| Prüfungsteilnehmer | Prüfungstermin | Einzelprüfungsnummer |
|---|---|---|

**Kennzahl:** _____

**Kennwort:** _____

**Arbeitsplatz-Nr.:** _____

# Herbst 2010

# 66114

## Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

## — Prüfungsaufgaben —

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Datenbank- und Betriebssysteme**

Anzahl der gestellten Themen (Aufgaben): **4 Aufgaben, von denen zwei gemäß untenstehender Auswahlregel zu bearbeiten sind!**

Anzahl der Druckseiten dieser Vorlage: **21**

Zu den zwei Themenschwerpunkten A (Datenbanksysteme) und B (Betriebssysteme) ist jeweils entweder die Aufgabe 1 oder 2 zu wählen. Auf der Vorderseite des Kopfbogens sind im Feld „gewähltes Thema: Nr." die Nummern der beiden gewählten Aufgaben anzugeben (z. B. A2, B1)!

**Bitte wenden!**

# Themenschwerpunkt A
# (Datenbanksysteme)

## Teilaufgabe 1:

### 1. Entity-Relationship-Modell

Ein Handelsunternehmen möchte seine Struktur verbessern und ein Datenbanksystem zur Verwaltung seiner Filialen, angebotenen Waren und Kunden erstellen.

Die Basis dieses Systems bilden die Filialen des Unternehmens. Jede Filiale ist eindeutig durch ihre Filialnummer gekennzeichnet und befindet sich in einer Stadt. Außerdem hat jede Filiale einen Filialleiter.

Zu jeder Filiale gehört genau ein Lager mit einer eindeutigen Lagernummer und ebenfalls einem Leiter. Jedes Lager verfügt über eine bestimmte Menge an verschiedenen Waren. Jede Ware kann in mehreren Lagern vorrätig sein und ist über eine Nummer, einen Namen und einen Preis gekennzeichnet.

Ein Kunde kann in einer Filiale des Unternehmens Bestellungen aufgeben. Der Kunde hat eine Kundennummer, einen Namen und eine Adresse. Eine Bestellung enthält dabei jeweils einen Warenartikel, dessen gewünschte Menge und das Datum, an dem die Bestellung abgeholt wird.

(a) Erstellen Sie ein Entity-Relationship-Diagramm für obige Datenbank.

(b) Setzen Sie das in Teilaufgabe a) erstellte Entity-Relationship-Diagramm in ein Relationenschema um. Relationships sollen mit einer möglichst geringen Anzahl von Relationen realisiert werden. Dabei sind unnötige Redundanzen zu vermeiden. Ein Relationenschema ist in folgender Form anzugeben: `Relation (Attribut1, Attribut2, ...)`. Schlüsselattribute sind dabei zu unterstreichen.
Achten Sie bei der Wahl des Schlüssels auf Eindeutigkeit und Minimalität.

**Fortsetzung nächste Seite!**

## 2. Normalisierung

Gegeben sei folgende Datenbank für Wareneingänge eines Warenlagers. Die Primärschlüssel-Attribute sind unterstrichen.

| ZulieferungsNr | ArtikelNr | Datum | Artikelname | Menge |
|---|---|---|---|---|
| 1 | 1 | 01.01.2009 | Handschuhe | 5 |
| 1 | 2 | 01.01.2009 | Mütze | 10 |
| 2 | 3 | 05.01.2009 | Schal | 2 |
| 2 | 1 | 05.01.2009 | Handschuhe | 8 |
| 3 | 4 | 06.01.2009 | Jacke | 2 |

(a) Erläutern Sie, inwiefern obiges Schema die 3. Normalform verletzt.

(b) Geben Sie für obige Datenbank alle vollen funktionalen Abhängigkeiten (einschließlich der transitiven) an.

(c) Überführen Sie das obige Relationenschema in die 3. Normalform.
Erläutern Sie die dazu durchzuführenden Schritte jeweils kurz.

## 3. SQL

Gegeben sei das folgende Relationenschema:

```
Fahrzeug(MNR:int(3), FZGNR:char(12), Baujahr:int(4),
         KMStand:int(5),Preis:int(5))
Modell(MNR:int(3), HNR:int(3), Typ:char(20), Neupreis:int(5),
       ps:int(3))
Hersteller(HNR:int(3), Name:char(20))
```

Dabei sind die Schlüsselattribute jeweils unterstrichen und zusätzlich für alle Attribute die Typen angegeben. Formulieren Sie die folgenden Anfragen bzw. Anweisungen in SQL.

(a) Geben Sie die Anweisungen in SQL-DDL an, die notwendig sind, um die Relationen „Fahrzeug", „Modell" und „Hersteller" zu erzeugen. Achten Sie dabei darauf, die Primärschlüssel der Relationen zu kennzeichnen.

(b) Bestimmen Sie die Typen aller Modelle des Herstellers mit Namen BMW.

(c) Bestimmen Sie den Mindestpreis, bezogen auf das Attribut „Preis", der Fahrzeuge eines jeden Herstellers.

(d) Bestimmen Sie die Namen der Hersteller, für die von jedem ihrer Modelle mindestens ein Fahrzeug in der Datenbank gespeichert ist.

