ulm university universität ulm

1. Klausur zur Veranstaltung

# Einführung in die Informatik II - Vertiefung

und **Allgemeine Informatik 2**

im Sommersemester 2022

Prüfer: Dr. Jens Kohlmeyer — Fakultät Ingenieurwissenschaften, Informatik, Psychologie

**25.07.2022, 11 Uhr** — **Bearbeitungszeit: 90 min**

| Nachname: | Vorname: | Matrikelnummer: |
|---|---|---|
| Studiengang und Abschluss: | | Fachsemester: |

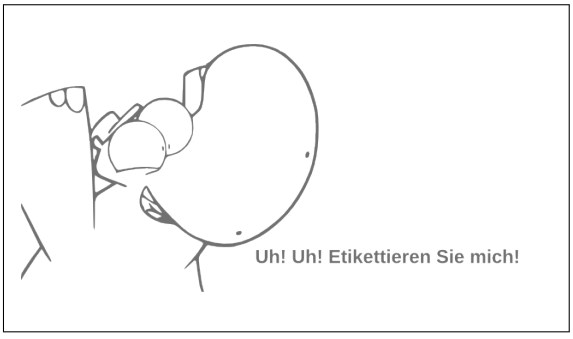
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

Datum, Unterschrift des Prüfungsteilnehmers

## Zur allgemeinen Beachtung:

- Füllen Sie das Deckblatt vollständig und korrekt aus.
- Lesen Sie sich zunächst die Klausur sorgfältig durch (sie besteht aus 14 Seiten).
- Bearbeiten Sie die Aufgaben direkt auf den Aufgabenblättern.
- Als Hilfsmittel ist das vorgegebene Cheatsheet im DIN-A4-Format zugelassen. Beide Seiten dürfen handschriftlich beschrieben sein.
- Aufgaben, welche nicht mit einem dokumentenechten Stift in den Farben blau oder schwarz bearbeitet worden sind, werden nicht bewertet.




Zusätzlich benötigtes Papier wird Ihnen von der Aufsicht zur Verfügung gestellt.

| *Punkteverteilung* | | | | | | |
|---|---|---|---|---|---|---|
| 1<br>von 12 | 2<br>von 12 | 3<br>von 12 | 4<br>von 12 | 5<br>von 12 | $\sum$<br>von 60 | **Note** |
| | | | | | | Korrektur |
| Einsichtnahme ohne Nachkorrektur ◯ | | | | Einsichtnahme mit Nachkorrektur ◯ | | |

## Aufgabe 1 - Verständnisfragen

**3 + 5 + 4 = 12 Punkte**

**a)** Erläutern Sie, **in ganzen Sätzen**, was unter der "bad-character"-Strategie im Boyer-Moore Algorithmus verstanden wird. Gehen Sie in Ihrer Erläuterung auf die verschiedenen Fälle, welche die Strategie abdeckt, mittels Beispielen ein.

**b)** Erläutern Sie, **in ganzen Sätzen**, das Prinzip, die Voraussetzungen und die Vorgehensweise des *LSD-Radixsort*.

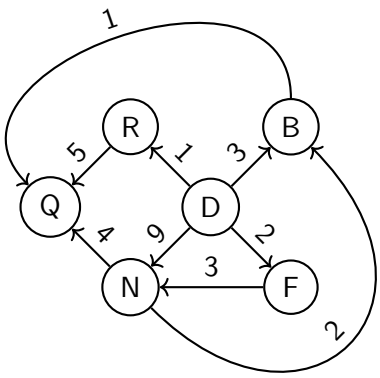
**c)** Nennen und Beschreiben Sie, **in ganzen Sätzen**, **zwei** Maßnahmen zur Qualitätssicherung von Softwaresystemen.

Diese Seite wurde für ein besseres Layout leer gelassen.

## Aufgabe 2 - Graphen

**2 + 3.5 + 3.5 + 3 = 12 Punkte**

Betrachten Sie folgenden gerichteten, gewichteten Graphen $G$:




**a)** Bewerten Sie folgenden Aussage bezüglich ihrer Richtigkeit und begründen Sie Ihre Antwort: "Jeder gerichtete azyklische Graph ist ein Baum."

