ulm university universität ulm

2. Klausur zur Veranstaltung

Einführung in die Informatik II - Vertiefung

und **Allgemeine Informatik 2**

im Sommersemester 2022

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

06.10.2022, 11 Uhr — **Bearbeitungszeit: 90 min**

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

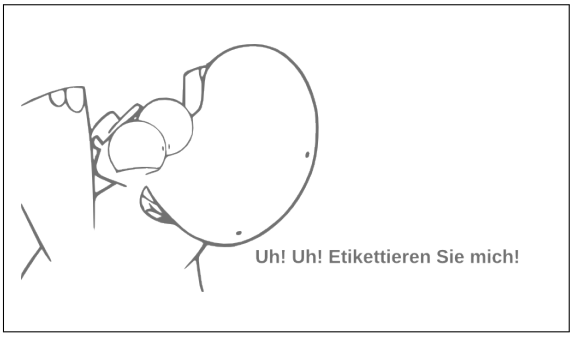
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

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 10 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 von 12	2 von 12	3 von 12	4 von 12	5 von 12	$\sum$ von 60	**Note**
						Korrektur
Einsichtnahme ohne Nachkorrektur ◯				Einsichtnahme mit Nachkorrektur ◯		

Aufgabe 1 - Verständnisfragen

3 + 6 + 3 = 12 Punkte

a) Erläutern Sie die Eigenschaften der Produktionsregeln, die für eine **Typ 3**-Grammatik gelten.

b) Erläutern Sie **in Stichpunkten** einen rekursiven Algorithmus, der eine *depth-first* Suche auf einem Graphen durchführt.

c) Begründen Sie warum folgende Aussage nicht allgemein gültig ist:
"Ein Algorithmus mit einer Laufzeit von $O(n)$ ist schneller als ein Algorithmus mit einer Laufzeit von $O(n^2)$"
Verwenden Sie ein illustrierendes Beispiel zur Unterstützung Ihrer Erklärung.

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 2 - Suchen

2.5 + 2.5 + 7 = 12 Punkte

a) Erklären Sie, warum bei einem Suchbaum nach dem Einfügen eines neuen Elements manchmal eine Umverteilung der Elemente vorgenommen werden sollte.

b) Erstellen Sie die delta1-Tabelle, welche für den Boyer-Moore-Algorithmus benötigt wird, für das Pattern *hanna*.

c) Suchen Sie mit dem Algorithmus von Boyer-Moore (BM) im unteren Text nach dem **ersten Vorkommen** des Patterns *hanna* aus Teilaufgabe **b)**. Füllen Sie für jedes Verschieben des Musters eine Zeile der nachstehenden Tabelle aus. Schreiben Sie in jede Zeile das gesamte Muster und **umkringeln** Sie alle Zeichen des Musters, die mit dem Text verglichen werden. Geben Sie für jeden Shift an, wie Sie diesen bestimmt haben. Notieren Sie dies **hinter** dem Muster auf dem Sie den Shift berechnet haben.
Hinweis: Sie benötigen nicht alle Zeilen der Tabelle.

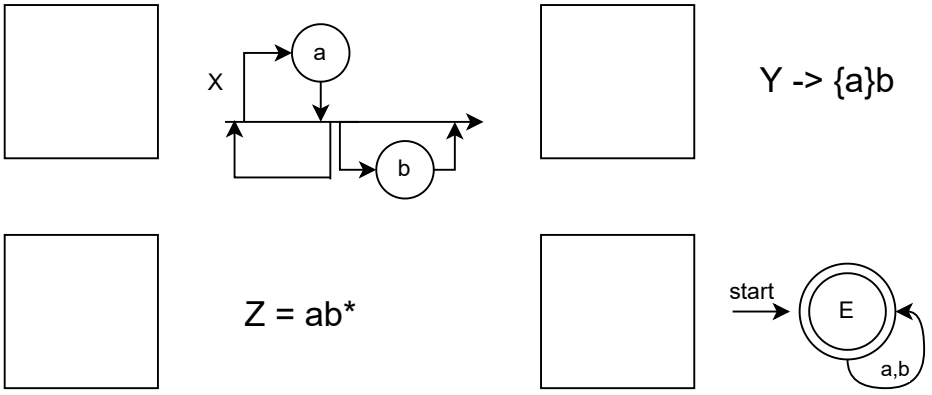
h	a	n	h	a	n	a	h	a	n	n	a

Diese Seite wurde für ein besseres Layout leer gelassen.