(e) Bestimmen Sie die Namen aller Hersteller, von denen mindestens fünf Fahrzeuge eines beliebigen Modells in der Datenbank gespeichert sind.

**Fortsetzung nächste Seite!**

**Teilaufgabe 2:**

## 1. Datenbankanfragen in SQL

Wir betrachten die Datenbank FUSSBALL-EUROPAMEISTERSCHAFT mit den folgenden Tabellen:

- Spieler(Rückennummer, Land, Name, Vorname, Länderspiele, Verein, Land_V):
  Spielerinformation, einschließlich des Heimatlandes (Land), der Anzahl der Länderspiele, dem Verein und dem Land des Vereins (Land_V)

- Nationen(Land, Hauptstadt):
  Informationen zu den teilnehmenden Nationen

- nimmt_teil_an(Rückennummer, Land, Spiel_ID):
  Beteiligung eines Spielers an Spielen

- Spiele(Spiel_ID, Land_Heim, Land_Gast, Datum, Stadion_ID, Tore_Heim, Tore_Gast):
  Informationen zu den Spielen, wobei Land_Heim und Land_Gast die beiden an einem Spiel beteiligten Nationen sind

- Stadion(Stadion_ID, Stadt, Land, Zuschauerkapazität):
  Informationen zu den Stadien

Erstellen Sie in SQL folgende Anfragen:

(a) Bestimmen Sie alle teilnehmenden Spieler, die beim "FC Bayern München" (Verein) spielen.

(b) Bestimmen Sie alle Spiele, die im Heimatland von einer der beiden beteiligten Nationen stattfinden.

(c) An wie vielen Spielen nimmt "Bastian Schweinsteiger" teil?

(d) Bestimmen Sie für jede Nation den Spieler mit den meisten Länderspielen. Dazu können Sie eine Unteranfrage oder eine Sicht (View) verwenden, die in der Hauptanfrage aufgerufen wird.

**Fortsetzung nächste Seite!**

## 2. Datenmodellierung und Datenbankaufbau

Wir betrachten die Datenbank FUSSBALL-EUROPAMEISTERSCHAFT aus Aufgabe 1.

(a) Definieren Sie die Relationenschemata „Spieler" und „nimmt_teil_an" in SQL mittels CREATE TABLE und geben Sie dabei geeignete Schlüssel- und Fremdschlüsselbedingungen an. Nehmen Sie dabei an, dass die anderen Tabellen schon erzeugt sind.

(b) Fügen Sie in SQL mittels INSERT in jede der beiden Relationen „Spieler" und „nimmt_teil_an" jeweils mindestens ein Tupel ein, so dass die natürlichen Schlüssel- und Fremdschlüssel bedingungen aus Teil a) erfüllt sind. Eventuelle weitere Fremdschlüsselbedingungen auf andere Tabellen müssen nicht berücksichtigt werden.

(c) Geben Sie ein ER-Diagramm mit Funktionalitätsbedingungen für das Datenbankschema an, in dem möglichst viele Relationen als Relationships modelliert sind.

## 3. Funktionale Abhängigkeiten

Gegeben sei das Relationenschema $R = (U, F)$ mit der Attributmenge

$$U = \{A,B,C,D,E,G\}$$

und der folgenden Menge $F$ von funktionalen Abhängigkeiten:

$$F = \{ A \rightarrow B,\ AB \rightarrow CD,\ CD \rightarrow E\}.$$

a) Bestimmen Sie die Attributhüllen $\{A\}_F^+$ und $\{C,D\}_F^+$.

b) Erklären Sie, warum die funktionale Abhängigkeit $CD \rightarrow E$ zu Update-Anomalien (d. h. beim Einfügen, Löschen und Ändern) führt.

c) Bestimmen Sie alle Schlüssel von $R$.

d) Bestimmen Sie eine Basis und eine 3NF-Zerlegung von $R$.

e) Zerlegen Sie $R$ bezüglich der BCNF-verletzenden funktionalen Abhängigkeit $CD \rightarrow E$.

# Themenschwerpunkt B
# (Betriebssysteme)

## Teilaufgabe 1:

### 1. Verklemmungen

a) Nennen Sie alle Bedingungen für das Auftreten von Verklemmungen (Deadlocks).

b) Gegeben sei folgender Systemzustand:

| | maximum | | | |
|---|---|---|---|---|
| | A | B | C | D |
| P0 | 2 | 5 | 8 | 7 |
| P1 | 0 | 3 | 3 | 1 |
| P2 | 1 | 2 | 2 | 2 |
| P3 | 8 | 5 | 2 | 2 |
| P4 | 5 | 3 | 3 | 3 |

| | allocation | | | |
|---|---|---|---|---|
| | A | B | C | D |
| P0 | 0 | 0 | 8 | 4 |
| P1 | 0 | 1 | 0 | 0 |
| P2 | 0 | 2 | 0 | 1 |
| P3 | 2 | 2 | 1 | 1 |
| P4 | 5 | 0 | 0 | 3 |

| available | | | |
|---|---|---|---|
| A | B | C | D |
| 1 | 0 | 3 | 1 |

Überprüfen Sie mit Hilfe des Bankier-Algorithmus, ob sich das System in einem sicheren Zustand befindet. Geben Sie im positiven Fall eine sichere Prozessfolge an und im negativen Fall den Zustand, in dem keinem weiteren Prozess die benötigten Betriebsmittel zugeteilt werden können.

