Lehrstuhl für Automatisierung und Informationssysteme
Prof. Dr.-Ing. B. Vogel-Heuser

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Sommersemester 2013

30.08.2013

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält 29 nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

Bitte nicht mit rot oder grün schreibenden Stiften oder Bleistift ausfüllen!

Diesen Teil nicht ausfüllen.

Aufgabe	GL	BS	MSE	C	∑	**Note**
erreichte Punkte						
erzielbare Punkte	48	48	48	96	240	

Nachname, Vorname _____ Matrikelnummer _____

Punkte

Aufgabe GL: Zahlensysteme und logische Schaltungen

Aufgabe GL: 48 Punkte

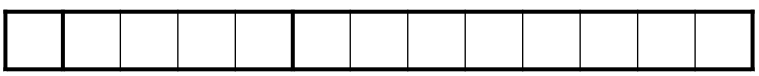
1. Zahlensysteme

a) Überführen Sie die unten angegebenen Zahlen in die jeweils anderen Zahlensysteme.
Hinweis: Achten Sie genau auf die jeweils angegebene Basis!

(1) **(102)$_3$ = ( )$_{10}$ = ( )$_2$**

(2) **(10,75)$_{10}$ = ( )$_2$**

b) Rechnen Sie die gegebene Zahl $Z_{10} = 6{,}353$ in eine Gleitkommazahl nach IEEE 754 um. Die Größe der Speicherbereiche für Vorzeichen, Mantisse und Exponent entnehmen Sie der folgenden Speicherdarstellung.



V e (4 Bit) M (8 Bit)

Hinweis: Ergebnisse und Nebenrechnungen außerhalb der dafür vorgesehenen Textblöcke werden nicht bewertet.

Vorzeichen: V

V =

Mantisse (ohne Vorkommastelle): M

M =

Bias: B **Exponent: E**

B = **E =**

Biased Exponent: e

e =

IEEE 754 Gleitkommazahl: Z

Z =

Nachname, Vorname _____ Matrikelnummer _____

Punkte

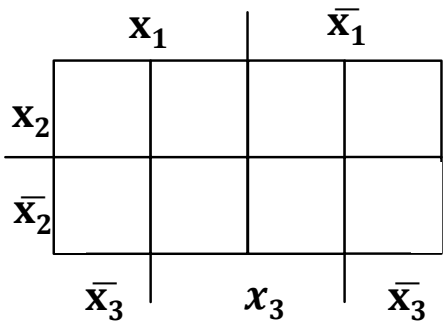
2. Normalformen und Minimierung

Gegeben ist die folgende Wahrheitstabelle mit drei Eingängen und einem Ausgang.

Dez	x_1	x_2	x_3	y
0	0	0	0	X
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

a) Erstellen Sie die KNF (*Konjunktive Normalform*) aus der Wahrheitstabelle.

b) Minimieren Sie die DNF (*Disjunktive Normalform*) mit Hilfe des *KV-Diagramms* und schreiben Sie die minimierte Funktion in *boolescher Algebra* auf.

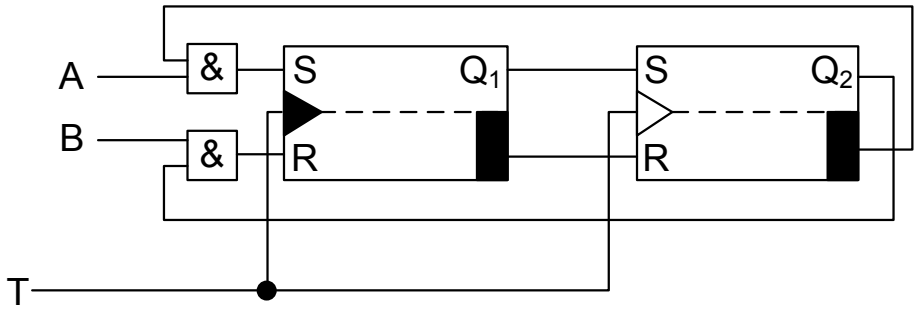


Nachname, Vorname _____ Matrikelnummer _____

Punkte

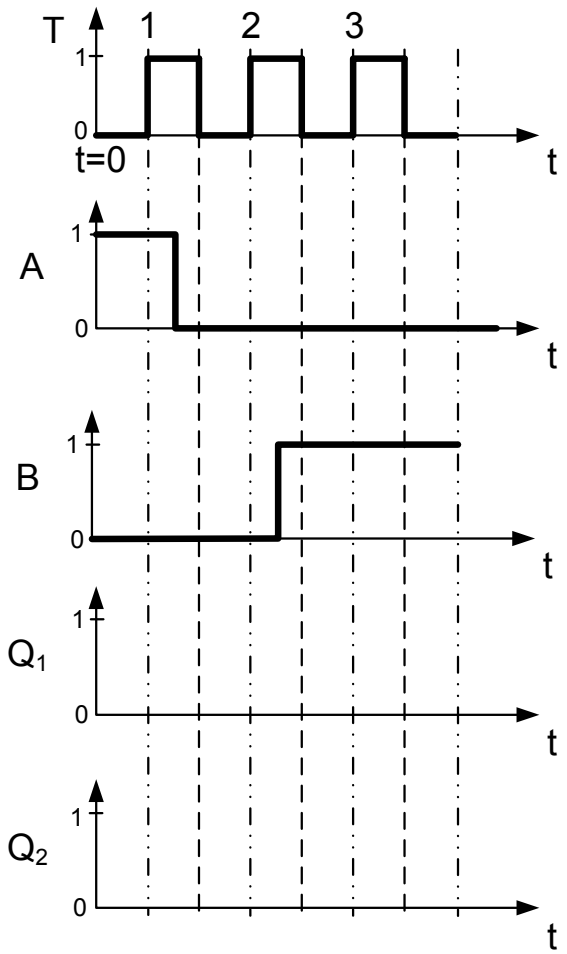
3. Flip-Flops

Gegeben sei die folgende Master-Slave Flip-Flop-Schaltung:



Bei t = 0 seien alle Flip-Flops im folgenden Zustand: $Q_1 = 1$ und $Q_2 = 1$.

Analysieren Sie die Schaltung, indem Sie die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme eintragen, mit den Eingangssignalen A und B. Die Signallaufzeiten können bei der Analyse vernachlässigt werden.



Nachname, Vorname

Matrikelnummer

Punkte

4. Befehlsabarbeitung MMIX-Rechnerarchitektur

Im Befehlsspeicher eines MMIX-Rechners liegt als nächster abzuarbeitender Befehl (hexadezimal*) 0x89 17 16 19* an. Zu diesem Zeitpunkt befinden sich im Register- und Datenspeicher die in Tabelle GL-2 und Tabelle GL-3 (siehe folgende Seite) gezeigten Werte.

Wandeln Sie den Maschinenbefehl mit Hilfe der Tabelle GL-1 in Assemblersprache um und führen Sie diesen mit den Werten der beiden untenstehenden Tabellen aus. Umkreisen Sie alle verwendeten Register- und Datenspeicheradressen und tragen Sie das Ergebnis der MMIX-Befehlsabarbeitung an die entsprechende Adresszeile in den Registerspeicher ein (in die Spalte *Wert nach Befehlsausführung*).

