Freie und Hansestadt Hamburg
Behörde für Schule und Berufsbildung

_____/_____
Name / Kurs-Nr.

Schriftliche Abiturprüfung
Schuljahr 2022/2023

Informatik

auf erhöhtem Anforderungsniveau

an allgemeinbildenden gymnasialen Oberstufen

Haupttermin
Donnerstag, 20. April 2023, 09:00 Uhr

Unterlagen für die Prüflinge

Allgemeine Arbeitshinweise

- Überprüfen Sie diese Unterlagen auf Vollständigkeit.
- Schreiben Sie auf alle Prüfungsunterlagen Ihren Namen und zusätzlich auf dieses Deckblatt Ihre Kursnummer.
- Kennzeichnen Sie Ihre Entwurfsblätter (Kladde) und Ihre Reinschrift.

Fachspezifische Arbeitshinweise[1]

- Die Arbeitszeit beträgt **330 Minuten**.[2]
- Eine Lese- und Auswahlzeit von **30 Minuten** ist der Arbeitszeit vorgeschaltet. In dieser Zeit darf noch nicht mit der Bearbeitung der Aufgaben begonnen werden.
- Hilfsmittel: Taschenrechner (nicht programmierbar, nicht grafikfähig), Formelsammlung „Das große Tafelwerk“ (Cornelsen-Verlag), Rechtschreibwörterbuch

Aufgabenauswahl

- Sie erhalten **drei** Aufgaben zu unterschiedlichen Schwerpunkten. Die Aufgabe I erhalten Sie in einer Java-Version – die Aufgaben II und III in jeweils einer Scheme- und einer Haskell-Version.
- Bearbeiten Sie die **Pflichtaufgabe (Aufgabe I)** und **eine** der Aufgaben II oder III in einer der zur Verfügung gestellten Programmiersprachen (Scheme oder Haskell).
- Vermerken Sie auch auf Ihrer Reinschrift, welche Aufgaben Sie in welcher Programmiersprache ausgewählt und bearbeitet haben.

Bearbeitet wurden die folgenden Aufgaben (bitte kreuzen Sie an):

I	Objektorientierte Modellierung und Programmierung (Pflicht)	
II	Datensicherheit in verteilten Systemen	(☐ Scheme ☐ Haskell)
III	Intelligente Suchverfahren	(☐ Scheme ☐ Haskell)

[1] Entsprechend der „Richtlinie über die Gewährung von Erleichterungen für neu zugewanderte Schülerinnen, Schüler und Prüflinge bei Sprachschwierigkeiten in der deutschen Sprache“ (MBlSchul Nr. 08, 7. Oktober 2016, S. 60) werden für die betroffenen Prüflinge die folgenden Erleichterungen gewährt:
- Die Bearbeitungszeit wird um 30 Minuten auf **360 Minuten** erhöht.
- Ein nicht-elektronisches Wörterbuch Deutsch – Herkunftssprache / Herkunftssprache – Deutsch wird bereitgestellt.

[2] Gemäß Entscheidung vom 5. Januar 2023.

Bewertung

Jeder Aufgabe sind 50 Bewertungseinheiten (BE) zugeordnet. In allen Teilaufgaben werden nur ganze BE vergeben. Insgesamt sind 100 BE erreichbar. Bei der Festlegung von Notenpunkten gilt die folgende Tabelle:

Erbrachte Leistung (in BE)	Notenpunkte
≥ 95	15
≥ 90	14
≥ 85	13
≥ 80	12
≥ 75	11
≥ 70	10
≥ 65	9
≥ 60	8

Erbrachte Leistung (in BE)	Notenpunkte
≥ 55	7
≥ 50	6
≥ 45	5
≥ 40	4
≥ 33	3
≥ 27	2
≥ 20	1
< 20	0

Für die Erteilung der **Note gut** (11 Punkte) ist mindestens erforderlich, dass annähernd vier Fünftel der erwarteten Gesamtleistung sowie Leistungen in allen drei Anforderungsbereichen erbracht werden. Dabei muss die Prüfungsleistung in ihrer Gliederung, in der Gedankenführung, in der Anwendung fachmethodischer Verfahren sowie in der fachsprachlichen Artikulation den Anforderungen voll entsprechen.

Für die Erteilung der **Note ausreichend** (5 Punkte) ist mindestens erforderlich, dass annähernd die Hälfte der erwarteten Gesamtleistung und über den Anforderungsbereich I hinaus Leistungen in einem weiteren Anforderungsbereich erbracht werden.

Die zwei voneinander unabhängigen Aufgaben der Prüfungsaufgabe werden jeweils mit 50 Bewertungseinheiten bewertet. Die erbrachte Gesamtleistung ergibt sich aus der Summe der Bewertungseinheiten in den beiden Aufgaben.

Bei erheblichen Mängeln in der sprachlichen Richtigkeit und der äußeren Form sind bei der Bewertung der schriftlichen Prüfungsleistung je nach Schwere und Häufigkeit der Verstöße bis zu zwei Notenpunkte abzuziehen. Dazu gehören auch Mängel in der Gliederung, Fehler in der Fachsprache, Ungenauigkeiten in Zeichnungen sowie falsche Bezüge zwischen Zeichnungen und Text.

Aufgabe I: Grundriss 50 BE

Schwerpunkt: Objektorientierte Modellierung und Programmierung von Grafiksystemen

Für Architektenzeichnungen soll eine Software zur Darstellung von Grundrissen entwickelt werden. Eine erste Version kann Wände, Türen und Fenster darstellen. Ein Beispiel zeigt die folgende Abbildung 1:

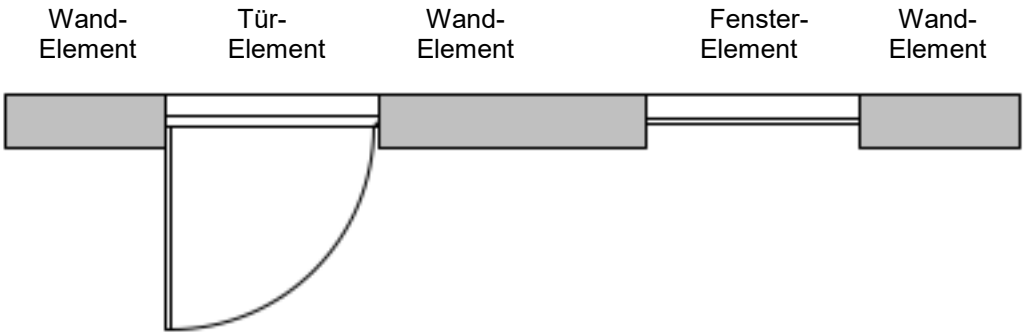


Abbildung 1

Die folgende Abbildung 2 zeigt ein passendes Klassendiagramm zu dieser Software-Version:

