

## Klausur Programmierparadigmen

SS20, 06. Oktober 2020, 09:00 – 11:00 Uhr

**Zugelassene Hilfsmittel:** Papierbasierte Quellen (Vorlesungsfolien, Übungsblätter, eigene Aufzeichnungen, Bücher, ...)

Die Verwendung von elektronischen Geräten ist verboten.

**Bearbeitungszeit:** 120 min

Aufgabe	max. Punkte	err. Punkte
1	12	
2	18	
3	18	
4	12	
5	12	
6	12	
7	5	
8	8	
9	5	
10	18	
$\Sigma$	<b>120</b>	
$[\Sigma]$	<b>120</b>	

Jeder Punkt entspricht ca. 1 min Bearbeitungszeit. Es ist garantiert, dass die Klausur mit 60 Punkten bestanden ist, und dass 115 Punkte für die Note 1,0 ausreichen.

Name: \_\_\_\_\_

Matrikelnummer: \_\_\_\_\_

Studiengang: \_\_\_\_\_

Schreiben Sie Ihre Lösungen direkt in die Klausur. Beschriften Sie **alle verwendeten Blätter** mit Ihrem Namen und Ihrer Matrikelnummer. Trennen Sie die geklammerten Blätter **nicht** auf. Weitere Blätter erhalten Sie bei Bedarf.

---

**Aufgabe 1** (Haskell: Socken sortieren)

[12 Punkte]

Angenommen, Sie haben gerade Ihre Socken gewaschen, die nun unsortiert vor Ihnen liegen. Um sie wieder zu Paaren zusammenzustellen, verwenden Sie folgenden Algorithmus:

1. Räume eine Ablagefläche frei
2. Für jede Socke:
  - Falls eine gleiche Socke auf der Ablagefläche liegt, bilde mit dieser ein Sockenpaar und räume es weg
  - Andernfalls lege die Socke auf die Ablagefläche

Wie groß sollte die Ablagefläche sein? In dieser Aufgabe sollen Sie das mit einer Simulation in Haskell abschätzen. Socken sind vom Typ `Sock`, der die Typklasse `Eq` implementiert.

(a) Implementieren Sie eine Funktion

[7 Punkte]

```
maxStorage :: [Sock] -> Int
```

Der Aufruf `maxStorage socks` soll den obenstehenden Algorithmus simulieren, wenn die Liste `socks` von vorne nach hinten abgearbeitet wird. Das Ergebnis soll die maximale Anzahl Socken sein, die während der Ausführung auf der Ablagefläche liegen.

Sie dürfen hierbei zusätzlich die Funktion `delete :: (Eq a) => a -> [a] -> [a]` benutzen: Der Aufruf `delete x list` gibt die Liste `list` ohne das erste Vorkommen von `x` zurück.

**Hinweis:** Neben `delete` könnte auch `elem` nützlich sein.

Nun geht es darum, eine zufällige Eingabeliste zu erzeugen.

(b) Implementieren Sie eine Funktion

[2 Punkte]

```
swap :: Int -> Int -> [a] -> [a]
```

Der Aufruf `swap i j list` soll `list` zurückgeben, wobei die Elemente an den Indizes `i` und `j` vertauscht sind.

Sie dürfen hierbei zusätzlich die Funktion `setAt :: Int -> a -> [a] -> [a]` benutzen: Der Aufruf `setAt i x list` gibt `list` zurück, wobei das `i`-te Element auf `x` gesetzt ist.

(c) Implementieren Sie eine Funktion

[3 Punkte]

```
shuffle :: [(Int, Int)] -> [a] -> [a]
```

Das erste Argument von `shuffle` ist eine Liste von Paaren von Indizes, wobei jedes Paar eine Vertauschung codiert. `shuffle` soll auf seinem zweiten Argument diese Vertauschungen nacheinander ausführen und das Ergebnis zurückgeben.

Name:

Matrikelnummer:

---

## Aufgabe 2 (Haskell: Pattern Matching)

[18 Punkte]

In dieser Aufgabe sollen Sie Pattern Matching für binäre Bäume implementieren. Sowohl Patterns als auch die gematchten Bäume haben eine eigene Haskell-Datenstruktur:

```
data Tree                data Pattern
  = Leaf                  = PLeaf
  | Node Tree Tree        | PNode Pattern Pattern
                           | PVar String — Variablenname als String
```

Dabei kommt keine Variable in einem Pattern mehrfach vor.