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9		0x_C	0x_D	
0x0_	TRAP	FCMP	…	FADD	FIX	…
	FLOT	FLOT I		SFLOT	SFLOT I	
0x1_	FMUL	FCMPE	…	FDIV	FSQRT	…
	MUL	MUL I		DIV	DIV I	
0x2_	ADD	ADD I	…	SUB	SUB I	…
	2ADDU	2ADDU I		8ADDU	8ADDU I	
…	…	…	…	…	…	…
0x8_	LDB	LDB I	…	LDW	LDW I	…
	LDT	LDT I		LDO	LDO I	
0x9_	LDSF	LDSF I	…	CSWAP	CSWAP I	…
	LDVTS	LDVTS I		PREGO	PREGO I	
…	…	…	…	…	…	…

Tabelle GL-1: MMIX-Code-Tabelle

Nachname, Vorname _____ Matrikelnummer _____

Punkte

Registerspeicher		
Adresse	Wert	Wert nach Befehlsausführung
$0x00	0x0…50	0x
…		
$0x15	0x0…A1	0x
$0x16	0x0…04	0x
$0x17	0x0…02	0x
$0x18	0x0…A1	0x
$0x19	0x0…06	0x
$0x1A	0x0…BD	0x
$0x1B	0x0…E3	0x
$0x1C	0x0…33	0x
…		
$0xFF	0x0…50	0x

Tabelle GL-2: Registerspeicher

Datenspeicher	
Adresse	Wert
M[0x0…00]	0x01
…	
M[0x0…18]	0x21
M[0x0…19]	0x8E
M[0x0…1A]	0xBB
M[0x0…1B]	0x2C
M[0x0…1C]	0xA1
M[0x0…1D]	0x21
M[0x0…1E]	0x32
M[0x0…1F]	0xCB
M[0x0…20]	0xE3
M[0x0…21]	0x8F
M[0x0…22]	0x77
M[0x0…23]	0x6F
…	
M[0xF…FF]	0x6F

Tabelle GL-3: Datenspeicher

Maschinensprache:	**0x89 17 16 19**
Assembler-Code:	

Nachname, Vorname

Matrikelnummer

Punkte

Aufgabe BS: Betriebssysteme

Aufgabe BS:
48 Punkte

1. Scheduling

Fünf Prozesse (P1 bis P5) sollen mit einem Einkernprozessor abgearbeitet werden. Das Diagramm BS-1 zeigt die Zeiten, zu denen die Prozesse am Einkernprozessor eintreffen und die Ausführungszeiten der einzelnen Prozesse. Diese Prozesse sollen zur Laufzeit mit unterschiedlichen Scheduling-Verfahren geplant werden. Alle Schedulingverfahren beginnen zum Zeitpunkt t = 0T. Für die Scheduling-Verfahren, bei denen feste Prioritäten berücksichtigen werden müssen, ist in der Tabelle BS-2 die entsprechende Prioritätenverteilung (Prioritäten 1, 2 und 3) gegeben.

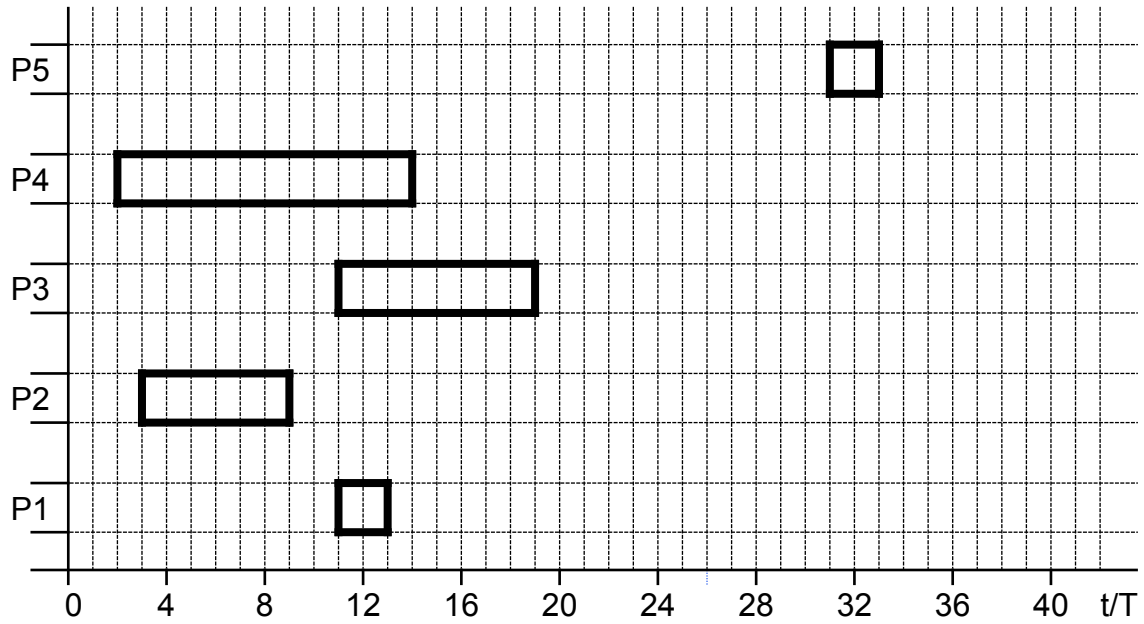


Diagramm BS-1: Sollzeitverlauf der Prozesse P1 bis P5

Prozess	P1	P2	P3	P4	P5
Priorität	1 (hoch)	1	2	3	3 (niedrig)

Tabelle BS-2: Prioritätenverteilung

Hinweis: Bitte füllen Sie die nachfolgenden Aufgaben nach dem folgenden Schema aus:

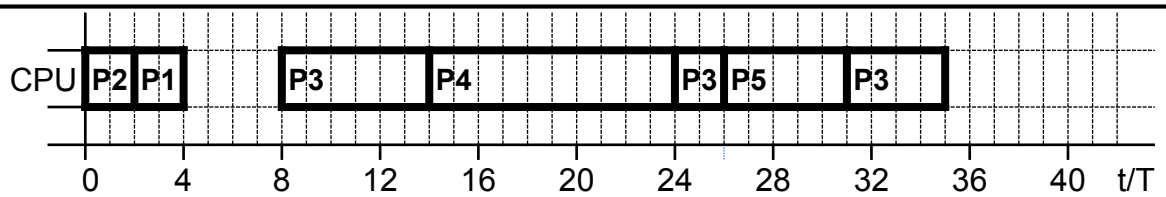


Diagramm: Lösungsbeispiel

Nachname, Vorname

Matrikelnummer

Punkte

a) In der ersten Teilaufgabe sollen die fünf Prozesse nicht-präemptiv nach ihren Prioritäten eingeplant werden. Besitzen zwei Prozesse die gleiche Priorität, wird nach dem First In First Out-Verfahren (FIFO) entschieden.

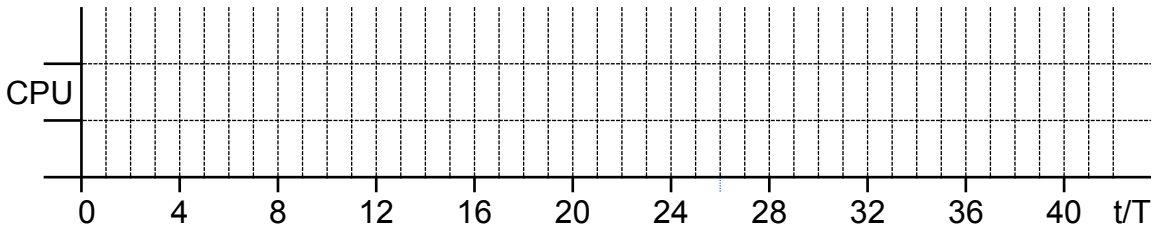


Diagramm: Prioritäten und First In First Out (FIFO)

b) In der zweiten Teilaufgabe erfolgt die Abarbeitung der oben gegebenen fünf Prozesse präemptiv mit Hilfe des Round Robin-Verfahrens (RR) und der Zeitschlitzgröße von 4T. Kann zwischen zwei Prozessen mit Round Robin keine eindeutige Reihenfolge bestimmt werden, erfolgt die Einplanung dieser beiden Prozesse nach den drei Prioritätsstufen. Für die Zeitscheibe soll angenommen werden, dass diese unendlich viele Schlitze besitzt und so zu jedem Einteilungszeitpunkt ausreichend freie Schlitze vorhanden sind.

