Prof. Christian Bischof, Ph.D.
Jonathan Heinen, Michael Lülfesmann, Carsten Otto

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte ankreuzen):

- ○ Informatik Bachelor
- ○ Mathematik Bachelor
- ○ Informatik Lehramt
- ○ Sonstige: _____

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**10**	
Aufgabe 2	**14**	
Aufgabe 3	**16**	
Aufgabe 4	**26**	
Aufgabe 5	**15**	
Aufgabe 6	**19**	
Summe	**100**	

Hinweise:

- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit dokumentenechten Stiften, nicht mit roten Stiften oder mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern (benutzen Sie auch die Rückseiten).
- Zusätzlich bereitgestellte Blätter werden **sofort** an die Aufgabenblätter **geheftet**.
- **Auf alle Blätter** (inklusive zusätzliche Blätter) müssen Sie **Ihren Namen und Ihre Matrikelnummer** schreiben.
- Was nicht bewertet werden soll, streichen Sie bitte durch.
- Werden **Täuschungsversuche** beobachtet, so wird die Klausur mit **0 Punkten** bewertet.
- Geben Sie am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.

Matrikelnummer: **Name:**

Aufgabe 1 (Programmanalyse): (10 Punkte)

Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Schreiben Sie hierzu jeweils die ausgegebenen Zeichen in die Kästchen hinter den Kommentar „`OUT:`“.

```
public class A {
  public int x = 1;
  public static int y = 3;
  public A() {
    this(5, 7.0);
  }
  public A(int y, double z) {
    A.y++;
    x = this.y + (int) z;
  }
  public void f(int y, double z) {
    this.x += x + y;
  }
}

public class B extends A {
  public int z = 6;
  public B() {
    this.z = 1;
  }
  public void f(int x, int y) {
    x = this.x;
  }
  public void f(int x, double y) {
    this.x += x;
    this.z++;
  }
}

public class M {
  public static void main(String[] args) {
    A a = new A();
    System.out.println(a.x + " " + a.y);              // OUT: [   ] [   ]

    B b = new B();
    System.out.println(b.x + " " + b.y + " " + b.z); // OUT: [   ] [   ] [   ]

    b.f(10, 2.0);
    System.out.println(b.x + " " + b.z);              // OUT: [   ] [   ]

    a = b;
    a.f(10, 2);
    System.out.println(a.x + " " + b.x + " " + b.z); // OUT: [   ] [   ] [   ]
  }
}
```

Matrikelnummer: **Name:**

Aufgabe 2 (Verifikation): **(12 + 2 Punkte)**

Gegeben sei folgender *Java*-Algorithmus P zur Berechnung von $\sum_{i=1}^{n} 7^i$.

$\langle\varphi\rangle$ (Vorbedingung)

```
res = 0;
sp = 1;
k = 0;
while (k < n) {
    sp = sp * 7;
    res = res + sp;
    k = k + 1;
}
```

$\langle\psi\rangle$ (Nachbedingung)

a) Als Vorbedingung für den oben aufgeführten Algorithmus P gelte $n \geq 0$ und als Nachbedingung:

$$res = \sum_{i=1}^{n} 7^i \wedge sp = 7^n$$

Vervollständigen Sie die Verifikation des folgenden Algorithmus im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Hinweise:

- Eine leere Summe hat den Wert 0, beispielsweise gilt $\sum_{i=a}^{b} i := 0$ für $a > b$.
- Zur Erinnerung: $7^0 = 1$
- Auf der nächsten Seite finden Sie eine Vorlage, die Sie direkt ausfüllen dürfen.

Matrikelnummer: **Name:**

$\langle n \geq 0 \rangle$

$\langle \_\_\_\_\_ \rangle$

```
res = 0;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
sp = 1;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
k = 0;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
while (k < n) {
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    sp = sp * 7;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    res = res + sp;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    k = k + 1;
```

$\langle \_\_\_\_\_ \rangle$

```
}
```

$\langle \_\_\_\_\_ \rangle$

$\langle res = \sum_{i=1}^{n} 7^i \wedge sp = 7^n \rangle$

b) Beweisen Sie die Terminierung des Algorithmus P. Geben Sie hierzu eine Variante für die `while`-Schleife an. Zeigen Sie, dass es sich tatsächlich um eine Variante handelt, und beweisen Sie damit unter Verwendung des Hoare-Kalküls mit der Voraussetzung $n \geq 0$ die Terminierung.