Durch das Pattern Matching ergibt sich eine Variablenbelegung in Form einer Assoziativliste:

```
type Subst = [(String, Tree)]
```

Beispielsweise bedeutet `[("a", Leaf), ("b", Node Leaf Leaf)]`, dass Variable "a" mit Leaf belegt ist und "b" mit Node Leaf Leaf.

(a) Implementieren Sie die Funktion

[4 Punkte]

```
subst :: Subst -> Pattern -> Tree
```

`subst sub pat` substituiert alle Variablenvorkommen in `pat` durch ihre Belegung in `sub`. Die restliche Baumstruktur bleibt erhalten. Sie dürfen annehmen, dass alle in `pat` vorkommenden Variablen eine Belegung in `sub` haben.

**Hinweis:** `lookup :: Eq a => a -> [(a, b)] -> Maybe b` aus der Prelude könnte im Umgang mit `sub` nützlich sein.

(b) Implementieren Sie die Funktion

[2 Punkte]

```
liftMaybe2 :: (a -> b -> c) -> Maybe a -> Maybe b -> Maybe c
```

`liftMaybe2 f ma mb` soll `Just c` mit geeignetem `c` zurückgeben, falls `ma` und `mb` auf `Just _` matchen, ansonsten `Nothing`.

(c) Implementieren Sie die Funktion

[6 Punkte]

```
match :: Pattern -> Tree -> Maybe Subst
```

Der Aufruf `match pat tree` matcht `tree` gegen das Muster `pat`. Ist der Match erfolgreich, wird `Just sub` zurückgegeben, wobei `sub` die entstehende Variablenbelegung ist. Ansonsten ist das Ergebnis `Nothing`.

`PLeaf` matcht nur für `Leaf`. `PNode p1 p2` matcht nur für `Node n1 n2`, in welchem Fall `n1` auf `p1` und `n2` auf `p2` matchen muss. Die beiden Variablenbelegungen werden konkateniert. Jeder `tree` matcht auf `PVar v`, in welchem Fall `v` in `sub` mit `tree` belegt ist.

**Hinweis:** `liftMaybe2` könnte nützlich sein.

**Beispiele:**

```
> match PLeaf Leaf
Just []
> match (PNode (PVar "a") PLeaf) (Node Leaf Leaf)
Just [("a", Leaf)]
> match PLeaf (Node Leaf Leaf)
Nothing
```

(d) Implementieren Sie die Funktion

[6 Punkte]

```
matchFirst :: [(Pattern, Subst -> a)] -> Tree -> a
```

`matchFirst alts tree` sucht die erste Alternative `(pat, f)` aus `alts`, deren Pattern `pat` `tree` matcht. Für die Rückgabe wird dann `f` auf die entstandene Variablenbelegung angewendet. Sie können davon ausgehen, dass immer eine passende Alternative existiert.

**Beispiele:**

```
> alts = [(PLeaf, f), (PNode (PVar "a") PLeaf, g), (PVar "b", h)]
> matchFirst alts Leaf
f []
> matchFirst alts (Node Leaf Leaf)
g [("a", Leaf)]
```

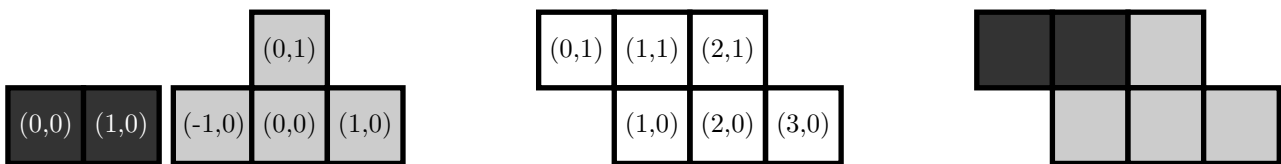
### Aufgabe 3 (Prolog: Legepuzzle)

[18 Punkte]

Bei einem Legepuzzle muss ein Feld mit einem Vorrat von Steinen bedeckt werden. Das Feld und jeder Stein bestehen jeweils aus einzelnen Quadraten. Es gelten folgende Bedingungen:

- Alle Steine müssen verwendet werden.
- Das Feld muss vollständig bedeckt sein.
- Zwei Steine dürfen sich nicht überlappen.
- Ein Stein darf nicht über das Feld hinausragen.
- Ein Stein darf nicht gedreht oder gespiegelt werden.