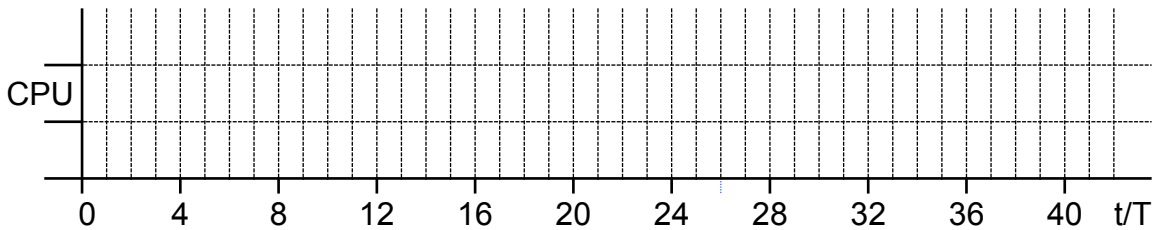


Diagramm: Round Robin (RR) und Prioritäten

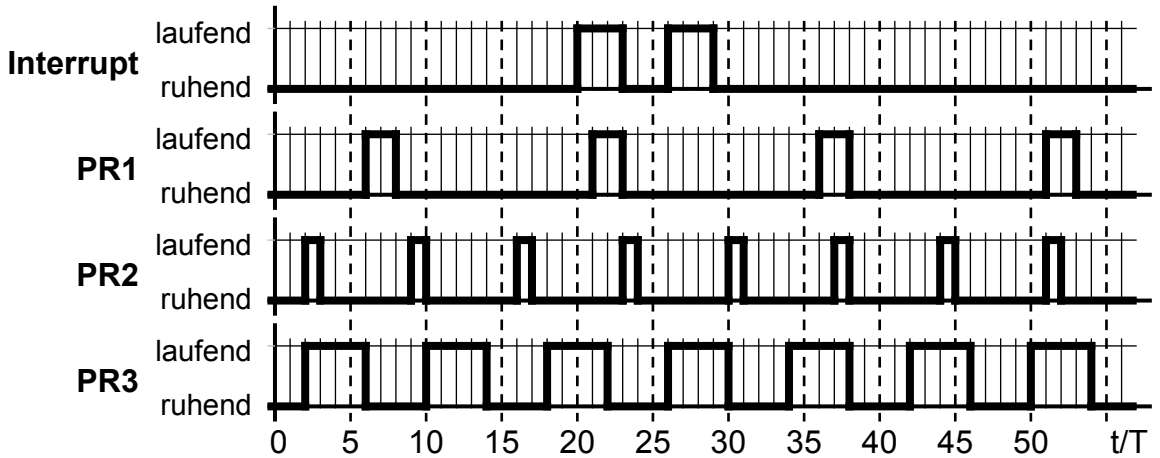
Nachname, Vorname _____ Matrikelnummer _____

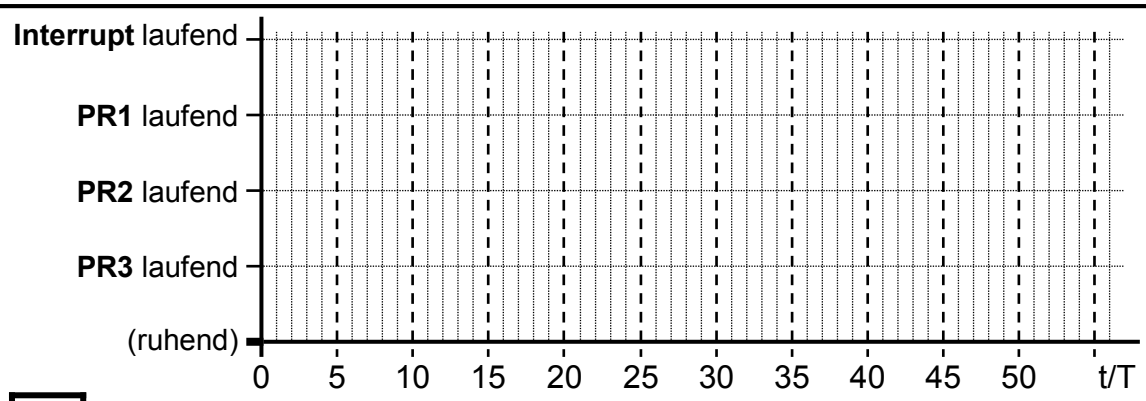
Punkte

2. Asynchrone Programmierung

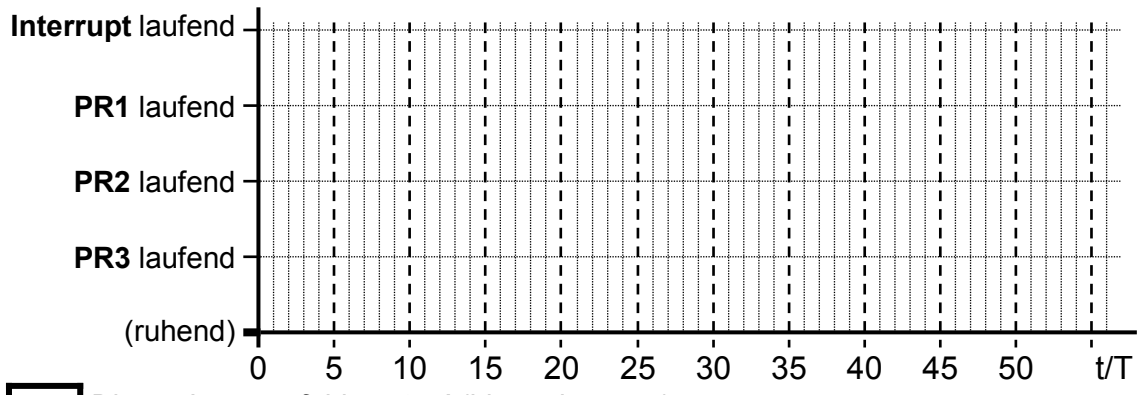
Drei periodische und präemptive Prozesse (PR1 bis PR3) sollen mit dem Verfahren der asynchronen Programmierung auf einem Einkernprozessor ausgeführt werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigste Priorität. Die Ausführung wird durch zwei Interrupts unterbrochen. Tragen Sie in das unten angegebene Diagramm die tatsächliche Abarbeitung der Rechenprozesse nach dem Verfahren der asynchronen Programmierung ein.

Hinweis: Bei größeren Korrekturen verwenden Sie bitte das Ersatzfeld und markieren das zu wertende Lösungsfeld.





☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*



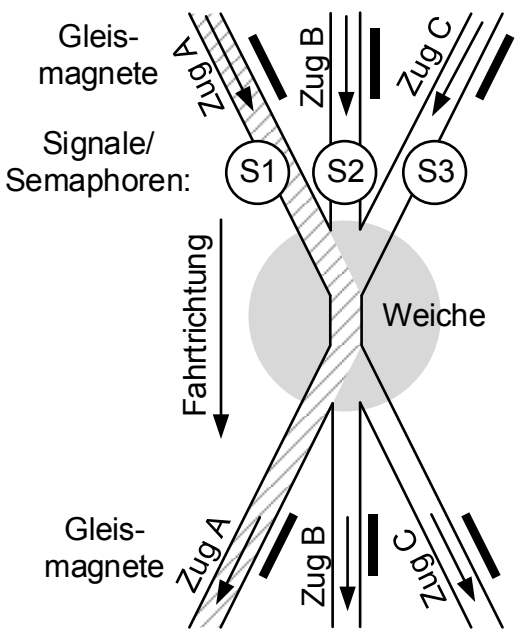
☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*

Nachname, Vorname _____ Matrikelnummer _____

Punkte

3. Semaphoren

Eine Weiche soll die Durchfahrt von drei Zugklassen (Zug A, Zug B, Zug C) durch eine Engstelle regeln. Da die Zugklasse B (Zug B) doppelt so häufig verkehrt wie die anderen Klassen, soll die Weiche die Züge immer in folgender Reihenfolge passieren lassen: $\overline{ABBC}$.



Entwickeln Sie mit Hilfe der drei Semaphoren S1, S2 und S3 für jede der drei Tasks (Züge) eine Anordnung von Semaphoren-Operationen. Geben Sie auch Initialwerte für die drei Semaphoren an.

Hinweis: Die für die Ausführung eines Tasks notwendigen Semaphoren sollen nur im Block verwendet werden. Beispielsweise würde ein Task X mit folgenden Semaphoren-Operationen

Task	X
Semaphoren-Operationen ↓	P(S7)
	P(S7)
	P(S8)
	…
	V(S8)
	V(S9)

nur starten, wenn alle benötigten Semaphoren (zweimal S7 und einmal S8) gleichzeitig frei sind.

Semaphore:	**S1**	**S2**	**S3**
Initialwert:			

Task:	**A**	**B**	**C**
Semaphoren-Operationen ↓			

sich wiederholende Folge der Zugklassen: $\overline{ABBC}$

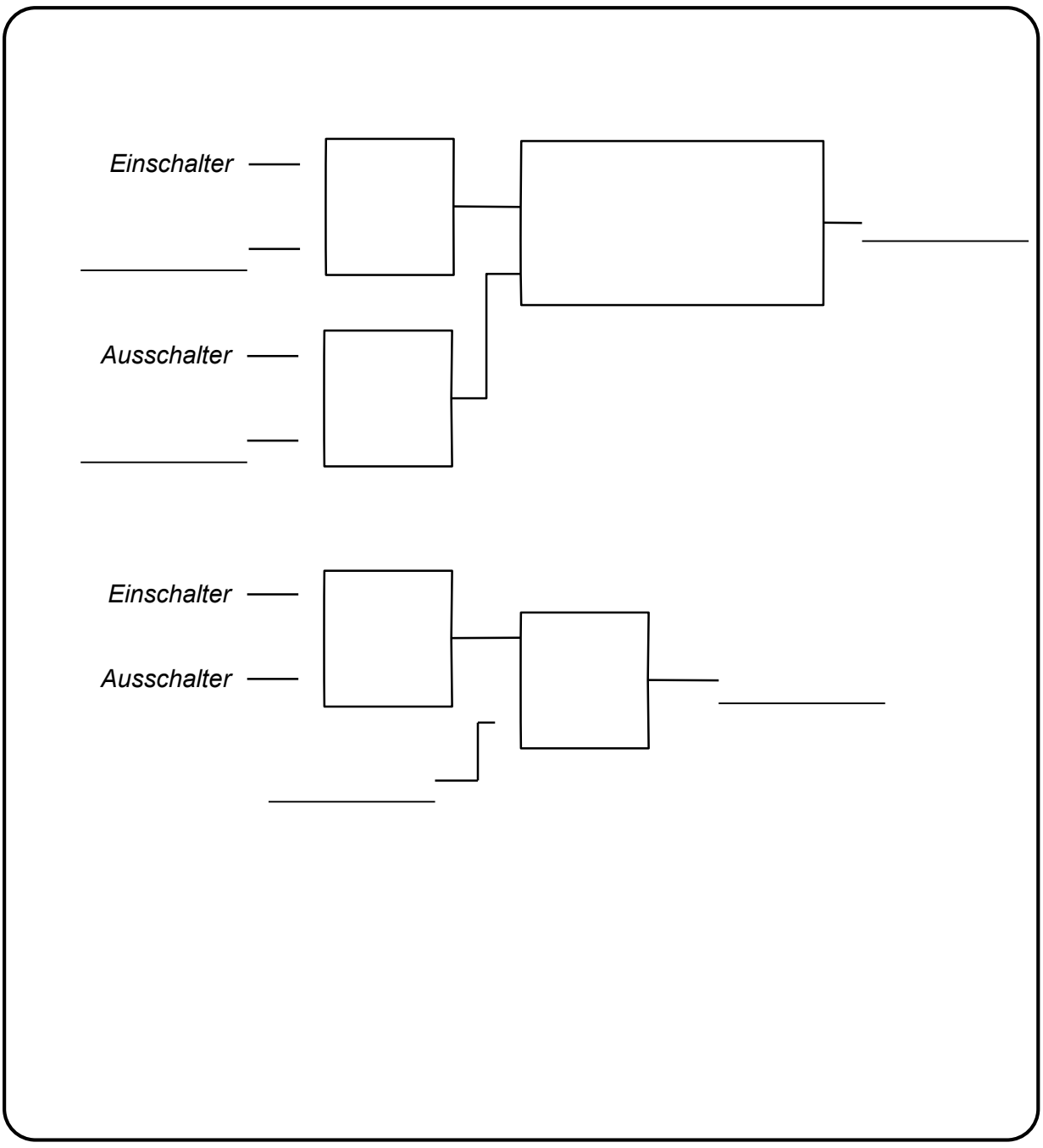
Nachname, Vorname _____ Matrikelnummer _____

Punkte

4. IEC 61131-3: Funktionsbausteinsprache (FBS)

Eine Glühlampe soll durch ein Tippen auf einen von zwei Tastern gesteuert werden. Ein Einschalter schaltet die Lampe ein (*Glühlampe* = 1), ein Ausschalter aus (*Glühlampe* = 0). Zusätzlich besitzt das System einen Not-Aus-Schalter. Die Lampe bleibt aus, solange dieser betätigt ist (*Not-Aus* = 0). Wenn der Not-Aus gedrückt ist (*Not-Aus* = 0) und ein oder beide Taster gedrückt werden, soll ein kurzer Signalton (einen Zyklus lang) zu hören sein (*Signalton* = 1).

Vervollständigen Sie den untenstehenden Funktionsbausteinplan mit den fehlenden Variablennamen, Funktionsblöcken und den Verbindungen der Variablen.



Nachname, Vorname

Matrikelnummer

Punkte

Aufgabe MSE Teil 1: Automaten

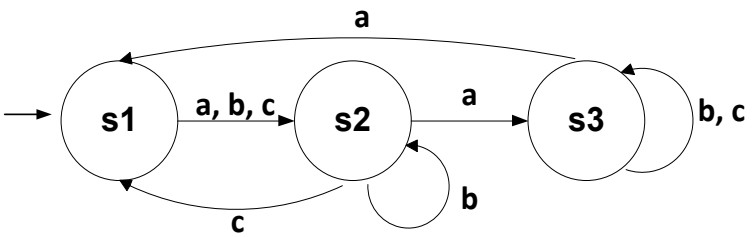
Aufgabe MSE1: 24 Punkte

1. Theorie

Was kann mit einer Turing-Maschine modelliert werden, das mit einem endlichen deterministischen Automaten nicht modelliert werden kann?