**b)** Wenden Sie den Dijkstra-Algorithmus mit Startknoten $D$ auf $G$ an.

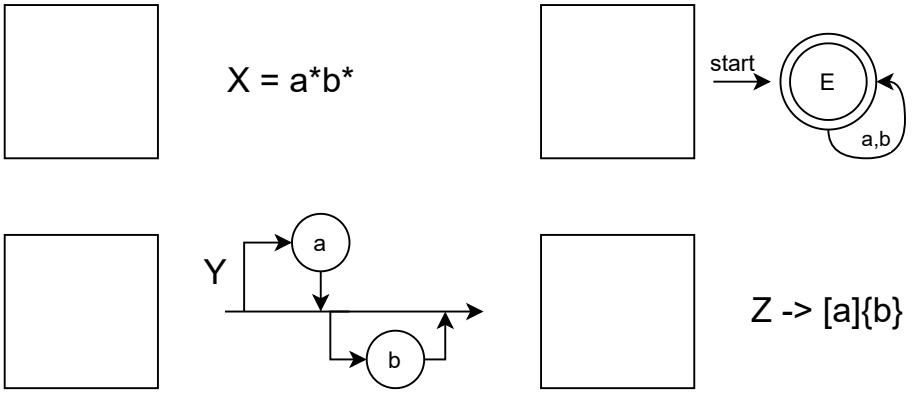
**c)** Geben Sie die Adjazenzliste für $G$ an.

**d)** Erläutern Sie eine Änderung an $G$ die Sie vornehmen müssten um den Graph nicht mehr topologisch sortierbar zu machen und erklären Sie warum dies der Fall ist.

## Aufgabe 3 - Formale Sprachen

**4 + 3 + 5 = 12 Punkte**

**a)** Im Folgenden sehen Sie einen regulären Ausdruck, einen endlichen Automaten, ein Syntaxdiagramm und eine EBNF. Markieren Sie, welche davon jeweils die gleiche Sprache erzeugen bzw. beschreiben, indem Sie in die Box eine Zahl eintragen. *Wären Sie zum Beispiel der Meinung, dass alle unterschiedliche Sprachen beschreiben bzw. erzeugen, dann tragen Sie in jede Box eine andere Zahl ein.*




**b)** Geben Sie einen **deterministischen** endlichen Automaten an, der alle positiven, geraden Dezimalzahlen, ohne führende Nullen akzeptiert.

**c)** Die Programmiersprache LISP zeichnet sich durch eine sehr simple Syntax aus die sich leicht in eine Grammatik und, daraus resultierend, leicht parsen lässt.
In dieser Aufgabe betrachten wir ein Subset valider LISP Ausdrücke, hier genannt $\text{LISP}^-$:
Jeder Ausdruck in $\text{LISP}^-$ beginnt mit einer öffnenden und endet mit einer schließenden Klammer. Ein $\text{LISP}^-$ Ausdruck ist eine Operation mit 2 bzw. 3 Argumenten in Präfixnotation (d.h der Name der Operation kommt vor den Operanden). Operationen bekommen immer entweder eine ganze Zahl $> 0$ oder weitere $\text{LISP}^-$ Ausdrücke als Operanden übergeben. Die erlaubten 2-Argument Operationen in $\text{LISP}^-$ sind: $+$, -, * und $/$. Die einzige erlaubte 3-Argument Operation in $\text{LISP}^-$ ist: *if*.
Definieren Sie eine Grammatik die alle gültigen $\text{LISP}^-$ Ausdrücke akzeptiert.
**Beispiel**: Folgender Ausdruck ist ein valider $\text{LISP}^-$ Ausdruck: $(* \ (+ \ 1 \ 2) \ (\mathit{IF} \ 0 \ (+ \ 1 \ 2) \ (- \ 900 \ 10)))$

## Aufgabe 4 - Java - Datenstrukturen **3 + 4 + 5 = 12 Punkte**

Betrachten Sie folgende Klassen und Beschreibung:

```java
public class Library {
    private X books;
}
```

```java
public class Book {
    public String title;
    protected Y pages;
}
```

```java
public class Page {
    public String content;
}
```

```java
public interface Readable<T> {
    public String read();
}
```

In einer Bibliothek (`Library`) werden Bücher nach Genre sortiert abgelegt. Genres werden direkt über ihren Namen identifiziert.
Bücher (`Books`) enthalten Seiten (`Pages`) in einer sortierten Reihenfolge, d.h. Seite 2 folgt auf Seite 1, Seite 3 folgt auf Seite 2 usw.