Beispiel aus einem Steinvorrat mit zwei Steinen, einem Feld und einer gültigen Platzierung:



Dabei codieren wir

- einen Stein als Liste von  $(X, Y)$ -Koordinaten der belegten Quadrate, wobei jeder Stein die Koordinate  $(0, 0)$  enthält. Der linke Stein wird z.B. als  $[(0, 0), (1, 0)]$  codiert.
- den Steinvorrat als Liste von Steinen, also z.B. den Steinvorrat aus den beiden obigen Steinen als

```
[
  [(0, 0), (1, 0)],
  [(-1, 0), (0, 0), (1, 0), (0, 1)]
].
```

- das Feld als Liste von  $(X, Y)$ -Koordinaten der freien Quadrate, also z.B. das obige Feld als  $[(0, 1), (1, 1), (2, 1), (1, 0), (2, 0), (3, 0)]$ .
- die Lösung als Liste von Stein-Koordinaten-Paaren, die für jeden Stein angibt, auf welches Quadrat des Feldes das  $(0, 0)$ -Quadrat gelegt wird. Die Lösung oben wird also codiert als

```
[
  ([ (0, 0), (1, 0) ], (0, 1)),
  ([ (-1, 0), (0, 0), (1, 0), (0, 1) ], (2, 0))
].
```

- (a) Definieren Sie ein vierstelliges Prädikat **move(Stein1, X, Y, Stein2)**, das [6 Punkte]  
genau dann erfüllbar ist, wenn Stein2 aus Stein1 durch Verschiebung um X in x-Richtung und Y in y-Richtung entsteht. Dadurch wird bei einem Stein aus dem Vorrat das Quadrat  $(0, 0)$  auf das Quadrat  $(X, Y)$  geschoben. Dabei sind Stein1, X und Y stets instanziiert.
- (b) Definieren Sie ein dreistelliges Prädikat **remove\_all(Feld1, Stein, Feld2)**, das genau dann erfüllbar ist, wenn Feld2 aus Feld1 entsteht, indem daraus alle Quadrate des Steins Stein entfernt werden. Ist ein Quadrat aus Stein nicht in Feld1 vorhanden, soll das Prädikat nicht erfüllbar sein. [4 Punkte]  
**Hinweis:** Das Prädikat `delete(Xs, Y, Zs)` aus der Vorlesung könnte nützlich sein.

- (c) Definieren Sie ein fünfstelliges Prädikat **place(Feld1, Stein, X, Y, Feld2)**, das genau dann erfüllbar ist, wenn Feld2 aus Feld1 entsteht, indem Stein um X in x-Richtung und Y in y-Richtung verschoben auf das Feld gelegt wird. Danach sind also die Quadrate aus Feld1, auf die der Stein gelegt wurde, nicht mehr frei. Sie können annehmen, dass Feld1, Stein1, X und Y stets instanziiert sind. [2 Punkte]
- (d) Definieren Sie ein dreistelliges Prädikat **solve(Feld, Steine, Platzierungen)**, das genau dann erfüllbar ist, wenn Platzierungen eine gültige Lösung für das Legepuzzle mit Feld Feld und Steinvorrat Steine ist. [6 Punkte]
- Hinweis:** Versuchen Sie für einen Stein, die  $(0, 0)$ -Koordinate auf ein noch freies Quadrat im Feld Feld zu setzen.

---

**Aufgabe 4** ( $\lambda$ -Kalkül: Auswertungsreihenfolgen)

[12 Punkte]

In dieser Aufgabe kürzen wir Call-by-Value als CBV und Call-by-Name als CBN ab.