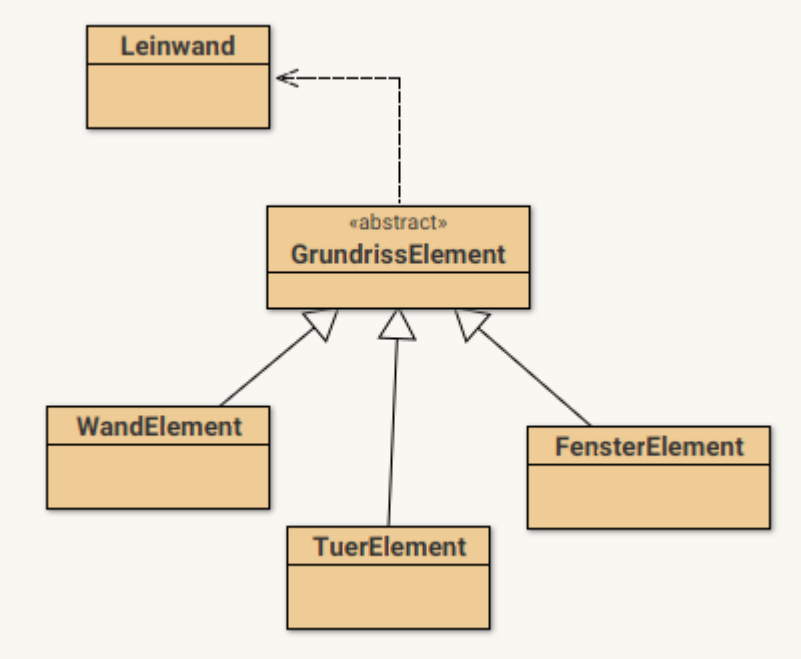


Abbildung 2

I.a **Erläutern** Sie das angegebene Klassendiagramm (Abbildung 2) zu dieser ersten Version der Software.

(5 BE)

I.b **Beschreiben** Sie, wodurch
- Vererbungsbeziehungen und
- Nutzerbeziehungen (auch: Verwendungsbeziehungen)

jeweils im Programmtext einer Klasse erkennbar sind, und **erläutern** Sie diese Beziehungstypen jeweils an einem einfachen Beispiel im Kontext der Aufgabe.

(5 BE)

I.c **Begründen** Sie, warum es sinnvoll ist, eine Klasse `GrundrissElement` (vgl. Abbildung 2) in der Modellierung einzuführen.

(5 BE)

Beim Einbauen weiterer Wandelemente, die in der ersten Modellierung als einfache Rechtecke modelliert worden sind, treten einige Ränder der Rechtecke als Trennstriche zu anderen Wandelementen auf.

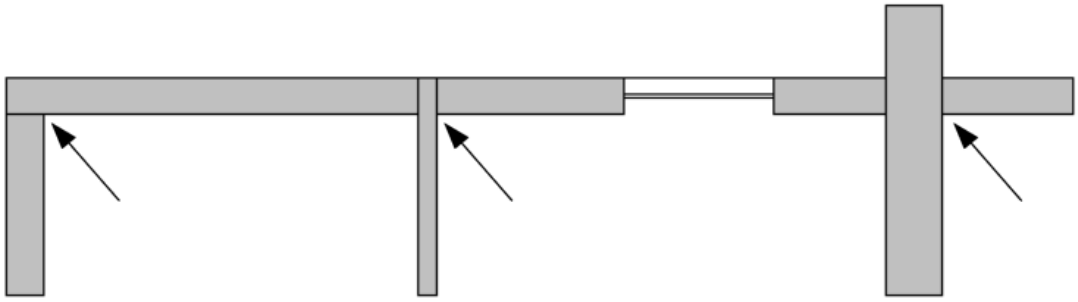


Abbildung 3

Die Entwickler*innen möchten Wandabschnitte aber mit einer durchgehenden hellgrauen Füllung dargestellt haben.

Als zweiten Entwurf verwenden sie für Wandelemente nicht ein einfaches Rechteck, sondern ein n-Eck, also eine geschlossene Figur mit gradlinigen Kanten, die mehr als vier Eckpunkte haben kann. Die folgende Abbildung 4 zeigt ein Beispiel:

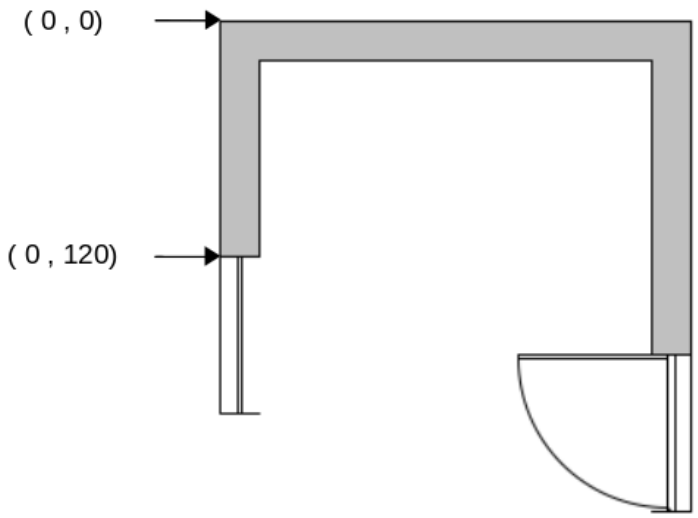


Abbildung 4

Bezogen auf die linke obere Ecke mit den (relativen) Koordinaten (0,0) können die (ebenfalls relativen) Koordinaten der Eckpunkte leicht angegeben werden, wenn man die einzelnen Längen kennt. Die Entwickler*innen entscheiden sich, die Daten intern in einer `ArrayList` zu speichern, die Elemente einer (*internen*) Klasse `Ecke` verwaltet (siehe dazu die Anlage 2).

I.d **Geben** Sie die (relativen) Koordinaten in Bildschirmpixeln in einer passenden Reihenfolge zu dem dargestellten Wandelement **an**. Verwenden Sie für diese Teilaufgabe auch das Bild in der Anlage 1, das mit Bemessungslinien versehen ist.

- **Stellen** Sie den Kopf eines Konstruktors und dessen notwendigen Inhalt für solche Wandelemente **dar**, der mit den übergebenen relativen Koordinaten der Eckpunkte (beispielsweise als `int[][] koordinaten, int[] koordinaten` oder als `ArrayList<Integer> koordinaten`, …) mit Hilfe einer passenden Methode `erzeugeEcken(koordinaten)` das Wandelement an der gewünschten Position mit der gewünschten Orientierung erzeugt.