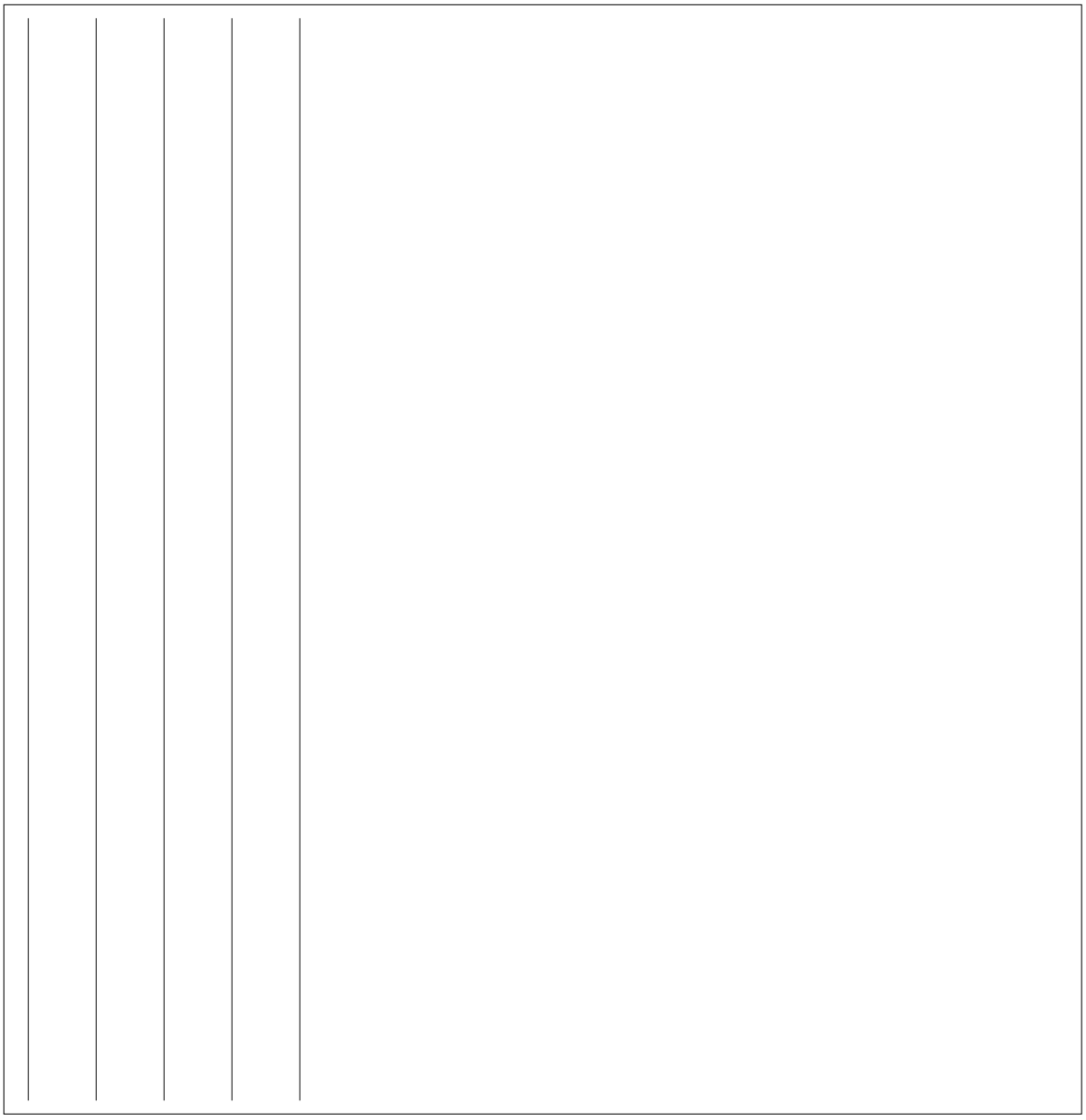
**a)** Geben Sie sinnvolle Typen für die Platzhalter `X` und `Y` an. Begründen Sie, warum Sie sich für diese Typen entschieden haben. Es ist Ihnen erlaubt selbst neue Typen zu definieren und zu verwenden. Geben Sie hierfür die komplette Klassendefinition der neu konzipierten Typen an.
Sie dürfen jeden Datentyp nur **einmal** verwenden.