Wir betrachten die folgende Familie von  $\lambda$ -Kalkül-Termen:

$$\begin{aligned} T_0 &= \lambda x. x \\ T_n &= (\lambda a. a \ a) \ T_{n-1} \quad \text{für } n \in \mathbb{N}, n > 0 \end{aligned}$$

**Hinweis:** Beachten Sie die korrekte Klammerung beim Einsetzen der Terme.

- (a) Geben Sie  $T_3$  vollständig eingesetzt an. Markieren Sie in  $T_3$  den Redex, der unter CBV zuerst reduziert wird, und den Redex, der unter CBN zuerst reduziert wird. [2 Punkte]

Nun seien  $V(n)$  und  $N(n)$  die Anzahl der  $\beta$ -Reduktionen, die CBV bzw. CBN benötigen, um  $T_n$  in seine Normalform zu überführen. Mit diesen wollen wir die Effizienz von CBV und CBN auf  $T_n$  untersuchen.

Für alle  $n \in \mathbb{N}$  gilt:  $T_n \Rightarrow^* \lambda x. x$ . Dies dürfen Sie ohne Beweis benutzen.

- (b) Stellen Sie für allgemeines  $n > 0$  dar, wie  $T_n$  unter CBV und CBN zu seiner Normalform reduziert wird. Fassen Sie die Reduktionen von  $T_{n-1}$  zu dessen Normalform in einem Schritt zusammen. [5 Punkte]  
Geben Sie bei jedem Schritt an, wie viele  $\beta$ -Reduktionen in ihm passieren. Um die Anzahl auszudrücken, dürfen Sie die Funktionen  $V$  und  $N$  verwenden.
- (c) Stellen Sie für  $V(n)$  und  $N(n)$  rekursive Definitionen (Rekurrenzen) auf und geben Sie ohne Begründung das asymptotische Verhalten von  $V(n)$  und  $N(n)$  an. [5 Punkte]  
**Hinweis:** Wenn Sie sich einige Werte von  $V(n)$  und  $N(n)$  tabellieren, können Sie deren Verhalten leicht erkennen.



Name:

Matrikelnummer:

---

# Aufgabe 5 (Typinferenz, Unifikation)

[12 Punkte]

Gegeben ist der folgende Ausschnitt aus einem Herleitungsbaum, wobei  $t$  für einen nicht weiter bestimmten Term steht:

$$\begin{array}{c}
 \vdots \\
 \hline
 \text{Let} \frac{x : \alpha_2, y : \alpha_4 \vdash \lambda f. \lambda z. f \times (f \ y \ z) : \beta_1}{x : \alpha_2, y : \alpha_4 \vdash \mathbf{let} \ g = \lambda f. \lambda z. f \times (f \ y \ z) \mathbf{in} \ t : \alpha_5}
 \end{array}
 \quad
 \begin{array}{c}
 \text{Var} \frac{\textcircled{1}}{\Gamma \vdash g : \alpha_7} \quad \text{Var} \frac{\textcircled{2}}{\Gamma \vdash g : \alpha_8} \\
 \vdots \quad \dots \quad \vdots \\
 \hline
 \Gamma \vdash t : \alpha_6
 \end{array}$$

Aus dem linken (nicht dargestellten) Teilbaum ergibt sich der allgemeinste Unifikator

$$\begin{aligned}
 \sigma_{let} = [ & \\
 & \beta_1 \dot{\vdash} (\beta_8 \rightarrow \beta_4 \rightarrow \beta_4) \rightarrow \beta_4 \rightarrow \beta_4, \\
 & \beta_{10} \dot{\vdash} \beta_8, \\
 & \beta_5 \dot{\vdash} \beta_4, \\
 & \beta_7 \dot{\vdash} \beta_4, \\
 & \beta_3 \dot{\vdash} \beta_4 \rightarrow \beta_4, \\
 & \beta_6 \dot{\vdash} \beta_4 \rightarrow \beta_4, \\
 & \beta_9 \dot{\vdash} \beta_4 \rightarrow \beta_4, \\
 & \beta_2 \dot{\vdash} \beta_8 \rightarrow \beta_4 \rightarrow \beta_4, \\
 & \alpha_2 \dot{\vdash} \beta_8, \\
 & \alpha_4 \dot{\vdash} \beta_8 \\
 & ].
 \end{aligned}$$

(a) Geben Sie  $\Gamma$  an. [3 Punkte]