(12 BE)

Um ein bereits eingebautes Wandelement um weitere Eckpunkte erweitern zu können, soll die Klasse eine Methode
`public void neueEcke (int neuX, int neuY, int ecke1X, int ecke1Y, int ecke2X, int ecke2Y)`
(alternativ: `public void neueEcke (int[] neu, int[] ecke1, int[] ecke2)`)
bereitstellen, die zwischen zwei der bisherigen Ecken (`ecke1` und `ecke2`), die mit einer Kante verbunden sind, eine neue Ecke (`neu`) einfügt.

I.e **Entwickeln** Sie diese Methode `neueEcke(…)` (siehe oben).
Gewährleisten Sie bei Ihrer Lösung, dass die neue Ecke nur dann eingefügt wird, wenn `ecke1` und `ecke2` benachbarte Ecken sind.
Sie können davon ausgehen, dass `ecke1` und `ecke2` in der Eckenliste vorhanden sind.
Weiterhin können Sie davon ausgehen, dass die Klasse `WandElement` eine Methode `public int indexVonEcke(int x, int y)` bereitstellt, die zu den beiden Koordinatenwerten die Position der Ecke in der `ArrayList` zurückgibt.

(12 BE)

Die Entwickler*innen fürchten, dass die Bestimmung der Eckpunkte für die Anwender*innen möglicherweise zu kompliziert ist. Daher versuchen sie einen anderen Entwurf, bei dem Wandelemente allein rechteckig sind, zusätzlich aber die Klassen `EckElement`, `TElement` und `PlusElement` realisiert werden (siehe dazu auch die Anlage 3 derselben Grundrisselemente in zerfallender Darstellung).

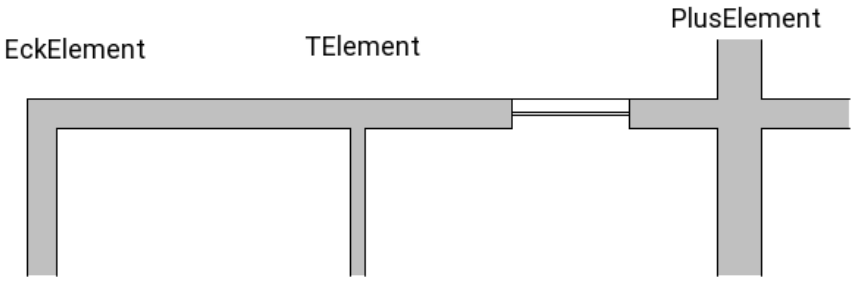


Abbildung 5

I.f **Stellen** Sie eine mögliche Erweiterung des Klassendiagramms um die Klassen (`EckElement`, `TElement` und `PlusElement`) **dar**. (Alternativ können Sie diese Erweiterung auch **skizzieren**.)

(5 BE)

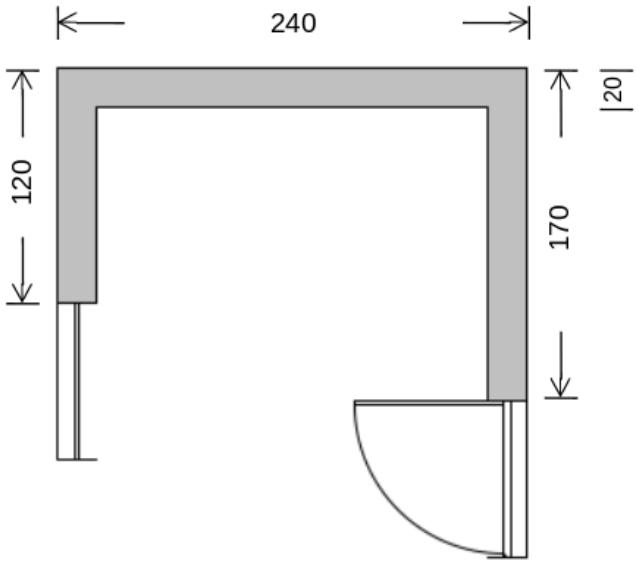
Alle diese im erweiterten Entwurf verwendeten Wandelemente sind nun aber nicht mehr als abgeschlossene Figuren wie Rechtecke o. ä. realisiert, damit sie sich ohne Trennstriche miteinander verbinden lassen. Das hat allerdings zur Folge, dass der Einsatz einer Füllung nicht funktioniert, da der zu füllende Bereich für die Grafik nicht eindeutig definiert ist.
Eine Möglichkeit, das Problem zu lösen, ist das gesonderte Darstellen der Füllung durch ein spezielles `GrundrissElement`, dessen Klasse `Fuellung` benannt wird.

I.g **Beschreiben** Sie eine Erweiterung des Klassendiagramms um die Klasse `Fuellung` und **erläutern** Sie die neu aufgetretenen Beziehungen.

(6 BE)

Anlagen zur Aufgabe I Grundriss

Anlage 1 Bild mit Bemessungslinien

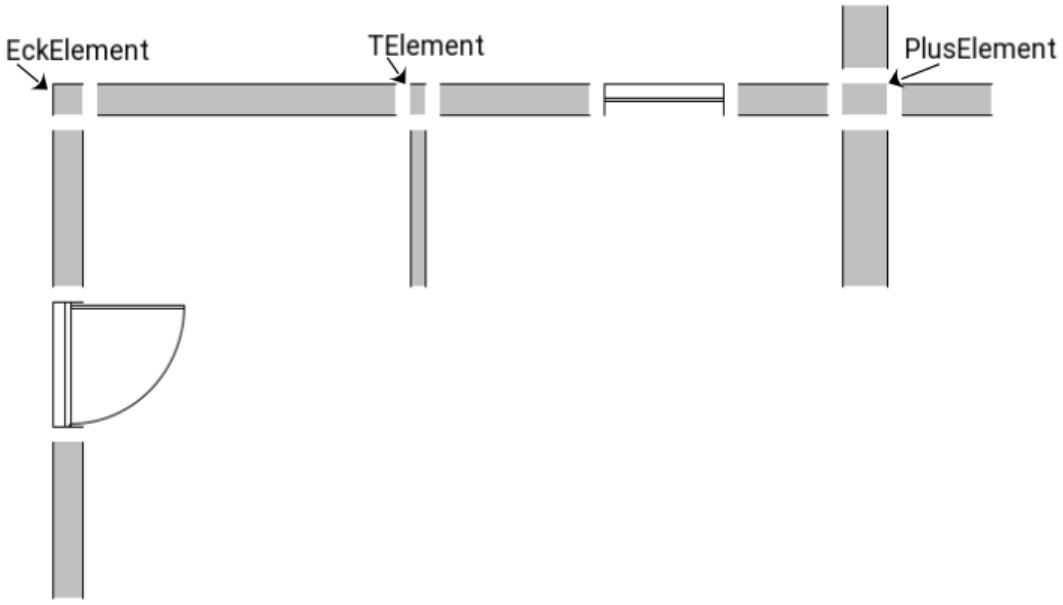


Alle Größen sind in Pixel angegeben. Die Wanddicke der `WandElemente` ist durchgehend 20 Pixel.