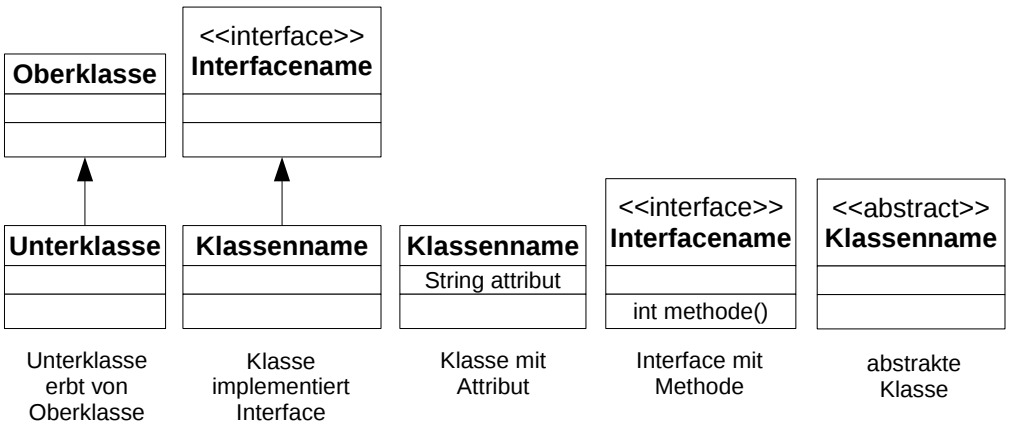
Matrikelnummer: **Name:**

Aufgabe 3 (Datenstrukturen in Java): **(6 + 10 Punkte)**

a) Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Betrieben zu entwerfen. Bei der vorhergehenden Analyse wurden folgende Eigenschaften der verschiedenen Betriebe ermittelt.

- Jeder Betrieb zeichnet sich dadurch aus, dass er eine bestimmte Anzahl von Mitarbeitern beschäftigt.
- Ein Ackerbaubetrieb ist ein Betrieb, der eine bestimmte Ackerfläche bewirtschaftet. Außerdem kann ein solcher Betrieb berechtigt sein, Agrarsubventionen zu erhalten.
- Bei einem Zuchtbetrieb ist relevant, wie viele Tiere dieser Betrieb hat. Auch ein Zuchtbetrieb kann Agrarsubventionen erhalten.
- Ein Bio-Zuchtbetrieb ist ein spezieller Zuchtbetrieb, der durch ein Biosiegel ausgezeichnet ist, dessen Name bekannt ist.
- Jeder Betrieb, der Subventionen erhalten kann, ist entweder ein (Bio-)Zuchtbetrieb oder ein Ackerbaubetrieb. Solche Betriebe nennt man auch Landwirtschaftsbetriebe.
- Ein Handwerksbetrieb ist ein Betrieb, der eine bestimmte Anzahl von Handwerksmeistern beschäftigt.
- Ein Frisörbetrieb ist ein Handwerksbetrieb, zu dem die Telefonnummer bekannt ist.
- Ein Metzgerbetrieb ist ein Handwerksbetrieb, zu dem bekannt ist, ob er Wild verkauft.
- Jeder Metzgerbetrieb und alle Landwirtschaftsbetriebe werden regelmäßig einer Gesundheitskontrollen unterzogen. Es gibt daher eine Methode, die eine solche Kontrolle in einem Betrieb durchführt und zurückliefert, ob diese Kontrolle bestanden wurde.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Betrieben. Achten Sie darauf, dass gemeinsame Merkmale in (eventuell abstrakten) Oberklassen zusammengefasst werden. Notieren Sie Ihren Entwurf graphisch und verwenden Sie dazu die folgende Notation:



Geben Sie für jede Klasse ausschließlich den jeweiligen Namen und die Namen und Datentypen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen und Ein- und Ausgabetypen seiner Methoden an.