(b) Ergänzen Sie, was an den mit ① und ② markierten Stellen einzutragen ist. [3 Punkte]

$$\begin{aligned}
 \textcircled{1} \quad \Gamma(g) &= \dots, \\
 \Gamma(g) &\succeq \text{ }
 \end{aligned}$$

$$\begin{aligned}
 \textcircled{2} \quad \Gamma(g) &= \dots, \\
 \Gamma(g) &\succeq \text{ }
 \end{aligned}$$

(c) Geben Sie einen allgemeinsten Unifikator für die folgende Constantmenge in Listenform  $[\dots, \alpha_i \dot{\vdash} \tau_i, \dots]$  an. [6 Punkte]

$$\begin{aligned}
 C = \{ & \\
 & \alpha_5 = \alpha_4 \rightarrow \alpha_6, \\
 & \alpha_1 = \alpha_2 \rightarrow \alpha_3 \rightarrow \alpha_4, \\
 & \alpha_1 = \alpha_5 \rightarrow \alpha_6, \\
 & \}
 \end{aligned}$$

Name:

Matrikelnummer:

---

## Aufgabe 6 (MPI)

[12 Punkte]

Gegeben ist die untenstehende Methode `blockwisePartialSum`. Das im Parameter `elements` übergebene Array beinhaltet eine Sequenz an Integer-Ganzzahlen und der Parameter `elementCount` beinhaltet die Anzahl an Elementen in `elements`.

Gehen Sie bei einer Ausführung der Methode davon aus, dass die Anzahl an Elementen in `elementCount` ein Vielfaches der gestarteten Prozesse ist. Gehen Sie weiterhin davon aus, dass eine korrekte Initialisierung und Finalisierung von MPI vom Aufrufenden durchgeführt wird.

```
1 void blockwisePartialSum(int elements[], int elementCount) {
2
3     int size, rank;
4     MPI_Comm_size(MPI_COMM_WORLD, &size);
5     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6     MPI_Request request;
7     MPI_Status status;
8
9     int elementsPerProcess = elementCount / size;
10
11     int localElements[elementsPerProcess];
12     MPI_Scatter(elements, elementsPerProcess, MPI_INT, localElements,
13                 elementsPerProcess, MPI_INT, 0, MPI_COMM_WORLD);
14
15     int received[(rank + 1) * elementsPerProcess];
16
17     for (int i = rank; i < size; i++){
18         MPI_Isend(localElements, elementsPerProcess, MPI_INT, i, 0,
19                   MPI_COMM_WORLD, &request);
20     }
21
22     for (int i = 0; i <= rank; i++){
23         MPI_Recv(received + (i * elementsPerProcess), elementsPerProcess,
24                 MPI_INT, i, 0, MPI_COMM_WORLD, &status);
25     }
26
27     int result = 0;
28
29     for (int i = 0; i < (rank + 1) * elementsPerProcess; i++){
30         result += received[i];
31     }
32
33     int collected[size];
34     MPI_Allgather(&result, 1, MPI_INT, collected, 1, MPI_INT,
35                  MPI_COMM_WORLD);
36 }
```

- (a) Im Folgenden finden Sie mehrere Tabellen. Jede Tabelle stellt den Inhalt eines Buffers auf allen gestarteten Prozessen *nach* Ausführung einer definierten Code-Zeile dar. Hierbei befindet sich in der ersten Spalte jeder Tabelle der `rank` der Prozesse und in der zweiten Spalte der Inhalt des Buffers auf dem zum `rank` zugehörigen Prozess. Der umschließende Kasten um eine Tabelle gibt sowohl den Namen des mit der Tabelle korrespondierenden Buffers sowie die Code-Zeile an, nach deren Ausführung der Inhalt dargestellt werden soll. Ergänzen Sie den Inhalt jeder Tabelle auf Basis der Annahme, dass das Array `elements` die Ganzzahl-Sequenz  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$  beinhalten und dass `elementCount` den Wert 9 beinhaltet. Gehen Sie davon aus, dass das Programm mit drei Prozessen gestartet wird. [5 Punkte]