Anlage 2 Interne Klasse `Ecke`

```
1  /*
2   * Interne Klasse Ecke
3   */
4  private class Ecke
5  {
6      private int x,y;
7
8      public Ecke(int x, int y) {
9          this.x=x;
10         this.y=y;
11     }
12
13     public Ecke(int[] koordinatenPaar) {
14         this.x=koordinatenPaar[0];
15         this.y=koordinatenPaar[1];
16     }
17
18     public int gibX() {return x;}
19     public int gibY() {return y;}
20
21     public boolean istEcke(int andereX, int andereY) {
22         if (andereX!=x) return false;
23         if (andereY!=y) return false;
24         return true;
25     }
26
27 }
```

Anlage 3 Elemente in getrennter Darstellung



Aufgabe II: Datensicherheit in der Cloud (Scheme-Version) 50 BE

Schwerpunkt: Datensicherheit in verteilten Systemen

Immer wieder werden interne Daten der Firma *Skilpaddel* öffentlich bekannt. Dies gilt sowohl für Daten, die nie über das Internet verschickt worden sind, als auch für Daten, die in einer Cloud[1] gespeichert werden, was auf Sicherheitslücken innerhalb der Firma hindeutet.
Die Sicherheitschefin der Firma möchte deshalb als eine erste Maßnahme die Dokumente, die auf den Rechnern der Firma gespeichert sind, verschlüsseln.
Sie stellt der Firmenleitung verschiedene symmetrische und asymmetrische Verfahren vor. Die Firmenleitung entscheidet sich schließlich dafür, ein symmetrisches Verfahren einzusetzen.

II.a
- **Nennen** Sie die wesentlichen Unterschiede zwischen symmetrischen und asymmetrischen Verfahren.
- **Stellen** Sie **dar**, warum die Entscheidung der Firmenleitung für ein symmetrisches Verfahren hier passend ist.

(14 BE)

Die Firmenleitung hat sich weiter informiert und gelesen, dass symmetrische Verfahren, bei denen der Schlüssel mindestens so lang ist wie der zu verschlüsselnde Text, nicht knackbar sind. Sie entscheidet, dass ein solches Verfahren innerhalb der Firma für die Verschlüsselung der internen Dokumente eingeführt werden soll.
Damit sich die Angestellten der Firma den Schlüssel zum Ver- und Entschlüsseln ihrer eigenen Dokumente leichter merken können, gibt sie die folgende Funktion zur Schlüsselgenerierung vor:

```
1  ;(schluesselGenerator '(f r i t z) '(b r a u s e) 15)
2  ; -> '(f r i t z b r a u s e f r i t z b r a u s e)
3  (define (schluesselGenerator vorname nachname textLaenge)
4    (generiere (append vorname nachname)
5               textLaenge
6               '()))
7
8  (define (generiere teilSchluessel textLaenge ausgabe)
9    (cond ((>= (laenge ausgabe) textLaenge) ausgabe)
10         (else (generiere teilSchluessel
11                          textLaenge
12                          (append ausgabe teilSchluessel)))))
13
14 ;Unterfunktion um die Länge einer übergebenen Liste zu bestimmen
15 ; '(a b c d) -> 4
16 (define (laenge liste)
17   ...
```

II.b
- **Stellen** Sie die Funktionsweise der Funktion
  `(define (schluesselGenerator vorname nachname textLaenge)`
  im Detail **dar**.
- **Implementieren** Sie die noch fehlende Funktion
  `(define (laenge liste),`
  welche die Länge der übergebenen Liste als Zahl zurückgibt.

(8 BE)

Um die Schlüsselgenerierung bei langen Texten effizienter zu gestalten, wurden die Funktionen `schluesselGenerator` und `generiere` neu implementiert:

[1] Eine Cloud sind ein oder mehrere Server, auf die über das Internet zugegriffen wird. Die Cloud-Server befinden sich in Rechenzentren, die weltweit verteilt sein können. (Für eine ausführliche Erläuterung siehe die Anlagen 1 und 2.)

```
1  (define (schluesselGenerator2 vorname nachname textLaenge)
2    (generiere2 textLaenge(append vorname nachname)))
3
4  (define (generiere2 textLaenge ausgabe)
5    (cond ((>= (laenge ausgabe) textLaenge) ausgabe)
6          (else (generiere2 textLaenge
7                            (append ausgabe ausgabe)))))
```

II.c
- **Erklären** Sie, wieso die Funktion `generiere2` effizienter arbeitet als die Funktion `generiere`.
- **Begründen** Sie, weshalb dieses Verfahren zur Schlüsselgenerierung nach dem Prinzip von Kerckhoffs nicht sicher ist.

(10 BE)

Die Sicherheitschefin überlegt, wie sie die Schlüsselgenerierung sicherer machen könnte. Anstatt immer wieder die gleichen `vorname` und `nachname` hinten an die Schlüsselliste zu hängen, soll bei jedem neuen Anhängen jeder Buchstabe von `vorname` und `nachname` um einen bestimmten Zahlenwert im Alphabet nach „rechts“ geschoben werden (für die Umsetzung siehe Anlage 3).

II.d **Implementieren** Sie ausgehend von dem Ansatz in Anlage 3 eine Lösung, bei der eine Verschiebung der Buchstaben um beliebige Werte möglich ist.

(4 BE)

Da für die Anbindung verschiedener Geschäftsstellen regelmäßig Informationen auch in Cloud-Diensten (siehe Anlage 1) bereitgestellt werden müssen, geht die Sicherheitschefin nun der Frage nach, ob es hier eventuell zusätzliche Sicherheitslücken gibt.

II.e **Stellen** Sie mögliche Sicherheitsrisiken **dar**, die durch die Nutzung eines Cloud-Dienstes entstehen können.

(6 BE)

Der Cloud-Dienstleister der Firma *Skilpaddel* wurde danach ausgewählt, dass er eine Ende-zu-Ende-Verschlüsselung anbietet (siehe Anlage 2). Annabel, eine Mitarbeiterin der Firma, möchte Max, einem als vertrauenswürdig eingestuften Kollegen, Daten über die Cloud zur Verfügung stellen.