Matrikelnummer: **Name:**

b) Implementieren Sie in Java eine Methode **betriebeKontrollieren**. Die Methode bekommt als Parameter ein Array von Betrieben übergeben. Sie soll die Betriebe, die einer Gesundheitskontrolle unterzogen werden müssen, kontrollieren. Als Ergebnis soll zurückgegeben werden, ob *alle* relevanten Betriebe die Kontrolle bestehen. Falls kein solcher Betrieb im Array existiert, ist also alles in Ordnung. Achten Sie darauf, dass immer, unabhängig vom Endergebnis, *alle* relevanten Betriebe kontrolliert werden.

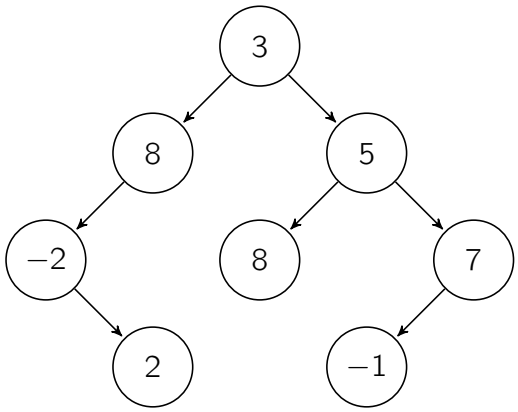
Gehen Sie dabei davon aus, dass das übergebene Array nicht **null** ist und dass es keinen **null**-Eintrag enthält. Kennzeichnen Sie die Methode mit dem Schlüsselwort **static**, falls angebracht.

Matrikelnummer: **Name:**

Aufgabe 4 (Programmierung in Java): **(10 + 6 + 10 Punkte)**

Wir betrachten Binärbaume, bei denen jeder Knoten maximal zwei Nachfolger hat. Jeder Baumknoten speichert einen **int**-Wert und referenziert zwei Nachfolger vom Typ **Baum**, wobei diese jeweils **null** sein können. Die Abbildung enthält ein Beispiel für einen solchen Binärbaum. Binärbäum-Objekte werden als Objekte der Klasse **Baum** dargestellt, wobei der leere Binärbaum durch **null** repräsentiert wird.

```
public class Baum {
  public int wert;
  public Baum links;
  public Baum rechts;
}
```



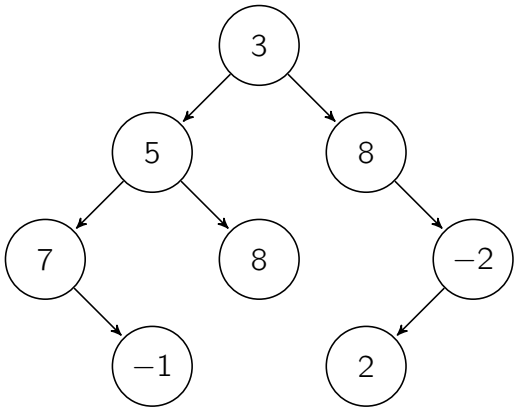
Zusätzlich ist das folgende Interface gegeben, mit dem **int**-Werte auf beliebige Eigenschaften überprüft werden können:

```
interface Tester {
  public boolean teste(int i);
}
```

Verwenden Sie in den Implementierungen dieser Aufgabe keine Schleifen, sondern ausschließlich Rekursion. Versehen Sie Methoden- und Klassendeklarationen mit **static**, falls angebracht.

a) Implementieren Sie die Methode **spiegeln**, die keinen Rückgabewert hat und einen gegebenen Baum spiegelt. Zum Spiegeln müssen für alle Knoten des Baums der linke und rechte Nachfolger vertauscht werden.