2. Darstellung

Gegeben sei folgendes Übergangsdiagramm:



Ergänzen Sie die zu dem Übergangsdiagramm gehörende Übergangstabelle.

T \ S	s1	s2	s3
a			
b			
c			

Nachname, Vorname

Matrikelnummer

Punkte

3. Automaten mit Ausgabe

Entwerfen Sie den Übergangsgraphen eines Moore-Automaten, der stets den Modulo von 2 ausgibt. Das Eingabealphabet ist T={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Die Zahlen werden bei der Eingabe dabei immer *aufaddiert*. Beispiel für eine Eingabe:

- Start; Ausgabe 0 → Eingabe 3; Ausgabe 1 → Eingabe 2; Ausgabe 1 → Eingabe 1; Ausgabe 0

Hinweis: Modulo 2 gibt als Ergebnis den Rest der Division (der aufaddierten Eingaben) durch 2 aus.

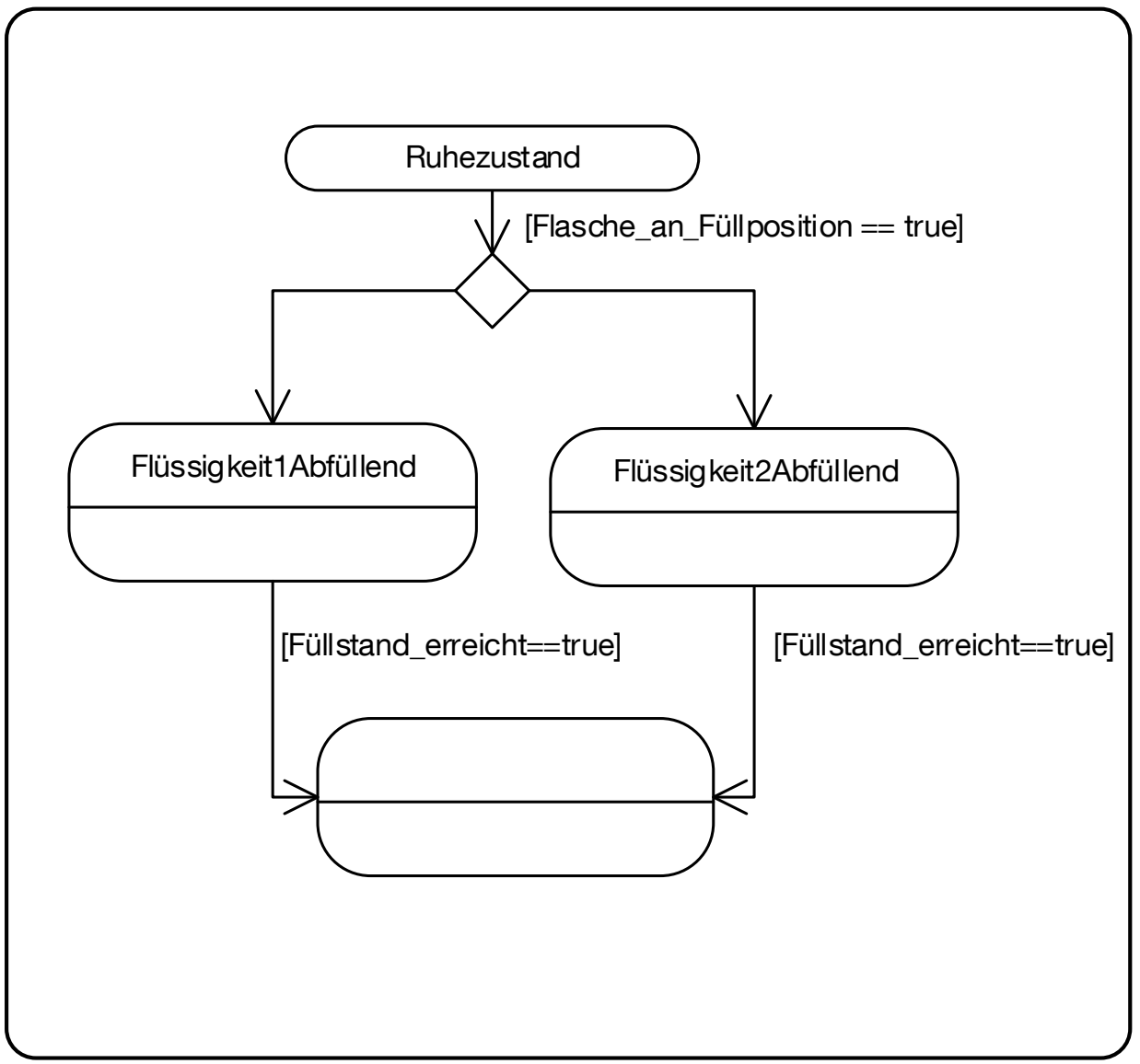
Nachname, Vorname _____ Matrikelnummer _____

Punkte

4. Zustandsdiagramm

Ein Abfüllvorgang zweier verschiedener Flüssigkeiten soll in einem Zustandsdiagramm beschrieben werden. Zu Beginn befindet sich das System im *Ruhezustand*. Sobald sich eine Flasche an der Füllposition befindet (*Flasche_an_Füllposition*) wechselt das System falls eine blaue Flasche erkannt wird (Variable *BlaueFlasche*) in den Zustand *Flüssigkeit1Abfüllend*, falls eine grüne Flasche erkannt wird (Variable *GrüneFlasche*) in den Zustand *Flüssigkeit2Abfüllend*. Im Zustand Flüssigkeit1Abfüllend muss die Aktion *Ventil1Öffnen*, im Zustand Flüssigkeit2Abfüllend die Aktion *Ventil2Öffnen,* durchgeführt werden. Sobald der Soll-Füllstand (*Füllstand_erreicht*) erreicht ist, wird in den Zustand *VentileSchließend* gewechselt, in welchem die Aktion *VentileSchließen* durchgeführt werden muss. Sind sowohl *Ventil1* und *Ventil2* geschlossen (true: offen, false: geschlossen), geht das System in den Endzustand über.

Vervollständigen Sie das Zustandsdiagramm. Fügen Sie Start- und Endzustand ein. Zeichnen Sie keine zusätzlichen Zustände ein.



Nachname, Vorname _____ Matrikelnummer _____

Punkte

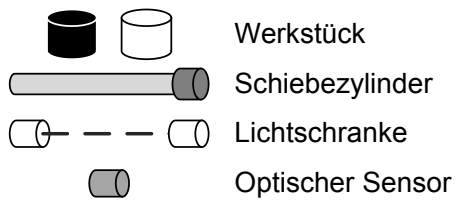
Aufgabe MSE Teil 2: Strukturierte Analyse/ Real-Time (SA/RT)