II.f
- **Erklären** Sie anhand des gegebenen Sachkontexts, wie eine „echte“ Ende-zu-Ende-Verschlüsselung aussehen könnte, bei der Annabel ihre Daten in der Cloud speichern und Max sie dann herunterladen und lesen kann.
- **Untersuchen** Sie, welche Sicherheitsrisiken sich bei einer passwortbasierten „Ende-zu-Ende-Verschlüsselung“ (siehe Klasse 2 in der Anlage 2) ergeben.

(8 BE)

Anlagen zur Aufgabe II Datensicherheit in der Cloud (Scheme-Version)

Anlage 1 Cloud-Computing

„Cloud Computing ist ein Modell, das es erlaubt, bei Bedarf jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können."

(Definition der US-amerikanischen Standardisierungsstelle NIST (*National Institute of Standards and Technology*), die auch von der ENISA (*European Network and Information Security Agency*) genutzt wird)

Ein Cloud-Dienstleister bietet diese Services an.

Anlage 2 Datenverschlüsselung in der Cloud

„Daten in Cloud-Diensten zu verschlüsseln, ist weitaus schwieriger. Die Datenverarbeitung findet beim Diensteanbieter statt. Das heißt, eine echte Ende-zu-Ende-Verschlüsselung ist beinahe unmöglich, falls der Diensteanbieter dies nicht explizit unterstützt.

[...]

Klasse 1: "Echte" Ende-zu-Ende-Verschlüsselung

Die Daten werden auf dem Endgerät des Anwenders verschlüsselt. Der zur Verschlüsselung genutzte Schlüssel wird lokal auf dem Endgerät erzeugt und gespeichert. Ein Upload des Schlüssels in die Cloud erfolgt nicht. Der Anwender muss zur Vermeidung von Datenverlusten selbst für ein Schlüssel-Backup sorgen. Die eigentlichen Daten werden vor dem Upload in die Cloud durch das Endgerät verschlüsselt, entsprechend erfolgt die Entschlüsselung nur auf dem Gerät.

Klasse 2: Passwortbasierte Ende-zu-Ende-Verschlüsselung

Der Dienst erzeugt den zur Verschlüsselung genutzten Schlüssel direkt oder indirekt aus dem Passwort des Anwenders. Das Passwort wird nicht gespeichert, sondern beim Login in die Applikation wird nach Industriestandards durch eine sichere Hashwert-Berechnung der Verschlüsselungsschlüssel wiederhergestellt. Die Ver- und Entschlüsselung der Daten erfolgt in der Applikation auf dem Server des Diensteanbieters. Da das Passwort und die Daten erst auf dem Server verschlüsselt werden, ist eine funktionierende Transportverschlüsselung (HTTPS) erforderlich.

Sollten Sie der Verschlüsselung der Klasse 2 vertrauen, ist die Frage nach dem Fall des Passwortverlustes zusätzlich interessant. Ist es dem Anbieter möglich, Ihre Daten auch bei Passwortverlust wiederherzustellen, so erfolgt eine zusätzliche Verschlüsselung Ihrer Daten mit einem Masterkey oder einer vergleichbaren Lösung. Hierfür gibt es verschiedene Implementierungsmöglichkeiten, die die Stärke der Verschlüsselung massiv abschwächen können."[2]

[2] Hißen, Frank (16.10.2015): Verschlüsselung ist nicht gleich Verschlüsselung, Datenverschlüsselung in der Cloud, [online] https://www.tecchannel.de/a/verschluesselung-ist-nicht-gleich-verschluesselung,3281630,2 [abgerufen am 02.07.2022].

Anlage 3 Implementierung zu Aufgabenteil II.d

Würde vor jedem erneuten Anhängen jeder Buchstabe von `vorname` und `nachname` um den Wert eins im Alphabet nach „rechts“ geschoben, so würde im ersten Schritt aus der Liste `'(f r i t z b r a u s e)` die Liste `'(g s j u a c s b v t f)` entstehen und in einem nächsten Schritt dann die Liste `'(h t k v b d t c w u g)`.

Insgesamt erhält man so die folgende Schlüsselliste:

`'(f r i t z b r a u s e g s j u a c s b v t f h t k v b d t c w u g).`

Folgende Funktionen liefern ein solches Ergebnis:

(*Gehen Sie zur Vereinfachung davon aus, dass die übergebene Liste aus char-Zeichen besteht, also* `'(#\f #\r #\i #\t #\z #\b #\r #\a #\u #\s #\e)`. *Das Ergebnis wird ebenfalls in dieser Form ausgegeben. Beachten Sie außerdem, dass lediglich Kleinbuchstaben verwendet werden. „97“ ist der ASCII-Code für den Buchstaben „a“.*)

```
1   (define (verschiebe charListe ausgabe)
2     (cond ((null? charListe) ausgabe)
3           (else (verschiebe (rest charListe)
4                             (append ausgabe
5                                     (list (berechneNaechsten
6                                                   (first charListe))))))))
7
8   (define (berechneNaechsten buchstabe)
9     (zahlinbuchstabe (modulo (+ 1 (buchstabeinzahl buchstabe)) 26)))
10
11  (define (buchstabeinzahl buchstabe)
12    (- (char->integer buchstabe) 97))
13
14  (define (zahlinbuchstabe zahl)
15    (integer->char (+ 97 zahl)))
```

Aufgabe II: Datensicherheit in der Cloud (Haskell-Version) 50 BE

Schwerpunkt: Datensicherheit in verteilten Systemen

Immer wieder werden interne Daten der Firma *Skilpaddel* öffentlich bekannt. Dies gilt sowohl für Daten, die nie über das Internet verschickt worden sind, als auch für Daten, die in einer Cloud[1] gespeichert werden, was auf Sicherheitslücken innerhalb der Firma hindeutet.
Die Sicherheitschefin der Firma möchte deshalb als eine erste Maßnahme die Dokumente, die auf den Rechnern der Firma gespeichert sind, verschlüsseln.
Sie stellt der Firmenleitung verschiedene symmetrische und asymmetrische Verfahren vor. Die Firmenleitung entscheidet sich schließlich dafür, ein symmetrisches Verfahren einzusetzen.

II.a
- **Nennen** Sie die wesentlichen Unterschiede zwischen symmetrischen und asymmetrischen Verfahren.
- **Stellen** Sie **dar**, warum die Entscheidung der Firmenleitung für ein symmetrisches Verfahren hier passend ist.