Beispielsweise wird der oben dargestellte Baum durch den Aufruf der Methode **spiegeln** so modifiziert, dass er anschließend wie folgt aussieht:



b) Schreiben Sie eine Klasse **GroesserTester**, die das Interface **Tester** implementiert. Es soll ein Konstruktor **GroesserTester(int vergleichswert)** geschrieben werden, dem man einen Vergleichswert übergibt. Die Methode **teste** soll überprüfen, ob der übergebene Wert größer als der bei der Objekterzeugung übergebene Vergleichswert ist. Falls das zutrifft, wird **true** zurückgegeben, andernfalls **false**.

c) Wir wollen nun wissen, ob es in einem Baum einen Pfad von der Wurzel bis zu einem der Blätter gibt, so dass für alle Knoten auf diesem Pfad (also einschließlich Wurzel und Blatt) die Knotenwerte eine bestimmte Eigenschaft haben. Schreiben Sie hierzu die Methode **hatPfad**, die für einen gegebenen **Baum** und eine Eigenschaft, gegebenen durch einen **Tester**, überprüft, ob es in dem Baum mindestens einen solchen Pfad gibt. Sobald ein entsprechender Pfad gefunden wird, gibt die Methode direkt **true** zurück, ohne nach weiteren Pfaden zu suchen. Nur wenn es keinen solchen Pfad gibt, ist der Rückgabewert **false**.

Für den leeren Baum definieren wir hier, dass unabhängig von der Eigenschaft ein entsprechender Pfad existiert. Gehen Sie bei der Implementierung davon aus, dass der übergebene Tester nicht **null** ist.

Als Beispiel betrachten wir den dargestellten Baum und den **GroesserTester** aus der vorherigen Teilaufgabe mit dem Vergleichswert 0. Es gibt hier genau einen Pfad ($3 \rightarrow 5 \rightarrow 8$) mit der gesuchten Eigenschaft.

Matrikelnummer: **Name:**

Aufgabe 5 (Funktionale Programmierung in Haskell): (2 + 2 + (2 + 1 + 5 + 3) Punkte)

a) Geben Sie den allgemeinsten Typ der Funktionen f und g an, die wie folgt definiert sind. Gehen Sie davon aus, dass **1** den Typ **Int** hat.

f x y = x ++ y

---

g x y z = (\z -> x+z+1) y

b) Gegeben sei das folgende Programm in Haskell:

```
h [x] i = 0
h (x:xs) i = i x (h xs i)

k x y = if x > 0 then x + y else y
```

Geben Sie das Ergebnis für den Aufruf **h [4, 5, -2, 10] k** an.

c) Ein Zug lässt sich als Liste von *Waggons* darstellen. Ein *Waggon* ist entweder ein *Güterwaggon*, ein *Personenwaggon* oder ein *Triebfahrzeug*.

- Jeder *Güterwaggon* hat ein bestimmtes Gewicht (in kg).
- In einem *Personenwaggon* befindet sich eine bestimmte Anzahl von Passagieren. Das Gewicht eines Personenwaggons ist nicht direkt bekannt, lässt sich aber wie folgt berechnen: Ein Personenwaggon mit n Passagieren wiegt $2000\ \text{kg} + (n \cdot 80)\ \text{kg}$.
- Bei einem *Triebfahrzeug* gibt die Kraft (ebenfalls in kg gemessen) an, wie viel Gewicht dieses Triebfahrzeug in Bewegung setzen kann. Die Position der Triebfahrzeuge in einem Zug ist unerheblich, ein Triebfahrzeug kann also andere Waggons ziehen, schieben oder gleichzeitig ziehen und schieben.
- Gehen sie davon aus, dass in Personenwaggons mindestens 0 Personen sitzen und jeder Güterwaggon mindestens 1 kg wiegt.

In dem folgenden Bild ist ein Zug, bestehend aus zwei Triebfahrzeugen sowie zwei Güterwaggons und einem Personenwaggon, zu sehen.

1. Geben Sie die Definition einer Datenstruktur **Waggon** für die verschiedenen Waggons an (also nicht für einen Zug aus mehreren Waggons!).

2. Geben Sie den im Bild dargestellten Zug als Haskell-Liste vom Typ **[Waggon]** an. Hierbei soll die Reihenfolge von links nach rechts (Triebfahrzeug, Personenwaggon, Güterwaggon, Triebfahrzeug, Güterwaggon) der Reihenfolge in der Liste entsprechen.