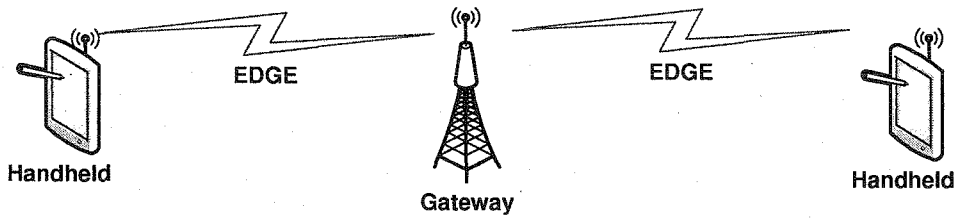
c) Wann kann es in einem System, in dem von Beginn an ein Verklemmungsvermeidungsalgorithmus läuft, trotzdem zu einer Verklemmung (einem Deadlock) kommen? Begründen Sie kurz Ihre Antwort.

**Fortsetzung nächste Seite!**

## 2. Flusskontrolle

Ein Student möchte mit seinem Handheld ein Musikstück auf das Handheld seines Freundes schicken. Hierfür verbindet er sich über eine EDGE-Datenverbindung mit einem Gateway, welches seinerseits eine EDGE-Datenverbindung mit dem Handheld des Freundes aufbaut. Die Datenverbindung besteht für die Dauer des Musikstückaustauschs.

Die folgende Abbildung illustriert diesen Zusammenhang:




Nehmen Sie an, die EDGE-Verbindung hat eine Bandbreite von 470 kBit/s und die Datenübertragung von einem Handheld zum anderen Handheld dauert 250 ms. Das Gateway benötigt 20 ms für die Umsetzung eines Frames zwischen den EDGE-Verbindungen.

Bearbeiten Sie nun die folgenden Teilaufgaben:

a) Nehmen Sie an, dass zur Ende-zu-Ende Flusskontrolle ein Sliding-Window-Verfahren eingesetzt wird. Die Framegröße ist jeweils auf genau 1024 Bytes normiert. Welche Fenstergröße (Anzahl der speicherbaren Pakete) ist mindestens erforderlich, um für die EDGE-Verbindungen eine Auslastung von 80% zu erreichen?

b) Betrachten Sie nun die Anwendung der Slow-Start Methode, wobei keine Überlastung vorliegt. Das Empfangsfenster am Gateway ist 40 kByte und die maximale Framegröße beträgt 2 kByte. Wie lange dauert es mindestens, bis das erste volle Fenster vom Handheld gesendet werden kann?

**Fortsetzung nächste Seite!**

## 3. ISO/OSI-Modell

a) Geben Sie die Schichten des ISO/OSI-Modells an. Achten Sie auf die richtige Reihenfolge der Schichten.

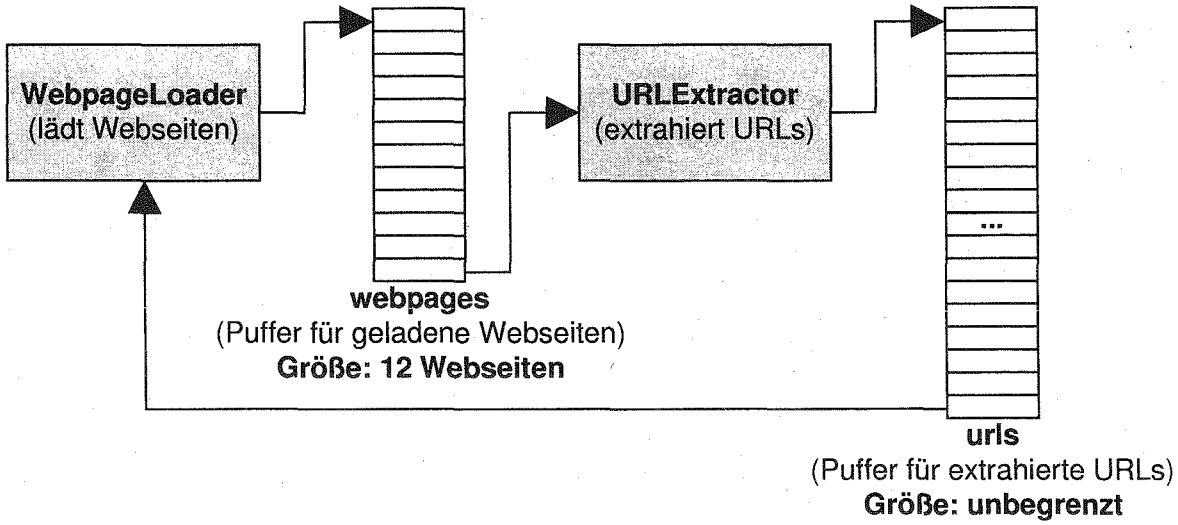
b) Führen Sie kurz (1-2 Sätze pro Schichtübergang) aus, wie eine HTTP-Anfrage im ISO-OSI Modell abgehandelt wird. Beginnen Sie hierbei bei der obersten Schicht am Web-Client („Klick auf den Link im Browser") bis hinunter zur Leitung und wieder hoch bei dem aufgerufenen Web-Server.