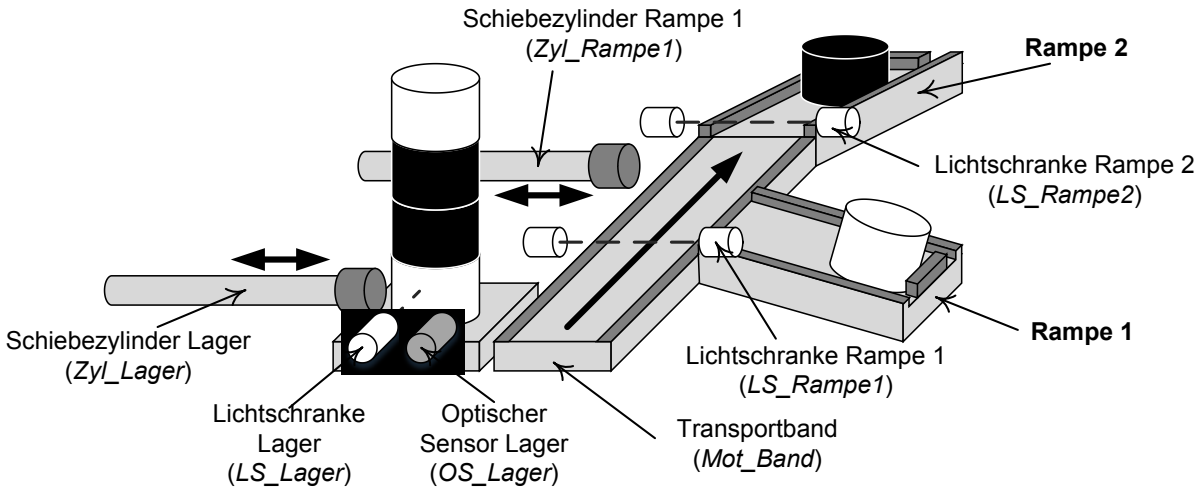
Aufgabe MSE2: 24 Punkte

Gegeben ist ein Sortiersystem, in welchem *Werkstücke übergeben und sortiert* werden sollen. Aus einem *Lager* werden zwei verschiedene Werkstücktypen (*schwarze Werkstücke* und *weiße Werkstücke*) durch den *Lagerzylinder* auf ein *Transportband* ausgeschoben und anschließend entsprechend ihrer Farbe in zwei *Rampen* sortiert.

Das Sortiersystem soll in den folgenden Aufgaben mittels Strukturierter Analyse mit Real-Time-Erweiterung modelliert werden.

Legende:





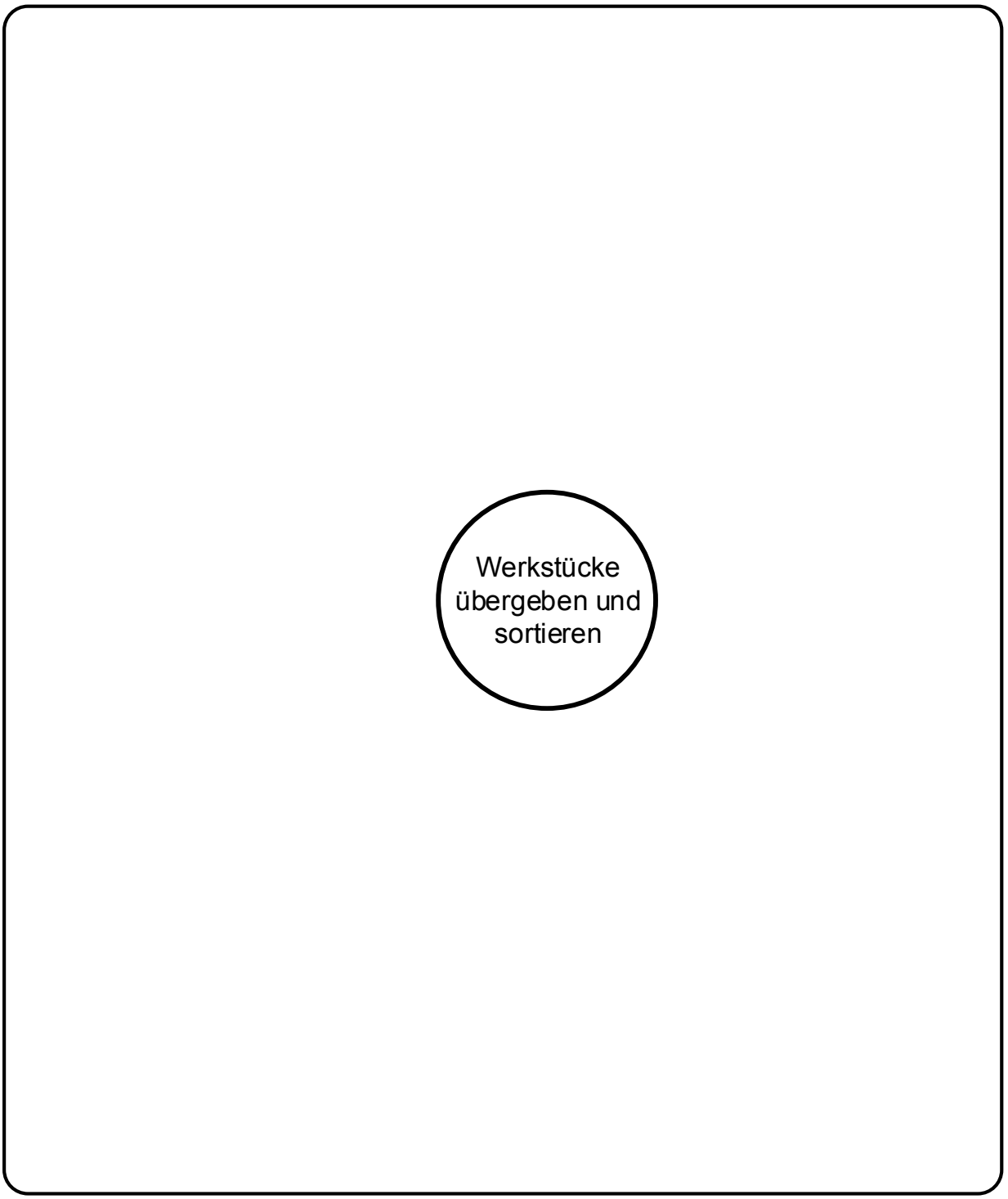
Nachname, Vorname _____ Matrikelnummer _____

Punkte

1. Kontextdiagramm

Es soll das Kontextdiagramm (Datenkontext- und Steuerkontextdiagramm in einem Diagramm) des Prozesses *Werkstücke übergeben und sortieren* modelliert werden.

Der *Bediener* gibt *Sollwerte* vor und erteilt *Kommandos* zur Prozess*steuerung*. Die *Steuerung* liefert dem Prozess die zur Berechnung notwendigen *Istwerte* und empfängt die berechneten *Stellwerte*. Das *Meldesystem* empfängt *Meldungen* vom Prozess.