(14 BE)

Die Firmenleitung hat sich weiter informiert und gelesen, dass symmetrische Verfahren, bei denen der Schlüssel mindestens so lang ist wie der zu verschlüsselnde Text, nicht knackbar sind. Sie entscheidet, dass ein solches Verfahren innerhalb der Firma für die Verschlüsselung der internen Dokumente eingeführt werden soll.
Damit sich die Angestellten der Firma den Schlüssel zum Ver- und Entschlüsseln ihrer eigenen Dokumente leichter merken können, gibt sie die folgende Funktion zur Schlüsselgenerierung vor:

```
1  schluesselGenerator "fritz" "brause" 15
2  -> "fritzbrausefritzbrause"
3
4  > schluesselGenerator vorname nachname textLaenge = generiere
   (vorname ++ nachname) textLaenge []
5
6  > generiere teilSchluessel textLaenge ausgabe
7  > | ((laenge ausgabe) >= textLaenge) = ausgabe
8  > | otherwise                                  = generiere
   teilSchluessel textLaenge (teilSchluessel ++ ausgabe)
9
10 Unterfunktion, um die Länge einer übergebenen Liste zu bestimmen
11 laenge [a,b,c,d] -> 4
12 > laenge (x:xs)  ...?...
```

II.b
- **Stellen** Sie die Funktionsweise der Funktion `schluesselGenerator vorname nachname textLaenge` im Detail **dar**.
- **Implementieren** Sie die noch fehlende Funktion `laenge (x:xs),` welche die Länge der übergebenen Liste als Zahl zurückgibt.

(8 BE)

Um die Schlüsselgenerierung bei langen Texten effizienter zu gestalten, wurden die Funktionen `schluesselGenerator` und `generiere` neu implementiert:

[1] Eine Cloud sind ein oder mehrere Server, auf die über das Internet zugegriffen wird. Die Cloud-Server befinden sich in Rechenzentren, die weltweit verteilt sein können. (Für eine ausführliche Erläuterung siehe die Anlagen 1 und 2.)

```
1  schluesselGenerator2 vorname nachname textLaenge = generiere2
   textLaenge (vorname ++ nachname)
2
3  generiere2 textLaenge ausgabe
4   | (laenge ausgabe) >= textLaenge   = ausgabe
5   | otherwise                        = generiere2 textLaenge
   (ausgabe ++ ausgabe)
```

II.c
- **Erklären** Sie, weshalb die Funktion `generiere2` effizienter arbeitet als die Funktion `generiere`.
- **Begründen** Sie, dass dieses Verfahren zur Schlüsselgenerierung nach dem Prinzip von Kerckhoffs nicht sicher ist.

(10 BE)

Die Sicherheitschefin überlegt, wie sie die Schlüsselgenerierung sicherer machen könnte. Anstatt immer wieder die gleichen `vorname` und `nachname` hinten an die Schlüsselliste zu hängen, soll bei jedem neuen Anhängen jeder Buchstabe von `vorname` und `nachname` um einen bestimmten Zahlenwert im Alphabet nach „rechts“ geschoben werden (für die Umsetzung siehe Anlage 3).

II.d **Implementieren** Sie ausgehend von dem Ansatz in Anlage 3 eine Lösung, bei der eine Verschiebung der Buchstaben um beliebige Werte möglich ist.

(4 BE)

Da für die Anbindung verschiedener Geschäftsstellen regelmäßig Informationen auch in Cloud-Diensten (siehe Anlage 1) bereitgestellt werden müssen, geht die Sicherheitschefin nun der Frage nach, ob es hier eventuell zusätzliche Sicherheitslücken gibt.

II.e **Stellen** Sie mögliche Sicherheitsrisiken **dar**, die durch die Nutzung eines Cloud-Dienstes entstehen können.

(6 BE)

Der Cloud-Dienstleister der Firma *Skilpaddel* wurde danach ausgewählt, dass er eine Ende-zu-Ende-Verschlüsselung anbietet (siehe Anlage 2). Annabel, eine Mitarbeiterin der Firma, möchte Max, einem als vertrauenswürdig eingestuften Kollegen, Daten über die Cloud zur Verfügung stellen.

II.f
- **Erklären** Sie anhand des gegebenen Sachkontexts, wie eine „echte“ Ende-zu-Ende-Verschlüsselung aussehen könnte, bei der Annabel ihre Daten in der Cloud speichern und Max sie dann herunterladen und lesen kann.
- **Untersuchen** Sie, welche Sicherheitsrisiken sich bei einer passwortbasierten „Ende-zu-Ende-Verschlüsselung“ (siehe Klasse 2 in der Anlage 2) ergeben.

(8 BE)

Anlagen zur Aufgabe II Datensicherheit in der Cloud (Haskell-Version)

Anlage 1 Cloud-Computing

„Cloud Computing ist ein Modell, das es erlaubt, bei Bedarf jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können."

(Definition der US-amerikanischen Standardisierungsstelle NIST (*National Institute of Standards and Technology*), die auch von der ENISA (*European Network and Information Security Agency*) genutzt wird)

Ein Cloud-Dienstleister bietet diese Services an.

Anlage 2 Datenverschlüsselung in der Cloud

„Daten in Cloud-Diensten zu verschlüsseln, ist weitaus schwieriger. Die Datenverarbeitung findet beim Diensteanbieter statt. Das heißt, eine echte Ende-zu-Ende-Verschlüsselung ist beinahe unmöglich, falls der Diensteanbieter dies nicht explizit unterstützt.

[...]

Klasse 1: "Echte" Ende-zu-Ende-Verschlüsselung

Die Daten werden auf dem Endgerät des Anwenders verschlüsselt. Der zur Verschlüsselung genutzte Schlüssel wird lokal auf dem Endgerät erzeugt und gespeichert. Ein Upload des Schlüssels in die Cloud erfolgt nicht. Der Anwender muss zur Vermeidung von Datenverlusten selbst für ein Schlüssel-Backup sorgen. Die eigentlichen Daten werden vor dem Upload in die Cloud durch das Endgerät verschlüsselt, entsprechend erfolgt die Entschlüsselung nur auf dem Gerät.

Klasse 2: Passwortbasierte Ende-zu-Ende-Verschlüsselung

Der Dienst erzeugt den zur Verschlüsselung genutzten Schlüssel direkt oder indirekt aus dem Passwort des Anwenders. Das Passwort wird nicht gespeichert, sondern beim Login in die Applikation wird nach Industriestandards durch eine sichere Hashwert-Berechnung der Verschlüsselungsschlüssel wiederhergestellt. Die Ver- und Entschlüsselung der Daten erfolgt in der Applikation auf dem Server des Diensteanbieters. Da das Passwort und die Daten erst auf dem Server verschlüsselt werden, ist eine funktionierende Transportverschlüsselung (HTTPS) erforderlich.