**Fortsetzung nächste Seite!**

## 4. Producer-Consumer-Problem

Ein Webcrawler oder Spider ist ein Computerprogramm, das automatisch das World Wide Web durchsucht und Webseiten analysiert. Webcrawler werden vor allem von Suchmaschinen eingesetzt. Wie beim Internetsurfen gelangt ein Webcrawler über Hyperlinks von einer Webseite zu weiteren URLs. Dabei werden alle aufgefundenen Adressen gespeichert und der Reihe nach besucht.

In dieser Aufgabe wird ein vereinfachter Webcrawler betrachtet. Er besteht aus zwei Komponenten. Der `WebpageLoader` entnimmt eine URL aus dem Puffer `urls`, lädt die Webseite dieser URL und legt sie im Puffer `webpages` ab. Der `URLExtractor` entfernt eine Webseite aus dem Puffer `webpages`, extrahiert alle URLs, die in dieser Webseite vorkommen, und speichert diese im Puffer `urls`. Dabei sei der Puffer für Webseiten, `webpages`, auf zwölf Elemente beschränkt. Der Puffer für URLs, `urls`, habe dagegen keine Begrenzung.




Die beiden Komponenten, der `WebpageLoader` und der `URLExtractor`, sollen in eigenen Threads laufen. Außerdem soll es möglich sein, dass mehrere Instanzen der Klassen `WebpageLoader` bzw. `URLExtractor` parallel arbeiten. Dafür ist eine adäquate Synchronisation des Zugriffs auf die beiden Puffer, `webpages` und `urls`, erforderlich.

Zur Synchronisation stehen Ihnen binäre Semaphore (`BinarySemaphore`) und zählende Semaphore (`CountingSemaphore`) zur Verfügung:

```
public interface Semaphore {
    public void wait();
    public void signal();
}

public class BinarySemaphore implements Semaphore {
    public BinarySemaphore() {
        [...]
    }
    [...]
}
```

**Fortsetzung nächste Seite!**

```
public class CountingSemaphore implements Semaphore {
    public CountingSemaphore(int initialCount){
        [...]
    }
    [...]
}
```

Vervollständigen Sie die Codeschablonen in den Teilaufgaben a), b) und c), indem Sie die Kommentare der Form `// Ergänzung X` durch Code für die Initialisierung der verwendeten Semaphore bzw. für die Synchronisation der Threads der Klassen `WebpageLoader` und `LinkExtractor` ersetzen. Sie können an den mit `// Ergänzung X` gekennzeichneten Stellen beliebig viele Codezeilen oder auch keine einfügen. Schreiben Sie Ihre Lösung in folgender Form auf:

| Ergänzung | Code |
|---|---|
| 1 | `new BinarySemaphore()` |
| 2 | ... |
| 3 | ... |
| ... | ... |

Ihre Lösung soll einen möglichst hohen Grad an Parallelität ermöglichen.

a) Vervollständigen Sie die folgenden Initialisierungen der Semaphore.

```
// Semaphore für Zugriffsschutz auf den Puffer für Webseiten
Semaphore webpagesAccess = // Ergänzung 1

// Semaphore, um bei leerem Puffer auf eine Webseite zu warten
Semaphore webpagesCounter = // Ergänzung 2

// Semaphore, um zu warten, wenn der Webseitenpuffer voll ist
Semaphore webpagesBuffer = // Ergänzung 3

// Semaphore für den Zugriffsschutz auf den Puffer für URLs
Semaphore urlsAccess = // Ergänzung 4

// Semaphore, um bei leerem Puffer auf eine URL zu warten
Semaphore urlsCounter = // Ergänzung 5
```

b) Ergänzen Sie die folgende Vorlage für die Klasse `WebpageLoader`.

```
public WebpageLoader extends Thread {
    public void run() {
        while (active) {
            load();
        }
    }
    private void load() {
        // Ergänzung 6
        URL url = urls.removeFirst();
        // Ergänzung 7
        Webpage webpage = loadWebpage(url);
        // Ergänzung 8
        webpages.add(webpage);
```

**Fortsetzung nächste Seite!**

```
            // Ergänzung 9
        }
        [...]
}
```

c) Ergänzen Sie die folgende Vorlage für die Klasse `URLExtractor`.

```
public URLExtractor extends Thread {
    public void run() {
        while (active) {
            load();
        }
    }
    private void load() {
        // Ergänzung 10
        Webpage webpage = webpages.removeFirst();
        // Ergänzung 11
        List<URL> newURLs = extractURLs(webpage);
        for (URL newURL: newURLs) {
            // Ergänzung 12
            urls.add(newURL);
            // Ergänzung 13
        }
    }
    [...]
}
```

d) Der Puffer für Webseiten kann bis zu 12 Webseiten fassen. Die Größe des Puffers für extrahierte URLs ist dagegen bisher nicht begrenzt. Was kann passieren, wenn man die Größe dieses Puffers ebenfalls beschränkt (z.B. auf 100 URLs)?