**b)** Beschreiben Sie, wie, auf Basis Ihrer in **a)** gewählten Datentypen folgende Aufgabe zu lösen ist. Nummerieren Sie die einzelnen Schritte Ihrer Lösung so, dass eine klare Reihenfolge erkennbar ist:
*Finden Sie, ausgehend von einer Library, heraus, ob ein Book mit dem Titel "LISP für Anfänger" in den Büchern des Genre "Sachbücher" existiert.*
**Beispiel für die Darstellung** *Prüfen Sie, ob die Anzahl der Zeichen auf einer Page größer als 1000 ist:*
1. Länge von `content` mittels content.length() ermitteln
2. Vergleich der ermittelten Länge mit 1000, falls größer `true` zurückgeben, sonst `false`

**c)** Welche Änderungen müssen Sie an der Klasse `Book` vornehmen, so dass diese das Interface `Readable<Page>` implementiert wenn folgende Eigenschaften für das Interface definiert sind:

- Mit `Readable<T>` wird angegeben, dass der Inhalt von `T` nach und nach stückweise gelesen werden kann.
- `read` liefert bei jedem Aufruf weiteren, bis jetzt noch nicht gelesen Inhalt, von `T` in Form eines Strings zurück.
- Wie `read` die Daten intern aufteilt, um stückweises Lesen zu ermöglichen, ist beliebig, muss aber sicherstellen, dass niemals 2 mal der selbe Inhalt zurückgeliefert wird.
- Sollte kein weiterer Inhalt lesbar sein so muss `read` `null` zurückliefern.

Geben Sie hierfür die neue Definition der gesamten Klasse `Book` mit allen Attributen und Methoden an.



Diese Seite wurde für ein besseres Layout leer gelassen.

## Aufgabe 5 - Java - Familien

**3 + 4 + 5 = 12 Punkte**

Betrachten Sie folgende Klassen, die Familienrelationen darstellen:

```
1 public class Person {
2     boolean female;
3     String name;
4     Female mother;
5     Male father;
6     Person[] siblings;
7     Person[] kids;
8 }
```

```
1 public class Male extends Person {
2     public Male() {
3         this.female = false;
4     }
5 }
```

```
1 public class Female extends Person {
2     public Female() {
3         this.female = true;
4     }
5 }
```

```
1 public enum Direction {
2     UP, DOWN, RIGHT;
3 }
```

**a)** Implementieren Sie die Methode `isAncestor(Person p)` der Klasse `Person` welche prüft, ob eine übergebene Person `p` Vorfahre der aktuellen Person ist.

```
public boolean isAncestor(Person p) {
```

**b)** Implementieren Sie die Methode `parseDirections(String str)` der Klasse `Person`. Die Methode soll aus einem übergebenen String ein Array mit Richtungsangaben erzeugen. Dabei gelten die folgenden Regeln:

- Richtungsangaben sind mit dem Symbol ',' getrennt.
- Die Richtung `UP` wird mit dem Wort "Hoch" angegeben.
- Die Richtung `DOWN` wird mit dem Wort "Runter" angegeben.
- Die Richtung `RIGHT` wird mit dem Wort "Side" angegeben.
- Angaben von anderen Wörtern werden ignoriert.

**Hinweis**: Das zurückgegebene Array darf leere Felder enthalten.

```
public Direction[] parseDirection(String str) {
```

**c)** Implementieren Sie die Methode `traverseFamily(String str)` der Klasse `Person`. Die Methode soll, auf Basis eines Strings der der Formatierung aus Aufgabe b) entspricht durch den Stammbaum der Person navigieren und die Person am Ende der Traversierung zurückgegeben. Dabei gilt:

- UP navigiert zum Vater, insofern dieser existiert.
- DOWN navigiert zum ersten Kind (an der ersten Position im kids-Array), insofern dieses existiert.
- RIGHT navigiert zum letzten Bruder oder zur letzten Schwester (an der letzten Position im siblings-Array), insofern diese/r existiert.

**Hinweis**: Sie dürfen Ihre parse-Methode aus Aufgabe b) verwenden. Achten Sie auf den Aufbau des zurückgegebenen Arrays!

```
public Person traverseFamily(String str) {
```

Falls Sie noch Platz benötigen, so können Sie dieses Blatt benutzen. Weitere Blätter erhalten Sie von der Aufsicht. Machen Sie **eindeutig kenntlich**, welche Aufgaben hier bearbeitet bzw. fortgesetzt werden und **streichen** Sie den Lösungsversuch eindeutig durch, den sie nicht bewertet haben wollen.