Sollten Sie der Verschlüsselung der Klasse 2 vertrauen, ist die Frage nach dem Fall des Passwortverlustes zusätzlich interessant. Ist es dem Anbieter möglich, Ihre Daten auch bei Passwortverlust wiederherzustellen, so erfolgt eine zusätzliche Verschlüsselung Ihrer Daten mit einem Masterkey oder einer vergleichbaren Lösung. Hierfür gibt es verschiedene Implementierungsmöglichkeiten, die die Stärke der Verschlüsselung massiv abschwächen können."[2]

[2] Hißen, Frank (16.10.2015): Verschlüsselung ist nicht gleich Verschlüsselung, Datenverschlüsselung in der Cloud, [online] https://www.tecchannel.de/a/verschluesselung-ist-nicht-gleich-verschluesselung,3281630,2 [abgerufen am 02.07.2022].

Anlage 3 Implementierung zu Aufgabenteil II.d

Würde vor jedem erneuten Anhängen jeder Buchstabe von `vorname` und `nachname` um den Wert eins im Alphabet nach „rechts“ geschoben, so würde im ersten Schritt aus der Liste `'(f r i t z b r a u s e)` die Liste `'(g s j u a c s b v t f)` entstehen und in einem nächsten Schritt dann die Liste `'(h t k v b d t c w u g)`.

Insgesamt erhält man so die folgende Schlüsselliste:

`'(f r i t z b r a u s e g s j u a c s b v t f h t k v b d t c w u g).`

Folgende Funktionen liefern ein solches Ergebnis:

(*Beachten Sie, dass lediglich Kleinbuchstaben verwendet werden. „97“ ist der ASCII-Code für den Buchstaben „a“.*)

```
1   import Data.Char
2
3   verschiebe []     ausgabe = reverse ausgabe
4   verschiebe (b:bs) ausgabe = verschiebe bs ((berechneNaechsten b) : ausgabe)
5
6   berechneNaechsten b = zahlInBuchstabe (mod ((buchstabeInZahl b) + 1) 26)
7
8   zahlInBuchstabe zahl = chr (97 + zahl)
9   buchstabeInZahl b    = (ord b) - 97
```

Aufgabe III: Sandsort (Scheme-Version) 50 BE

Schwerpunkt: Intelligente Suchverfahren

Das Spiel Sandsort der Firma Baustoffspiele erfreut sich großer Beliebtheit in allen Altersgruppen. Bei Sandsort geht es darum, verschiedenfarbigen Sand, der sich in Schichten in Gläsern befindet, durch sukzessives Umfüllen so zu sortieren, dass sich am Ende jede Sandfarbe gemeinsam in einem Glas befindet. Dabei darf Sand immer nur auf Sand derselben Farbe oder in ein leeres Glas umgefüllt werden. Außerdem passen in jedes Glas nur höchstens fünf Schichten Sand.
Im folgenden Beispiel darf also zunächst nur jede oberste Farbe in eines der leeren Gläser gefüllt werden.

Abbildung 1: Ausgangslage der Gläser vor dem Sortieren

III.a
- **Erläutern** und **begründen** Sie, warum bei Sandsort die Tiefensuche effizienter eine Lösung finden wird als die Breitensuche.
- **Stellen** Sie **dar**, welche Probleme beim Einsatz der klassischen Tiefensuche allgemein auftreten können.

(9 BE)

Eine mögliche Datenstruktur für die Verwaltung des aktuellen Stands ist die Verwendung von sechs Listen (vgl. Anlage 1). Diese können in einer Liste `glaeser` verwaltet werden.

III.b **Beschreiben** Sie den Aufbau dieser Datenstruktur und **erläutern** Sie, warum sie sich für die Lösung des Problems eignet.

(7 BE)

III.c **Untersuchen** Sie, welche zusätzlichen Bedingungen man berücksichtigen müsste, um mit der Tiefensuche sicherzustellen, dass eine Lösung des hier betrachteten Problems gefunden wird, sofern sie existiert.

(7 BE)

Eine zentrale Aufgabe im Verlauf des Lösungsprozesses ist es, zu überprüfen, ob das Umkippen von Sand von einem bestimmten Glas in ein anderes bestimmtes Glas zulässig ist:

```
(zugZulaessig? '(blau gruen rot) '(blau blau))       →     #t
(zugZulaessig? '(blau gruen rot) '(rot blau))        →     #f
(zugZulaessig? '(blau) '(blau rot rot rot rot))      →     #f
(zugZulaessig? '() '(rot blau gruen))                →     #f
```

III.d **Implementieren** Sie eine Funktion `(zugZulassig? von nach)`, mit Hilfe derer überprüft werden kann, ob eine Schicht Sand aus dem Glas `von` in das Glas `nach` gekippt werden darf.

(7 BE)

In Sandsort muss eine Schicht von Sand der gleichen Farbe immer vollständig in ein anderes Glas umgefüllt werden, auch, wenn sie in der hier vorgestellten Datenstruktur aus mehreren gleichfarbigen Schichten besteht.

III.e **Entwerfen** Sie eine mögliche Erweiterung des bisherigen Vorgehens, welche diese Einschränkung sicherstellt.
Hinweis: Hier wird keine Implementierung der Erweiterung erwartet.

(4 BE)

Die Funktion `fertig?` (siehe Anlage 2) ist Teil eines Lösungsversuchs. Als `glaeser` wird die Liste übergeben, die die Listen der aktuellen Belegung der Gläser enthält:

```
(fertig? '((rot rot) (rot gruen) (blau blau)))       →     #f
(fertig? '((rot rot) (gruen gruen) (blau)))          →     #t
```

III.f
- **Untersuchen** Sie, welche Aufgabe die Funktion `fertig?` erfüllt, und **stellen** Sie **dar**, wie dies erreicht wird.
- **Stellen** Sie **dar**, inwieweit es sich bei der Funktion `alleDieseFarbe?` um eine endrekursive Funktion handelt.

(11 BE)

In späteren Levels von Sandsort reicht es nicht mehr aus, dass überhaupt eine Lösung gefunden wird, sondern es soll auch die Lösung gefunden werden, für die am wenigsten Umfüllvorgänge notwendig sind.

III.g **Beurteilen** Sie, ob sich der Dijkstra-Algorithmus oder eine andere Optimierung bei diesen Levels für die Suche nach einer Lösung besser eignen würde.