**localElements** (Nach Zeile 12)

Prozess Nr.	Inhalt
0	
1	
2	

**received** (Nach Zeile 22)

Prozess Nr.	Inhalt
0	
1	
2	

**result** (Nach Zeile 28)

Prozess Nr.	Inhalt
0	
1	
2	

**collected** (Nach Zeile 31)

Prozess Nr.	Inhalt
0	
1	
2	

- 
- (b) Nehmen Sie an, dass der Befehl `MPI_Recv` in Zeile 21 durch `MPI_Irecv` ersetzt wird. [3 Punkte]
- i. Erklären Sie allgemein, wie sich `MPI_Recv` und `MPI_Irecv` in ihrem Verhalten unterscheiden.
  - ii. Wird beim Ersetzen von `MPI_Recv` in Zeile 21 durch `MPI_Irecv` das Verhalten des gesamten Programms beibehalten? Begründen Sie Ihre Antwort.
- (c) Die Implementierung soll nun so geändert werden, dass die Größe des Eingabe-Arrays `elements` beliebig sein kann. Für diese Änderung könnte unter anderem anstatt `MPI_Scatter` in Zeile 12 die Vektor-Variante `MPI_Scatterv` verwendet werden. Erklären Sie die Bedeutung der Parameter `int* sendcounts` und `int* displacements` von `MPI_Scatterv`. Erklären Sie hierfür wie die Inhalte dieser Arrays an einem bestimmten Index  $i$  von `MPI_Scatterv` verwendet wird. [2 Punkte]

- (d) Im Folgenden ist eine unvollständige Implementierung als Ersatz für die Zeilen 11 und 12 der Methode `blockwisePartialSum` gegeben, welche das Verhalten von `MPI_Scatterv` realisieren soll.

Ergänzen Sie im folgenden Quelltext den fehlenden Inhalt.

**Hinweis:** Beachten Sie die Kommentare in der folgendend unvollständigen Implementierung.

```
11 //Sie können die folgenden Variablen "sendcounts" and "displacements"
    in der Implementierung verwenden und annehmen, dass diese valide
    Eingaben für die korrespondierenden Parameter von MPI_Scatterv
    enthalten.
12 //int* sendcounts = ...;
13 //int* displacements = ...;
14
15 localElements[ ];
16
17 if(rank == 0){
18     for(int i = 0; i < size; i++){
19         MPI_Isend( ,
                    , MPI_INT, i, 0,
                    MPI_COMM_WORLD, &request);
20     }
21 }
22
23 MPI_Recv(localElements, , MPI_INT, 0,
          0, MPI_COMM_WORLD, &status);
```

## Aufgabe 7 (Java Happens-before)

[5 Punkte]

Gegeben ist untenstehende Klasse `ProducerConsumer` mit einer `main`-Methode. Hierbei erzeugt der Thread `producer` eine zufällige Ganzzahl zwischen 0 und 10 und speichert diese in der Variable `number`. Der Thread `consumer` konsumiert die von `producer` erzeugte Zahl. Auf Basis des gegebenen Programmcodes kann folgende beispielhafte Ausgabe erwartet werden:

```
Produced 3
Producer terminating
Consuming 3
Consumer terminating
```

Bei Ausführung des Programms bemerken Sie ein fehlerhaftes Verhalten, bei dem der Thread `consumer` (und somit das Programm) *nicht* terminiert und nur folgende beispielhafte Ausgabe zu sehen ist:

```
Produced 3
Producer terminating
```

```
1 public class ProducerConsumer {
2
3     public static Boolean numberAvailable = false;
4     public static int number = -1;
5
6     public static void main(String[] args){
7
8         Thread producer = new Thread(()-> {
9
10             while (numberAvailable) {/* busy waiting*/}
11
12             number = new Random().nextInt(10);
13             System.out.println("Produced " + number);
14             numberAvailable = true;
15
16             System.out.println("Producer terminating");
17         });
18
19         Thread consumer = new Thread(()-> {
20
21             while (!numberAvailable) {/* busy waiting*/}
22
23             System.out.println("Consuming " + number);
24             number = -1;
25             numberAvailable = false;
26
27             System.out.println("Consumer terminating");
28         });
29
30         producer.start();
31         consumer.start();
32     }
33 }
```



- (a) Erklären Sie den Begriff der *Happens-before-Beziehung*. Nennen Sie außerdem [2 Punkte]  
einen Ausdruck oder ein Keyword aus der Vorlesung, welches eine *Happens-before-Beziehung* herstellt.
- (b) In der gegebenen Implementierung kann es zu Speicherinkonsistenzen kommen, wo- [3 Punkte]  
durch das beschriebene fehlerhafte Verhalten auftritt. Beschreiben Sie, wie Sie die  
gegebene Implementierung verändern müssen, damit das Problem der Speicherin-  
konsistenzen nicht mehr auftritt und sich hierdurch das Programm korrekt verhält.  
Begründen Sie unter Verwendung der *Happens-before-Beziehung* und mit Bezug auf  
die gegebene Implementierung, warum Ihre Änderung das Problem behebt.