**Fortsetzung nächste Seite!**

## 5. Maschinennahe Programmierung

Betrachten Sie folgende stark vereinfachte Maschine:

- Es gibt 8 Register R1 - R8 mit je 1 Byte Breite.
- Einziges Datenformat ist eine 1 Byte lange Ganzzahl.
- Folgende Befehle sind verfügbar:

| **Befehl** | **Pseudocode** | **Erläuterung** |
|---|---|---|
| `MOV Rx, Ry` | `Rx := Ry` | Kopiert den Inhalt von Register Ry in Register Rx. |
| `MOV Rx, i` | `Rx := i` | Lädt die Zahl i in Register Rx. Die Zahl i kann auch negativ sein. |
| `MOV Rx, [Ry]` | `Rx := Hauptspeicher[Ry]` | Lädt den Wert der Hauptspeicheradresse, die in Register Ry steht, in Register Rx. |
| `MOV [Rx], Ry` | `Hauptspeicher[Rx] := Ry` | Kopiert den Wert von Ry in den Hauptspeicher an die Adresse, die in Rx steht. |
| `ADD Rx, Ry, Rz` | `Rx := Ry + Rz` | Weist dem Register Rx die Summe der Werte der Register Ry und Rz zu. |
| `ADD Rx, Ry, i` | `Rx := Ry + i` | Weist dem Register Rx die Summe des Wertes des Registers Ry und der Zahl i zu. Die Zahl i kann auch negativ sein. |
| `marke:` | `marke:` | Definiert eine Marke, zu der gesprungen werden kann. |
| `JMP marke` | `goto marke` | Springt zu einer definierten Marke *marke*. |
| `JZ Rx, marke` | `if Rx = 0 then goto marke` | Springt zu einer Marke *marke*, wenn das Register Rx den Wert 0 enthält. |
| `JLE Rx, Ry, marke` | `if Rx <= Ry then goto marke` | Springt zu einer Marke *marke*, wenn der Wert des Registers Rx kleiner oder gleich dem Wert des Registers Ry ist. |
| `JG Rx, Ry, marke` | `if Rx > Ry then goto marke` | Springt zu einer Marke *marke*, wenn der Wert des Registers Rx größer als der Wert des Registers Ry ist. |

- Jede Zeile eines Programms enthält genau einen Befehl.
- Jede Zeile kann am Anfang eine Marke gefolgt von einem Doppelpunkt enthalten, um als Sprungziel zu dienen.
- Beispiel für ein Programm:

```
        MOV R1, 100
        MOV R2, [R1]
        MOV R3, 3
        JG R2, R3, mind4
        MOV R2, 4
mind4:  MOV [R1], R2
```

**Fortsetzung nächste Seite!**

In dieser Aufgabe werden Sie in vier Schritten den Bubblesort-Sortieralgorithmus in Assembler programmieren.

a) Schreiben Sie ein Assemblerprogramm, welches das Datum von der Hauptspeicheradresse R1 + R4 in Register R5 und das Datum von der Hauptspeicheradresse R1 + R4 + 1 in Register R6 lädt. Die Werte von R1 und R4 dürfen nicht verändert werden. Verwenden Sie R7 als Hilfsregister für die Adressberechnung.

Pseudocode:

```
R5 := Hauptspeicher[R1 + R4]
R6 := Hauptspeicher[R1 + R4 + 1]
```

b) Schreiben Sie ein Assemblerprogramm, welches den Wert von R5 an Adresse R1 + R4 in den Hauptspeicher schreibt und den Wert von R6 an Adresse R1 + R4 + 1. Die Werte von R1 und R4 dürfen nicht verändert werden. Verwenden Sie R7 als Hilfsregister für die Adressberechnung.

Pseudocode:

```
Hauptspeicher[R1 + R4] := R5
Hauptspeicher[R1 + R4 + 1] := R6
```

c) Schreiben Sie ein Assemblerprogramm, welches, falls der Wert von R5 größer ist als der Wert von R6, die Werte der Register R5 und R6 vertauscht und R3 auf 1 setzt. Verwenden Sie R7 als Hilfsregister für die Vertauschung.

Pseudocode:

```
if (R5 > R6) then
    swap(R5, R6)
    R3 := 1
endif
```

d) Schreiben Sie ein Assemblerprogramm, welches in einer Schleife den Wert von R4 von 0 bis zum Wert von R2 - 2 hochzählt.
In der Schleife wird der Code aus (a), (c) und (b) ausgeführt. Kopieren Sie dabei nicht den bereits geschriebenen Code, sondern verwenden die Platzhalter (a), (c) und (b).

Pseudocode:

```
for R4 := 0 to R2 - 2 do
  (a)
  (c)
  (b)
endfor
```

**Fortsetzung nächste Seite!**

Den geschriebenen Assemblercode können wir nun folgendermaßen zum Bubblesort-Algorithmus vervollständigen.