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 ungeraden natürlichen Zahlen größer 4 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 LISP^-:
Jeder Ausdruck in LISP^- beginnt mit einer öffnenden und endet mit einer schließenden Klammer. Ein 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 LISP^- Ausdrücke als Operanden übergeben. Die erlaubten 2-Argument Operationen in LISP^- sind: $+$, -, * und $/$. Die einzige erlaubte 3-Argument Operation in LISP^- ist: *if*.
Definieren Sie eine Grammatik die alle gültigen LISP^- Ausdrücke akzeptiert.
Beispiel: Folgender Ausdruck ist ein valider LISP^- Ausdruck: $(* \ (+ \ 1 \ 2) \ (\mathit{IF} \ 0 \ (+ \ 1 \ 2) \ (- \ 900 \ 10)))$

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

Betrachten Sie folgendes Interface und die Beschreibung:

```
1 public interface BetterSet<T> {
2     public BetterSet<T> add(T elem);
3     public BetterSet<T> remove(T elem);
4     public BetterSet<T> union(BetterSet<T> set);
5 }
```

`BetterSet` definiert ein `Set` das Methodchaning unterstützt. Wie bei einem normalen `Set` werden Elemente nur hinzugefügt wenn sie noch nicht im Set existieren.

a) Definieren Sie eine Klasse `MySet<T>` welche das Interface `BetterSet<T>` implementiert. Implementieren Sie hierfür alle notwendigen Methoden **mit der Ausnahme von** `intersect`. Wählen Sie als interne Datenstruktur eine `LinkedList<T>`.

b) Erklären Sie einen Vorteil und einen Nachteil den `BetterSet` gegenüber eines "normalen" Sets hat. Demonstrieren Sie beides an je einem Beispiel.

c) Gehen Sie davon aus, dass eine Implementierung von `BetterSetk<T>` als interne Datenstruktur ein Array verwendet. Beschreiben Sie, wie, auf Basis dieser Vorgabe, die Methode `union` zu implementieren wäre. Die Methode vereinigt die Inhalte der zwei involvierten Mengen in der aufgerufenen Menge und verwirft dabei doppelte Einträge.
Beispiel für die Darstellung:

i. Lese erstes Element aus interner Datenstruktur aus
ii. Prüfe ob es `null` ist
iii. ...

Aufgabe 5 - Java - Bekannte

4 + 2 + 6 = 12 Punkte

Betrachten Sie folgend Klassen, welche Bekanntenrelationen darstellt:

```
1 public class Person {
2     String name;
3     Set<Person> knows;
4 }
```

a) Implementieren Sie die Methode `mingle(Person[] persons)` der Klasse `Person`, welche alle Bekannten der übergebenen Personen `persons` zu den Bekannten der Person hinzufügt. Dies soll auch umgekehrt stattfinden.

```
public void mingle(Person[] persons) {
```

b) Nennen und erläutern Sie **2** Gründe, warum für das Attribut `knows` ein `Set` geeigneter ist als ein Array.

c) Implementieren Sie die Methode `transitivelyKnows(Person p)` der Klasse `Person` welche prüft, ob die Person eine übergebene Person `p` kennt, oder ob eine der Bekannten sie kennt.
Hinweis: *Achten Sie auf Zyklen in den Bekanntenkreisen!*
Hinweis: *Objektidentität genügt als Vergleich.*

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