Nachname, Vorname _____ Matrikelnummer _____

Punkte

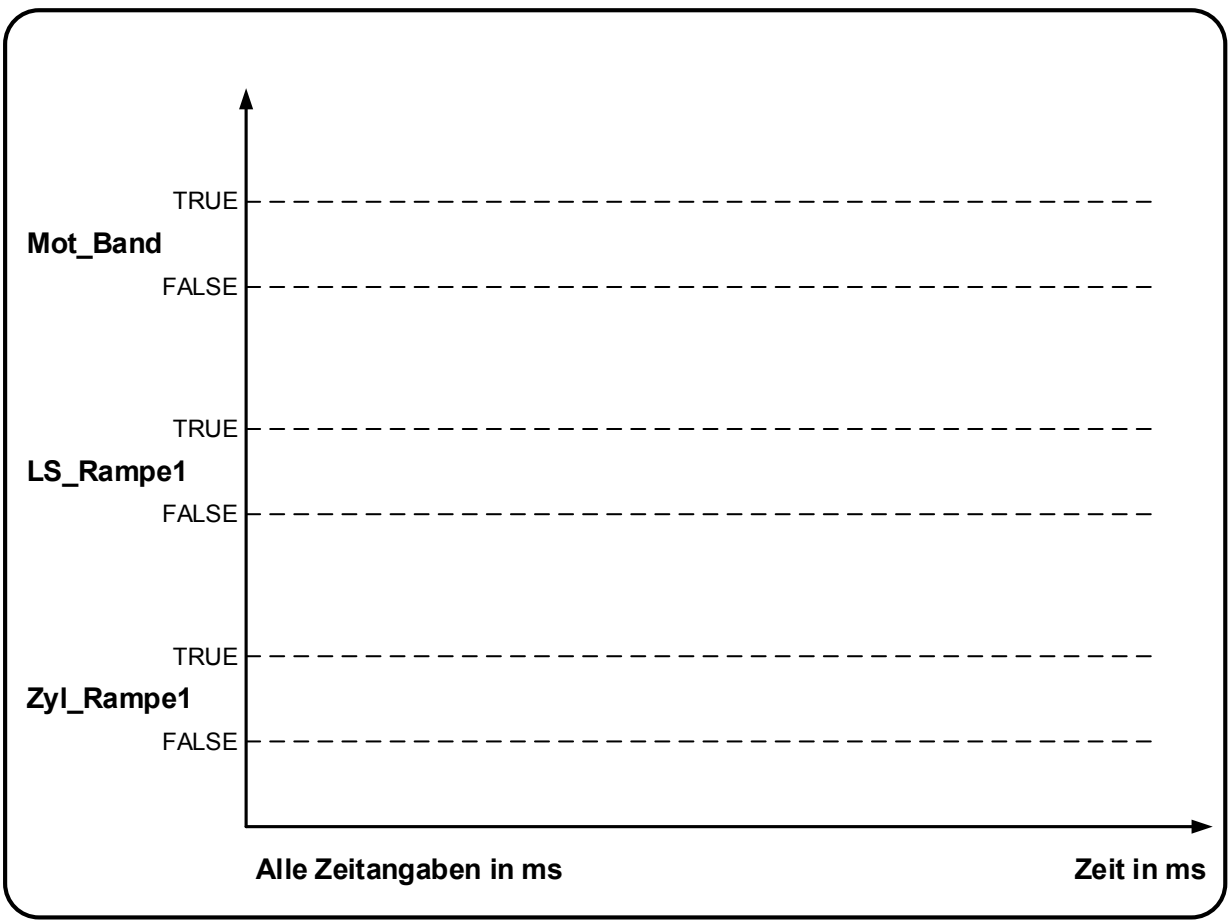
2. Antwortzeitspezifikation: Timing-Diagramm

Zur Verfeinerung des Teilprozesses *Werkstücke sortieren* soll nun eine Antwortzeitspezifikation in Form eines Timing-Diagramms durchgeführt werden, um sicherzustellen, dass der Sortiervorgang korrekt durchgeführt wird.

Die Werte der Sensoren bzw. Aktoren können jeweils *TRUE* (z.B. Werkstück erkannt, Zylinder ausfahren) oder *FALSE* (z.B. kein Werkstück erkannt, Zylinder einfahren) sein.

Ergänzen Sie das Timing-Diagramm gemäß folgender Angaben (Werteverläufe und Zeitangaben):

- Nachdem das Transportband *Mot_Band* eingeschalten wurde, muss das Werkstück innerhalb von 400+/− 40 ms am Sensor *LS_Rampe1* erkannt werden.
- Aufgrund der Breite des Werkstücks liefert der Sensor *LS_Rampe1* für 100+/− 6 ms den Wert TRUE.
- Im Falle eines weißen Werkstückes muss der Zylinder *Zyl_Rampe1* nun innerhalb von 200+/− 20 ms nach Auslösen des Sensors *LS_Rampe1* ausfahren, sodass das Werkstück korrekt aussortiert wird. Im Falle eines schwarzen Werkstückes fährt der Zylinder *Zyl_Rampe1* nicht aus.
- Das Transportband *Mot_Band* muss innerhalb von < 1500 ms nach seiner Aktivierung deaktiviert werden.



Nachname, Vorname

Matrikelnummer

Punkte

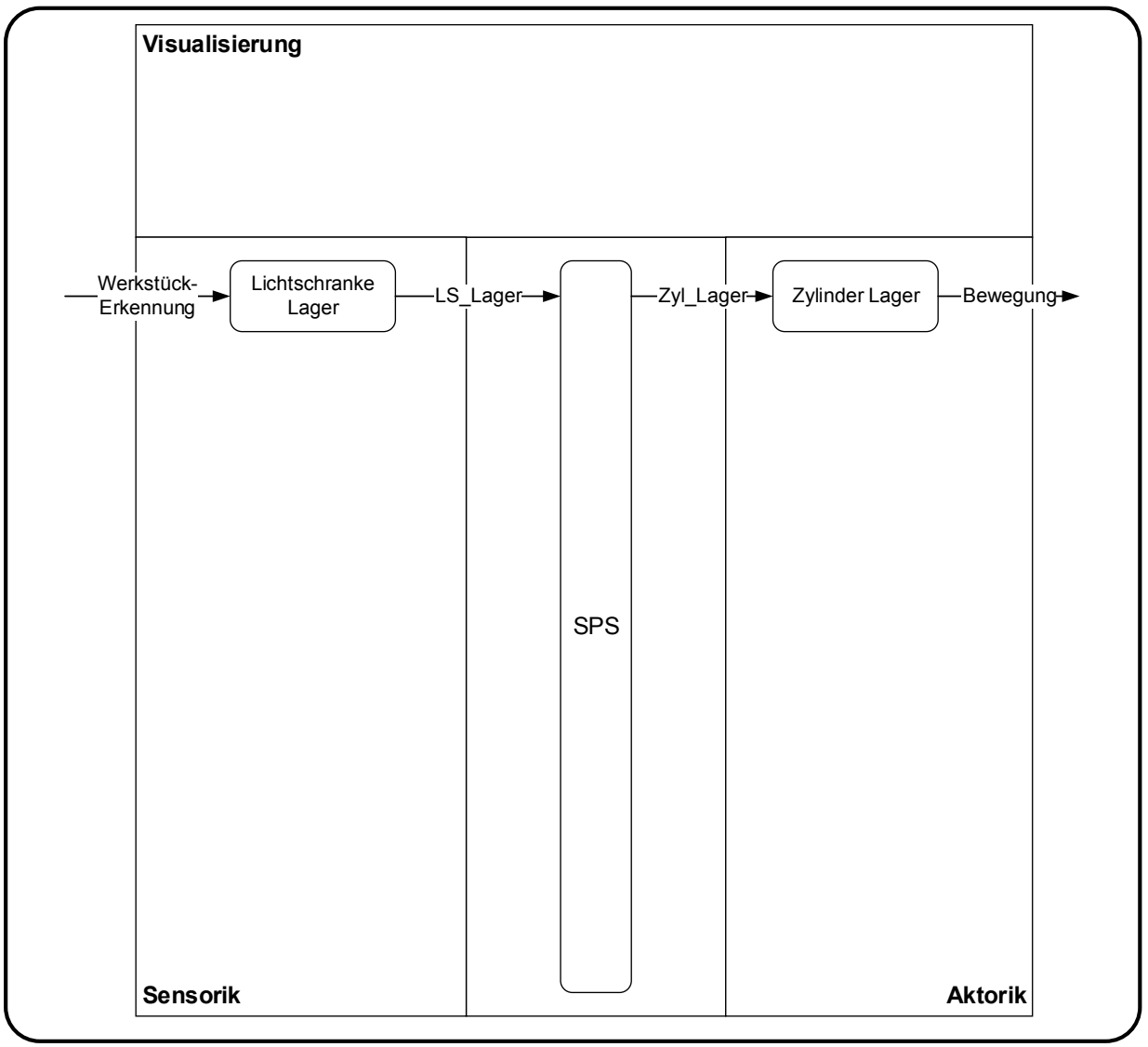
3. Architekturflussdiagramm

Eine zentrale Architektur, in der eine einzelne Speicherprogrammierbare Steuerung (SPS) das Sortiersystem steuert, soll als Architekturflussdiagramm modelliert werden.