Pseudocode:

```
R1 := <Startadresse des Arrays im Hauptspeicher>
R2 := <Länge des Arrays>
R3 := 1
while R2 != 0 and R3 != 0 do
    R3 := 0
    (d)
    R2 := R2 - 1
endwhile
```

Assemblerprogramm:

```
          MOV R1, <Startadresse des Arrays im Hauptspeicher>
          MOV R2, <Länge des Arrays>
          MOV R3, 1
while:    JZ R2, endwhile
          JZ R3, endwhile
          MOV R3, 0
          (d)
          ADD R2, R2, -1
          JMP while
endwhile: MOV R7, 0
```

**Fortsetzung nächste Seite!**

**Teilaufgabe 2:**

## 1. Speichermanagement

### a) Virtueller Speicher - Seiten und Kacheln

Ein 32 *Bit*-System habe einen 1 *Gigabyte* großen physischen Speicher. Eine Kachel beziehungsweise Seite habe eine Größe von je 4 *Kilobyte*.

i) In wieviele Kacheln ist der physische Speicher unterteilt?

ii) In wieviele Seiten ist der virtuelle Speicher unterteilt?

### b) Virtueller Speicher - Seitenkacheltabellen

Betrachtet wird wiederum dasselbe 32 *Bit*-System mit entsprechend großem virtuellen Adressraum und einer Seitengröße von 4 *Kilobyte*.

i) Wie groß ist die Seitenkacheltabelle für jeden Prozess, wenn pro Eintrag 8 *Byte* benötigt werden?

Um Speicherplatz zu sparen, soll eine zweistufige Hierarchie von Seitenkacheltabellen eingesetzt werden. Für das *Page Directory* werden nun 7 *Bit* verwendet.

ii) Wieviele Einträge hat dann jede Seitenkacheltabelle der zweiten Stufe?

iii) Wieviel Speicher wird für die Verwaltung des virtuellen Adressraums *mindestens* benötigt, das heißt wenn von einem Prozess nur eine einzige Seite belegt würde? Die Größe eines jeden Eintrags sei wiederum 8 *Byte*.

### c) Seitenersetzung - Optimale Strategie

Für die notwendige Ersetzung von Seiten in den Kacheln des physischen Speichers gibt es verschiedene Strategien, so zum Beispiel *First In First Out*, *Least Frequently Used* oder *Least Recently Used* etc. Was wäre die optimale Strategie und findet diese in der Praxis ihren Einsatz?

### d) Seitenersetzung - Second Chance-Strategie

Außer den vorgenannten Strategien zur Seitenersetzung gibt es noch die sogenannte *Second Chance*-Strategie. Das Betriebssystem macht sich hierbei diejenigen zwei Bits eines jeden Seitenkacheltabellen-Eintrags zunutze, die angeben, ob lesend (*Read-Bit*) oder schreibend (*Dirty-Bit*) auf eine Seite zugegriffen wurde, um das Working Set der Seiten eines Prozesses abzuschätzen.

Während letzteres Bit benötigt wird, um zu bestimmen, ob eine Seite auf den Hintergrundspeicher geschrieben werden muss, bevor sie ausgetauscht werden kann, wird aufgrund des ersteren entschieden, ob eine Seite eine zweite Chance bekommen und somit vorerst noch nicht ersetzt werden soll.

Im Folgenden wird ein System mit 7 Seiten und 4 Kacheln betrachtet. Beim Einlagern sowie bei Zugriffen auf eine Seite wird deren *Read-Bit* gesetzt, im Falle einer zweiten Chance aber gelöscht.

Fortsetzung nächste Seite!

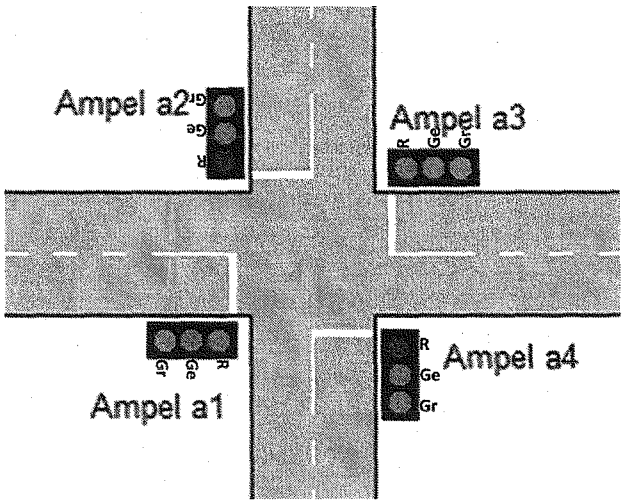
i) Zeigen Sie den Verlauf der Second Chance-Strategie anhand der folgenden *Lese*-Zugriffe in der angegebenen Reihenfolge. Notieren Sie das *Read-Bit* als Superskript, so zum Beispiel als $2^1$, falls das Bit für die Seite 2 gesetzt ist, und als $2^0$, falls dies nicht der Fall sein sollte.

| | Kacheln | | | | Page Queue | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Zugriff | $f_1$ | $f_2$ | $f_3$ | $f_4$ | 1 | 2 | 3 | 4 | Pagefault |
| 1 | 1 | - | - | - | $1^1$ | - | - | - | 1 |
| 3 | 1 | 3 | - | - | $3^1$ | $1^1$ | - | - | 1 |
| 7 | 1 | 3 | 7 | - | $7^1$ | $3^1$ | $1^1$ | - | 1 |
| 4 | | | | | | | | | |
| 3 | | | | | | | | | |
| 5 | | | | | | | | | |
| 2 | | | | | | | | | |
| 7 | | | | | | | | | |
| 3 | | | | | | | | | |
| 2 | | | | | | | | | |
| 5 | | | | | | | | | |

ii) Allgemein könnte man annehmen, dass bei der Verwendung von mehr Kacheln im Gegenzug weniger Seitenzugriffsfehler auftreten. Ist das wirklich immer so? Begründen Sie Ihre Antwort kurz!