## Aufgabe 8 (Design-By-Contract)

[8 Punkte]

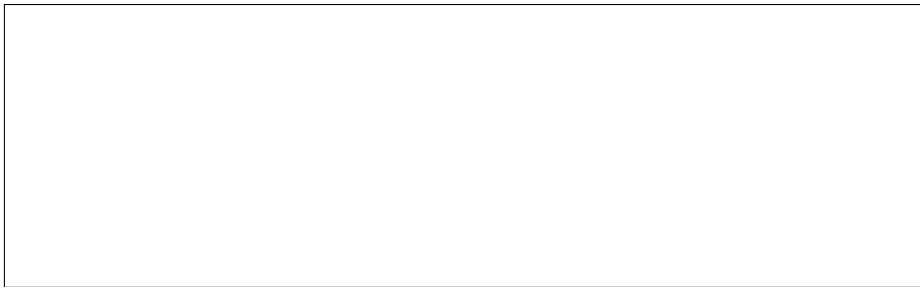
Gegeben sei das Fragment der Klasse `Bicycle`. Diese Klasse stellt ein Fahrrad mit zwei Schalthebeln und einem daraus resultierenden Gang dar. In `firstShift` steht der Wert des ersten Schalthebels und in `secondShift` der Wert des zweiten Schalthebels. Für den Wert von `secondShift` gilt `secondShift < maxSecondShift`. Der aktuelle Gang-Wert in `currentGear` wird durch die Formel `firstShift * maxSecondShift + secondShift` berechnet und muss stets kleiner als `maxGear` sein (`currentGear < maxGear`).

Gegeben ist die Methode `shiftUp(int increase)` mit Vor- und Nachbedingungen. Diese Methode berechnet für einen zu erhöhenden Wert `increase` des aktuellen Ganges die neuen Werte von `firstShift`, `secondShift` und `currentGear`. Zusätzlich ist eine Methode `main` gegeben, welche eine Folge von Aufrufen von `shiftUp` auf einem `Bicycle`-Objekt ausführt.

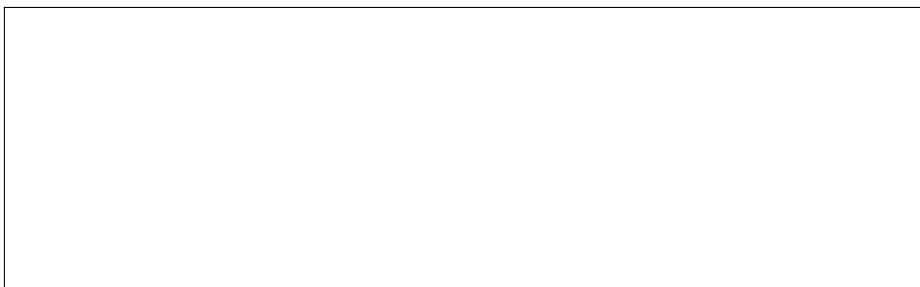
```
1 public class Bicycle {
2     private int firstShift = 0;
3     private int secondShift = 0;
4     private int currentGear = 0;
5     private final int maxGear = 21;
6     private final int maxSecondShift = 7;
7
8     /*
9     @ requires increase > 0;
10    @ requires currentGear + increase < maxGear;
11    @ ensures secondShift < maxSecondShift;
12    @ ensures increase >= maxSecondShift ==>
13        firstShift == \old(firstShift) + 1;
14    */
15    public void shiftUp(int increase) {
16        currentGear += increase;
17        firstShift = currentGear / maxSecondShift;
18        secondShift = currentGear % maxSecondShift;
19    }
20
21    public static void main(String[] args) {
22        Bicycle bicycle = new Bicycle();
23        bicycle.shiftUp(3);
24        bicycle.shiftUp(10);
25        bicycle.shiftUp(9);
26    }
27 }
```