(5 BE)

Anlagen zur Aufgabe III Sandsort (Scheme-Version)

Anlage 1 Datenstruktur zur Verwaltung des aktuellen Stands der Suche

```
1  (define startzustand '( (blau gruen gelb blau rot)
2                           (gelb lila rot gruen blau)
3                           (blau lila rot gelb rot)
4                           (lila gruen lila gruen gelb)
5                           ()
6                           ()
7                         )
```

Anlage 2[1]

```
1  define (alleDieseFarbe? glas farbe)
2    (cond
3      ((null? glas) #t)
4      ((equal? (first glas) farbe)
5        (alleDieseFarbe? (rest glas) farbe))
6      (else #f)))
7
8  (define (nurEineFarbe? glas)
9    (cond
10     ((null? glas) #t)
11     (else (alleDieseFarbe? (rest glas) (first glas)))))
12
13 (define (fertig? glaeser)
14   (cond
15     ((null? glaeser) #t)
16     ((nurEineFarbe? (first glaeser)) (fertig? (rest
   glaeser)))
17     (else #f)))
```

[1] Folgende Funktionen erfüllen jeweils dieselbe Aufgabe in Scheme: `first` und `car`; `rest` und `cdr`; `null?` und `empty?`

Aufgabe III: Sandsort (Haskell-Version) 50 BE

Schwerpunkt: Intelligente Suchverfahren

Das Spiel Sandsort der Firma Baustoffspiele erfreut sich großer Beliebtheit in allen Altersgruppen. Bei Sandsort geht es darum, verschiedenfarbigen Sand, der sich in Schichten in Gläsern befindet, durch sukzessives Umfüllen so zu sortieren, dass sich am Ende jede Sandfarbe gemeinsam in einem Glas befindet. Dabei darf Sand immer nur auf Sand derselben Farbe oder in ein leeres Glas umgefüllt werden. Außerdem passen in jedes Glas nur höchstens fünf Schichten Sand.
Im folgenden Beispiel darf also zunächst nur jede oberste Farbe in eines der leeren Gläser gefüllt werden.

Abbildung 1: Ausgangslage der Gläser vor dem Sortieren

III.a
- **Erläutern** und **begründen** Sie, warum bei Sandsort die Tiefensuche effizienter eine Lösung finden wird als die Breitensuche.
- **Stellen** Sie **dar**, welche Probleme beim Einsatz der klassischen Tiefensuche allgemein auftreten können.

(9 BE)

Eine mögliche Datenstruktur für die Verwaltung des aktuellen Stands ist die Verwendung von sechs Listen (vgl. Anlage 1). Diese können in einer Liste `glaeser` verwaltet werden.

III.b **Beschreiben** Sie den Aufbau dieser Datenstruktur und **erläutern** Sie, warum sie sich für die Lösung des Problems eignet.

(7 BE)

III.c **Untersuchen** Sie, welche zusätzlichen Bedingungen man berücksichtigen müsste, um mit der Tiefensuche sicherzustellen, dass eine Lösung des hier betrachteten Problems gefunden wird, sofern sie existiert.

(7 BE)

Eine zentrale Aufgabe im Verlauf des Lösungsprozesses ist es, zu überprüfen, ob das Umkippen von Sand von einem bestimmten Glas in ein anderes bestimmtes Glas zulässig ist:

```
zugZulaessig [Blau, Grün, Rot] [Blau,Blau]      → True
zugZulaessig [Blau, Grün, Rot] [Rot,Blau]       → False
zugZulaessig [Blau] [Blau,Rot,Rot,Rot,Rot]      → False
zugZulaessig [] [Rot,Blau,Grün]                 → False
```

III.d **Implementieren** Sie eine Funktion `zugZulassig von nach`, mit Hilfe derer überprüft werden kann, ob eine Schicht Sand aus dem Glas `von` in das Glas `nach` gekippt werden darf.

(7 BE)

In Sandsort muss eine Schicht von Sand der gleichen Farbe immer vollständig in ein anderes Glas umgefüllt werden, auch, wenn sie in der hier vorgestellten Datenstruktur aus mehreren gleichfarbigen Schichten besteht.

III.e **Entwerfen** Sie eine mögliche Erweiterung des bisherigen Vorgehens, welche diese Einschränkung sicherstellt.
Hinweis: Hier wird keine Implementierung der Erweiterung erwartet.

(4 BE)

Die Funktion `fertig` (siehe Anlage 2) ist Teil eines Lösungsversuchs. Als `glaeser` wird die Liste übergeben, die die Listen der aktuellen Belegung der Gläser enthält:

```
fertig [[Rot,Rot], [Rot,Grün], [Blau,Blau]]     → False
fertig [[Rot,Rot], [Grün,Grün], [Blau]]         → True
```

III.f
- **Untersuchen** Sie, welche Aufgabe die Funktion `fertig` erfüllt, und **stellen** Sie **dar**, wie dies erreicht wird.
- **Stellen** Sie **dar**, inwieweit es sich bei der Funktion `alleDieseFarbe` um eine endrekursive Funktion handelt.

(11 BE)

In späteren Levels von Sandsort reicht es nicht mehr aus, dass überhaupt eine Lösung gefunden wird, sondern es soll auch die Lösung gefunden werden, für die am wenigsten Umfüllvorgänge notwendig sind.

III.g **Beurteilen** Sie, ob sich der Dijkstra-Algorithmus oder eine andere Optimierung bei diesen Levels für die Suche nach einer Lösung besser eignen würde.

(5 BE)

Anlagen zur Aufgabe III Sandsort (Haskell-Version)

Anlage 1 Datenstruktur zur Verwaltung des aktuellen Stands der Suche

```
1  data Farbe = Blau | Grün | Gelb | Rot | Lila
2     deriving (Eq)
3
4  startzustand =
5    [ [Blau, Grün, Gelb, Blau, Rot],
6      [Gelb, Lila, Rot, Grün, Blau],
7      [Blau, Lila, Rot, Gelb, Rot],
8      [Lila, Grün, Lila, Grün, Gelb],
9      [],
10     []
11   ]
```

Der Übersichtlichkeit halber werden hier neue Datentypen für die Farben definiert. Alternativ könnte auch mit Strings gearbeitet werden, die Liste für das erste Glas wäre dann:
["Blau", "Grün", "Gelb", "Blau", "Rot"],
Dies ist für die sonstigen Programmierteile unerheblich.

Anlage 2

```
1  --alleDieseFarbe glas farbe
2  alleDieseFarbe []     _      = True
3  alleDieseFarbe (erste:rest) farbe
4    | (erste == farbe)         = alleDieseFarbe rest farbe
5    | otherwise                = False
6  --nurEineFarbe glas
7  nurEineFarbe []             = True
8  nurEineFarbe (erste:rest) = alleDieseFarbe rest erste
9  --fertig glaeser
10 fertig []                   = True
11 fertig (glas:glaeser)
12   | nurEineFarbe glas       = fertig glaeser
13   | otherwise               = False
```