3. Schreiben Sie eine Funktion **faehrt**, die als Argument einen Zug vom Typ **[Waggon]** erhält und genau dann **True** zurückgibt, wenn die Triebfahrzeuge des Zuges stark genug sind, um den Zug in Bewegung zu setzen.

   Ein Zug kann sich genau dann in Bewegung setzen, wenn die vereinte Kraft der Triebfahrzeuge ausreicht, um das Gewicht aller restlichen Waggons zu bewegen. Der im Bild dargestellte Zug kann also fahren, da gilt:

   $$40.000\ \text{kg} - (2.000\ \text{kg} + (20 \cdot 80\ \text{kg})) - 38.000\ \text{kg} + 23.000\ \text{kg} - 20.000\ \text{kg} = 1.400\ \text{kg} \geq 0\ \text{kg}$$

   Gehen Sie hierbei davon aus, dass der leere Zug (kein Waggon) fährt. Geben Sie auch den Typ der Funktion an! Sie dürfen bei Bedarf eine Hilfsfunktion definieren und benutzen.

4. In dieser Aufgabe suchen wir eine Möglichkeit, um aus einem Zug alle Personenwaggons ohne Passagiere (also mit 0 Passagieren) zu entfernen. Alle anderen Waggons sollen im Zug erhalten bleiben. Schreiben Sie unter Verwendung der unten angegebenen Funktion **filter** einen Ausdruck, der aus einem Zug **eingabe** vom Typ **[Waggon]** genau die Personenwaggons ohne Passagiere entfernt und eine entsprechend modifizierte Version von **eingabe** zurückgibt.

   Sie dürfen für diesen Aufruf eine Hilfsfunktion definieren und benutzen.

```
filter :: (a -> Bool) -> [a] -> [a]
filter f []     = []
filter f (x:xs) = if (f x) then x:(filter f xs) else (filter f xs)
```

Matrikelnummer: **Name:**

Aufgabe 6 (Logische Programmierung in Prolog): (3 + 5 + (2 + 4 + 3 + 2) Punkte)

a) Geben Sie für die folgenden Paare von Termen den allgemeinsten Unifikator an oder begründen Sie kurz, warum dieser nicht existiert.

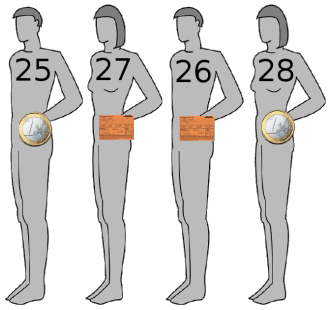
f(X,Z,c) und **f(a,g(X,Y),Y)**

f(X,b,Y) und **f(a,Y,g(a))**

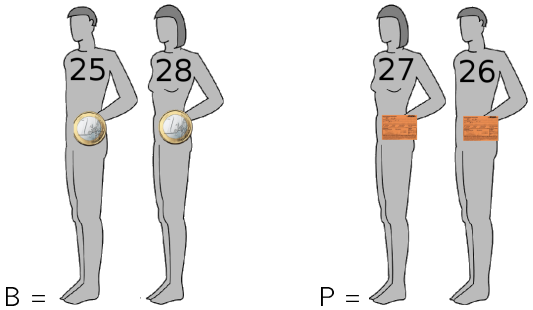
b) Erstellen Sie für das folgende Logikprogramm zur Anfrage **?- f(b(a), A).** den Beweisbaum und geben Sie alle Antwortsubstitutionen an. Sie dürfen dabei abbrechen, sobald die Anfrage aus mindestens drei Teilen (Atomen) bestehen würde. Kennzeichnen Sie solche Knoten durch „...“. Kennzeichnen Sie abgebrochene Pfade durch „↯“.

```
h(X, X).
h(i(X), b(X)).
f(X, Y) :- h(Y, X).
f(X, Y) :- f(Z, Y), h(X, Y).
```

c) In dieser Aufgabe möchten wir uns mit einer Postfiliale beschäftigen, in der einige Kunden in einer Warteschlange stehen und entweder Briefmarken kaufen oder Pakete abholen möchten. Die Kunden, die Briefmarken kaufen möchten, halten das entsprechende Geld in der Hand. Die anderen Kunden halten einen Paketabholschein (Abkürzung: PAS) in der Hand. Weiterhin ist zu jedem Kunden bekannt, der wievielte Kunde des Tages er ist.



