} \to \text{D} \to \text{B}}$, $\boxed{\text{D} \to \text{D} \to \text{C}}$, $\boxed{\text{D} \to \text{D} \to \text{D}}$, $\boxed{\text{D} \to \text{D} \to \text{E}}$
(B $\to$ C) $\to$ D	$\prec$	$\boxed{(\text{B} \to \text{E}) \to \text{D}}$, $\boxed{(\text{D} \to \text{C}) \to \text{D}}$, $\boxed{(\text{D} \to \text{E}) \to \text{D}}$

Aufgabe 10 Unendliche Sequenzen (__ /20 Punkte)

In der Vorlesung haben Sie die `IEnumerator` Schnittstelle kennengelernt. Es wurde darauf hingewiesen, dass man damit auch unendliche Sequenzen repräsentieren kann. Wir führen hier eine Schnittstelle ein für Sequenzen die stets unendlich sind.

```
type IStream<'elem> =
  interface
    abstract member Current  : 'elem
    abstract member MoveNext : Unit -> Unit
  end
```

Zur Erinnerung: Vor dem initialen Zugriff auf `Current` muss zwingend ein Aufruf von `MoveNext ()` erfolgt sein. Nehmen Sie an, dass die übergebene Sequenz sich an die Konvention hält, dass wenn dies nicht der Fall ist, eine Ausnahme geworfen wird, und stellen Sie sicher, dass diese Konvention in den von Ihnen definierten Sequenzen erhalten bleibt.

a) Definieren Sie eine Funktion `map<'a,'b> : (f : 'a -> 'b) -> IStream<'a> -> IStream<'b>`, die die übergebene Funktion `f` auf die Elemente der Eingabesequenz anwendet.

__ /10

```
let map<'a,'b> (f : 'a -> 'b) (i : IStream<'a>) : IStream<'b> =
        { new IStream<'b> with
          member _.Current = f i.Current
          member _.MoveNext () = i.MoveNext ()
        }
```

b) Definieren Sie eine Funktion `atSquareIndices<'a> : IStream<'a> -> IStream<'a>`, die die Teilsequenz von Elementen der ursprünglichen Sequenz zurückgibt, deren Indizes Quadratzahlen sind. Mathematisch ausgedrückt gilt:

$$\texttt{atSquareIndices}((a_i)_{i\in\mathbb{N}}) = (a_{j^2})_{j\in\mathbb{N}}.$$

Hinweis: Wir erinnern, dass die Folge der Differenzen von aufeinanderfolgenden Quadratzahlen die ungeraden Zahlen sind.

$$\begin{array}{ccccccc|c} 0 & & 1 & & 4 & & 9 & \ldots & n^2 \\ & 1 & & 3 & & 5 & & \ldots & 2n+1 \\ & & 2 & & 2 & & \ldots & \ldots & 2 \end{array}$$

__ /10

```
let atSquareIndices<'a> (i : IStream<'a>) : IStream<'a> =
        let mutable toSkip = 1N
        let mutable started = false
        { new IStream<'a> with
          member _.Current = i.Current
          member _.MoveNext () = do
            if started then
              for _ in 1N..toSkip do i.MoveNext ()
              toSkip <- toSkip + 2N;
            else
              i.MoveNext ()
              started <- true
        }
```