- (a) Wird der Vertrag von `shiftUp` vom *Aufrufenden* erfüllt? Begründen Sie ihre Antwort. [2.5 Punkte]
- (b) Wird der Vertrag von `shiftUp` vom *Aufgerufenen* erfüllt? Begründen Sie Ihre Antwort. [2.5 Punkte]
- (c) Ergänzen Sie folgende alternative Implementierung der oben gegebenen `shiftUp`-Methode mit Assertions (`assert expr;`), so dass ihre Vor- und Nachbedingungen sichergestellt werden. Sie dürfen hierfür weitere Variablen und Kontrollstrukturen einfügen. [3 Punkte]

```
public void shiftUp(int increase) {
```



```
currentGear += increase;  
firstShift = increase / maxSecondShift;  
secondShift = increase % maxSecondShift;
```



```
}
```

**Aufgabe 9** (Grammar Engineering)

[5 Punkte]

Gegeben sei die folgende Grammatik mit Startsymbol  $T$  und den Terminalen **a**, **b**, **c**, **d**, **e** und **f**.

$$T \rightarrow \mathbf{a} \mid \mathbf{a\,b} \mid \mathbf{c\,d\,f} \mid \mathbf{c\,e\,f}$$

- (a) Begründen Sie kurz, warum diese Grammatik nicht SLL(1) ist.

[2 Punkte]

- (b) Führen Sie eine Linksfaktorisierung der Grammatik durch.

[3 Punkte]

**Aufgabe 10** (Syntaktische Analyse)

[18 Punkte]

Gegeben sei im Folgenden eine Grammatik für Java-Generics mit Startsymbol *Generics*.

$$\begin{aligned} \textit{Generics} &\rightarrow \textbf{< GList >} & \textit{GAtom} &\rightarrow \textbf{id IdRest} \mid \textbf{?} \\ \textit{GList} &\rightarrow \textit{GAtom GRest} & \textit{IdRest} &\rightarrow \varepsilon \mid \textit{Generics} \\ \textit{GRest} &\rightarrow \varepsilon \mid \textbf{, GList} \end{aligned}$$

- (a) Tragen Sie die  $\text{Follow}_1$ -Menge aller Nichtterminale der Grammatik in die folgende Tabelle ein.

[5 Punkte]

Nichtterminal	$\text{Follow}_1$ -Menge
<i>Generics</i>	
<i>GList</i>	
<i>GRest</i>	
<i>GAtom</i>	
<i>IdRest</i>	

- (b) Begründen Sie, warum die Berechnung der Indizmengen für die Produktionen der Nichtterminale *Generics* und *GList* für die Überprüfung SLL(1)-Bedingung nicht notwendig ist.

[2 Punkte]

- (c) Zeigen Sie, dass die Grammatik in SLL(1) ist. Tragen Sie die hierzu notwendigen Indizmengen in die folgende Tabelle ein.

[3 Punkte]

Nichtterminal	Indizmengen (in der Reihenfolge der Produktionen)
<i>GRest</i>	
<i>GAtom</i>	
<i>IdRest</i>	

- (d) Vervollständigen Sie den unten gegebenen SLL(1)-Parser für die Grammatik, indem Sie die Methoden für die Nichtterminale *Generics*, *GAtom* und *IdRest* implementieren. Brechen Sie bei Parserfehlern mit der globalen `error()`-Funktion ab. [8 Punkte]

**Hinweis:** Sie können davon ausgehen, dass die Lexer-Schnittstelle wie folgt aussieht:

- Die Token sind Werte eines enum namens **Token** und heißen **LT**, **GT**, **COMMA**, **QUESTION**, **IDENT** und **EOF** für `<`, `>`, `,`, `?`, Identifier sowie das Ende der Eingabe. Token können bspw. mit `==` verglichen werden.
- `lexer.current` gibt nichtkonsumierenden Zugriff auf das aktuelle Token.
- `lexer.lex()` konsumiert das aktuelle Token.

```
1 void parseGList() {...} void parseGRest() {...}
2
3 void expect(Token tok) {
4     if (lexer.current != tok) { error(); }
5     lexer.lex();
6 }
```



Name:

Matrikelnummer:

---