(a) Beispiel-Warteschlange



(b) Antwortsubstitution

In der in Zeichnung (a) dargestellten Warteschlange stehen vier Kunden. Der Kunde am vorderen Ende der Warteschlange (links) ist der 25. Kunde des Tages und hat Geld für den Kauf von Briefmarken in der Hand. Von vorne gesehen der zweite Kunde ist der 27. Kunde des Tages und hat einen Paketabholschein in der Hand. Dahinter steht der 26. Kunde des Tages, welcher ebenfalls ein Paket abholen möchte. Am Ende der Warteschlange (rechts) steht der 28. Kunde des Tages, der Geld für Briefmarken in der Hand hält.

1. Überlegen Sie, wie Sie die beiden Eigenschaften der Kunden durch einen zweistelligen Term **kunde** darstellen können. Benutzen Sie die vordefinierten Listen und geben Sie damit eine Termdarstellung der in (a) dargestellten Beispiel-Warteschlange an. Wählen Sie die Darstellung so, dass das vordere Ende der Warteschlange (hier also der 25. Kunde des Tages) durch das erste Listenelement repräsentiert wird.

2. Die Post hat erkannt, dass viele Kunden nur Pakete abholen möchten und bietet daher einen eigenen Schalter für diese Kunden an. Programmieren Sie das dreistelliges Prädikat **schlangenTrennen**. Das Prädikat soll dafür verwendet werden, zu einer Warteschlange E zwei Warteschlangen B und P anzugeben, wobei alle Kunden in B Briefmarken kaufen möchten und alle Kunden in P Pakete abholen möchten. Beachten Sie hierbei, dass die Reihenfolge der Kunden untereinander nicht verändert wird und jeder Kunde aus E auch in B oder P auftaucht. Verwenden Sie keine vordefinierten Prädikate.

   Falls das erste Argument E die in (a) dargestellte Warteschlange ist, resultiert der Aufruf

   ?- schlangenTrennen(E, B, P).

   für ungebundene Variablen B und P also in einer Antwortsubstitution, die der Darstellung (b) entspricht.

3. Manchmal sind Kunden zu ungeduldig und versuchen sich vorzudrängeln - also den Platz mit einer Person zu tauschen, die schon länger wartet. In der Beispiel-Warteschlange steht der 27. Kunde des Tages vor dem 26. Kunden des Tages, hat sich also vorgedrängelt.

   Programmieren Sie das einstellige Prädikat **reihenfolgeGut**, das überprüft, ob in einer Warteschlange alle Kunden in der richtigen Reihenfolge stehen.

   Die Anfrage **?- reihenfolgeGut(E).** gibt also genau dann die Antwort **true**, wenn die *„Kunde des Tages“*-Nummer eines jeden Kunden in der Warteschlange E kleiner ist als die entsprechende Nummer aller Kunden weiter hinten in dieser Warteschlange. Beispielsweise ergibt **?- reihenfolgeGut(E).** für die Beispiel-Warteschlange **false**. Verwenden Sie außer $<$ kein vordefiniertes Prädikat.

4. Kombinieren Sie die beiden Prädikate **schlangenTrennen** und **reihenfolgeGut** so in einer Anfrage, dass zu einer Warteschlange nur die Reihenfolge der Kunden überprüft wird, die einen Paketabholschein in der Hand halten. Geben Sie eine Anfrage an, die zu einer Warteschlange genau dann **true** bzw. eine Antwortsubstitution zurückgibt, wenn die Kunden mit Paketabholschein untereinander in der richtigen Reihenfolge stehen. Gehen Sie hierbei davon aus, dass die Variable **E** mit der zu überprüfenden Eingabe-Warteschlange belegt ist.