Fortsetzung nächste Seite!

## 2. Petrinetze




Zur Regelung des Verkehrs an einer Kreuzung werden vier Ampeln eingesetzt. Schräg gegenüberliegende Ampeln (d.h. Ampel a1 und a3 sowie a2 und a4) haben stets identische Farbbilder. Die Signalabfolge jeder Ampel lautet:

- Grün: Die Einfahrt ist freigegeben.
- Gelb: Achtung, es wird gleich Rot-Kreuzung freigeben.
- Rot: Die Einfahrt ist verboten.
- Rot-Gelb: Übergang von Rot nach Grün. Gleich wird die Erlaubnis zur Fahrt gegeben.

a) **Petrinetz**

Zeichnen Sie ein Petri-Netz, das die Signalabfolgen der Ampeln a1 und a2 modelliert! Stellen Sie durch entsprechende Modellierung sicher, dass die Ampeln a1 und a2 nur bei Signalbild Rot den selben Zustand haben! Zudem soll das Netz derart aufgebaut sein, dass dieselbe Ampel mehrere Male die komplette Signalfolge durchlaufen kann, ohne dass inzwischen die andere Ampel schalten muss. Vergessen Sie nicht, eine Anfangsmarkierung anzugeben! Erläutern Sie kurz die Funktion der verwendeten Stellen und Transitionen!

b) **Kritischer Bereich**

Kennzeichnen Sie diejenigen Stellen in Ihrem Petrinetz, die den kritischen (exklusiven) Bereich darstellen!

c) **Erreichbarkeitsgraph**

Geben Sie den Erreichbarkeitsgraphen für Ihr Petrinetz an!

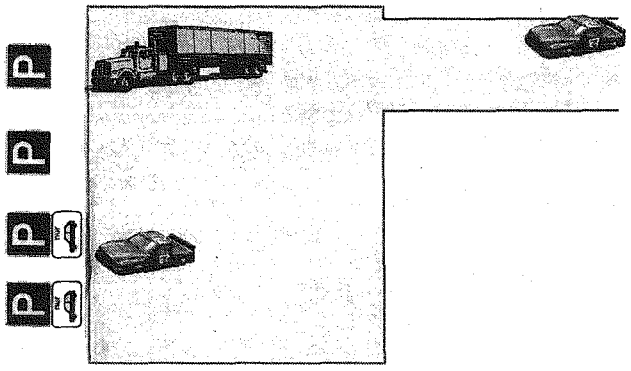
d) **Verklemmungen**

Diskutieren Sie mit Hilfe des Erreichbarkeitsgraphen, ob es im Netz zu Verklemmungen kommen kann!

Fortsetzung nächste Seite!

## 3. Semaphore

In dieser Aufgabe geht es um das Konzept der Semaphore am Beispiel eines Parkhauses. Fahrzeuge (im Falle von Betriebssystemen: Prozesse) fahren in ein Parkhaus mit 4 Parkplätzen ein und aus. Die Ein- und Ausfahrt ist so schmal, dass sie nur von einem Fahrzeug passiert werden kann. Dabei muss immer gelten, dass sich maximal so viele Fahrzeuge im Parkhaus befinden (inklusive eines Fahrzeuges in der Ein- und Ausfahrt), wie Stellplätze vorhanden sind. Ein Fahrzeug darf die Einfahrt also nicht rückwärts wieder verlassen, weil kein Stellplatz frei ist. Von den 4 Stellplätzen des Parkhauses sind nur 2 groß genug, dass auch LKWs dort abgestellt werden können. Generell darf auf einem Stellplatz nur ein Fahrzeug stehen. Die folgende Abbildung zeigt einen Beispielzustand des Parkhauses mit 2 belegten Stellplätzen.




### a) Semaphore - Funktionsweise

Geben Sie informell die Funktionsweise eines Semaphors an!

### b) Deadlockbedingungen

Nennen Sie die 4 Deadlock-Bedingungen! Geben Sie für jede Bedingung ein Beispiel, inwiefern sie in dem Parkhausszenario erfüllt wird!

### c) Synchronisation mit Semaphoren

Für sich alleine betrachtet laufen die beteiligten Prozesse folgendermaßen ab:

```
Process PKW
{
 while (TRUE)
 {
  //PKW vor Parkhaus
  <durch Ein- und Ausfahrt fahren>;
  <Parken>;
  <durch Ein- und Ausfahrt fahren>;
  //PKW vor Parkhaus
 }
}
```

```
Process LKW
{
 while (TRUE)
 {
  //LKW vor Parkhaus
  <durch Ein- und Ausfahrt fahren>;
  <Parken>;
  <durch Ein- und Ausfahrt fahren>;
  //LKW vor Parkhaus
  }
 }
```

Die beiden Prozesse sollen nun synchronisiert werden, indem (in Pseudocode-Notation) in geeigneter Weise Semaphore deklariert und Semaphor-Operationen eingefügt werden.

Fortsetzung nächste Seite!

Die Pseudocode-Operation zum Deklarieren eines Semaphors mit Namen `name` mit Startwert `n` lautet:

```
name(n);
```

Die Pseudocode-Operationen zum Aufrufen von P und V auf dem Semaphore mit Namen `name` lauten:

```
P(name)
```

```
V(name)
```

Des weiteren sei ein boolescher Wert `großerStellplatz` gegeben, über den stets bestimmt werden kann, ob ein Fahrzeug im Begriff ist, sich auf einen großen Stellplatz zu stellen, beziehungsweise ob es auf einem großen Stellplatz steht:

if (`großerStellplatz`) then *Pseudocode-Operation*

i) Zu Beginn sei das Parkhaus leer. Listen Sie alle benötigten Semaphoren auf und erklären Sie ihren Zweck beziehungsweise ihre Pragmatik!

ii) Fügen Sie in den oben aufgelisteten Pseudocode an geeigneter Stelle entsprechende Deklarationen sowie Aufrufe von P und V Operationen ein!

iii) Erläutern Sie kurz, warum Ihre Lösung die Bedingung der *Mutual Exclusion* erfüllt!

Fortsetzung nächste Seite!

## 4. Dateisysteme

### a) Kleine Blockgrößen

Beim Anlegen eines Dateisystems kann die zu verwendende Blockgröße unterschiedlich gewählt werden. Geben Sie je einen Vor- und Nachteil von kleinen Blockgrößen an!

### b) Unix Calls

Nennen und beschreiben Sie insgesamt vier UNIX-Systemaufrufe zur Manipulation von Dateisystemen, Dateien oder Verzeichnissen!

### c) Baumstruktur und Links

Üblicherweise sind UNIX-basierte Dateisysteme hierarchisch in Baumstrukturen gegliedert. Diese Baumstrukturen bilden gerichtete azyklische Graphen. Leider werden diese Strukturen durch Links zerstört, d. h. die Eigenschaft der Zyklenfreiheit geht verloren. Warum wird dies dennoch durchgeführt? Nennen Sie je einen Vor- und Nachteil, der durch das Zulassen von Links entsteht!

### d) Dateinamen

Wo speichert ein NTFS-basiertes Dateisystem einen Dateinamen? Wo ein UNIX-basiertes Dateisystem?

### e) Inode-Struktur

Skizzieren Sie graphisch eine UNIX-inode-Struktur! Gehen Sie davon aus, dass ein inode zehn Blöcke direkt adressieren kann, über je einen Eintrag für einfach indirekte, zweifach indirekte und dreifach indirekte Adressen verfügt. Metadaten müssen nicht vollständig aufgezählt werden – drei Beispiele genügen. Machen Sie in Ihrer Skizze den Unterschied zwischen direkten, einfach indirekten, zweifach indirekten und dreifach indirekten Adressen deutlich und erläutern Sie, wozu diese verwendet werden!

Fortsetzung nächste Seite!

## 5. Prozesse, Threads und wechselseitiger Ausschluss

### a) Bedingungen an WA-Lösung

Nennen und erläutern Sie kurz vier Bedingungen, die eine gute Lösung zur Garantie des wechselseitigen Ausschlusses für Prozesse erfüllen sollte!

### b) Thread ↔ Prozess

i) Erläutern Sie den Unterschied zwischen Thread und Prozess! Gehen Sie dabei auf deren Beziehung zueinander ein!

ii) Nennen Sie 2 Beispiele für Informationen bzw. Datenstruktur-Instanzen, die für jeden (User-)Thread spezifisch sind!

iii) Nennen Sie 2 Beispiele für Informationen bzw. Datenstruktur-Instanzen, die für jeden Prozess spezifisch sind (die also (User-)Threads gemeinsam nutzen müssen)!

### c) Gründe für Threads

Nennen Sie vier Gründe für die Einführung von Threads!

### d) Wechselseitiger Ausschluss

Sind folgende Ansätze zum wechselseitigen Ausschluss von Prozessen wirksam? Begründen Sie kurz! Wenn Sie wirksam sind, welche Nachteile haben Sie?

- Abschalten (Disabling) von Interrupts
- Einfache Lock Variable ohne TSL

P1 and P2

```
while (TRUE){
    if(lock == 0){
       lock = 1;
       critical_region();
       lock = 0;
    }
    non_critical_region();
}
```

- Lock Variable mit TSL

P1 and P2

```
enterCriticalRegion:
       TSL REGISTER, LOCK          copy lock to register and set lock to 1
       CMP REGISTER, #0            was lock zero?
       JNE enterCriticalRegion     if it was not zero (lock was set): loop
       RET                         return

leaveCriticalRegion:
       MOVE LOCK, #0               store 0 in lock
       RET                         return
```

- Strict Alteration

P1

```
while (TRUE){
    while(turn!=0);
    critical_region();
    turn = 1;
    non_critical_region();
}
```

P2

```
while (TRUE){
    while(turn!=1);
    critical_region();
    turn = 0;
    non_critical_region();
}
```