Modellieren Sie neben den bereits angegebenen Architekturmodulen die folgenden Architekturmodule sowie die Flüsse zwischen den Architekturmodulen und der Steuerung (Flüsse von der Umgebung zu den Sensoren bzw. von den Aktoren zu der Umgebung müssen nicht modelliert werden):

- **Sensorik:** *Optischer Sensor Lager*, *Lichtschranke Rampe 1*, *Lichtschranke Rampe 2*
- **Aktorik:** *Motor Band*, *Zylinder Rampe 1*
- **Visualisierung***: Visualisierungs-PC*

Der *Visualisierungs-PC* liest *Sollwerte* vom Benutzer ein und stellt *Istwerte* dar. Diese Daten (*Soll- und Istwerte*) werden zwischen Visualisierungs-PC und Steuerung ausgetauscht. **Hinweis**: Verwenden Sie die Bezeichnungen aus der Abbildung auf Seite 15.



Nachname, Vorname _____ Matrikelnummer _____

Punkte

Aufgabe C: Programmieren in C

Aufgabe C:
96 Punkte

1. Datentypen, Ein-/Ausgabe und boolesche Algebra

a) Vervollständigen Sie die Definition und Initialisierung der folgenden Variablen mit sinnvollen Datentypen und entsprechenden Anfangsbuchstaben der Variablenbezeichnung (Beispiel: `int i Zahl = 1;`).

```
_________  __Preis = 0.65;

_________  __Mahlmethode = 'A';

_________  __Mahlfeinheit = 7345.655345345e-7;
```

b) Sortieren Sie die Datentypen `float, char, int, double` und `short` aufsteigend nach ihrem benötigten Speicherplatz. Gehen Sie dabei von einer 32-bit Rechnerarchitektur aus.

_____ <= _____ <= _____ <= _____ <= _____

c) Mit welchen zwei Datentypen aus Aufgabe b) können Gleitkommazahlen dargestellt werden? Sortieren Sie diese nach ihrer Genauigkeit bezüglich der Anzahl auflösbarer Stellen. Benutzen Sie für die Darstellung das kleiner (<) bzw. größer (>) Zeichen.

d) Werten Sie folgende Ausdrücke nach den Regeln der booleschen Algebra aus. Verwendete Variablen:

Int-Variablen: X = 17; Y = 4;
Float-Variable: Z = 4.25;
Char-Variable: A = 0x11;

(X && Y) != !A	
((Y && A) & X)	
(A | !(char) X)	
((float) (X/Y) == Z)	

Nachname, Vorname _____ Matrikelnummer _____

Punkte

2. Kontrollstrukturen

a) Das Programm zur Preisberechnung eines Kaffeeautomaten soll so erweitert werden, dass Angebotspreise automatisch bei der Berechnung berücksichtigt werden.

- An *ungeraden* Wochentagen bis auf Sonntag sollen die Preise für alle Getränke um 10% reduziert werden. Nutzen Sie eine Modulo-Operation, um die ungeraden Wochentage zu bestimmen.
- An den *restlichen* Tagen (also jetzt inkl. Sonntag) sollen nur Getränke mit Größe XL um 20% reduziert werden.

Gehen Sie davon aus, dass der Code zum Einlesen von Größe und Wochentag bereits realisiert wurde und die zugehörigen Variablen belegt wurden.
Für die Überprüfung des Wochentags ist zu beachten, dass der ENUM WTAG hier erzwungen mit der Zahl 1 (für Montag) beginnt und mit 7 (für Sonntag) endet.
Manipulieren Sie die Variable für den Preis *fPreis* für die genannten Fälle.
Nutzen Sie hierfür eine *IF-ELSE* Abfrage.

```
#include <stdio.h>
enum GROESSE { M, L, XL };
enum WTAG {MO=1,DI,MI,DO,FR,SA,SO};
float main(){
    float fPreis;

    enum GROESSE groesse = einlesenGroesse();
    enum WTAG wtag = einlesenWochentag();

    fPreis = berechnePreis(IdGetraenk, groesse);


    //Erweiterte Preisberechnung mit IF-ELSE









   return fPreis;
}
```

Nachname, Vorname _____ Matrikelnummer _____

Punkte

b) Die abgegebene Menge an Kaffee wird durch Aufruf der Funktion `leseSensorFuell()` als Fließkommazahl zurückgegeben und kann dadurch mit der Sollmenge verglichen werden.
Zur Sicherheit ist ein Sensor in der Abtropfschale integriert, so dass eine fehlende oder überlaufende Tasse entdeckt und der Brühvorgang gestoppt wird.
Der Rückgabewert der Funktion `leseSensorUeber()` wechselt dann von 0 auf 1, wenn übergelaufene Flüssigkeit erkannt wurde.
Mit Hilfe einer *WHILE-Schleife* soll nun der Füllvorgang ausgeführt werden, solange das folgende Kriterium erfüllt ist: Solange die gewünschte Füllmenge nicht abgegeben wurde und der Überlaufsensor nicht ausgelöst hat, soll die Funktion `bruehe()` ausgeführt werden.

Ergänzen Sie hierfür das folgende Programm:

```
#include <stdio.h>
float leseSensorFuell(void) { ... }  // Rückgabe in [ml]
int leseSensorUeber(void) { ... }    // Überlaufsensor
void bruehe (void) { ... }

void main (void)
{
    float sollFuell = 200; // Soll-Füllmenge in [ml]
```

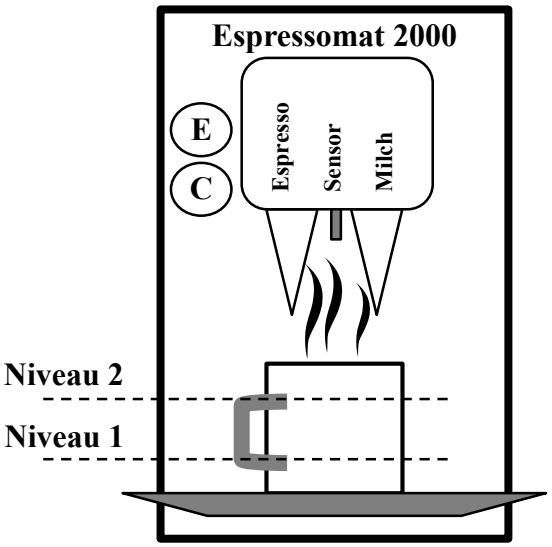
```

```

```
}
```

Nachname, Vorname _____ Matrikelnummer _____

Punkte

3. Zyklische Programmierung eines Espressovollautomaten



Die Firma ISA GmbH stellt einen Espressovollautomaten her. Dabei können zwei verschiedene Produkte zubereitet werden: Espresso und Cappuccino. Cappuccino wird aus Espresso und Milch zubereitet. Hierfür existieren zwei Knöpfe, welche die Eingabe erkennen und den jeweiligen Sensor auf '1', sowie beim Loslassen der Taste wieder auf '0' setzen.

Die Maschine besitzt darüberhinaus noch einige Komfortfunktionen. Hierfür erkennt ein Abstandssensor (vgl. Grafik) zum einen, ob eine Tasse vorhanden ist, sowie die Füllhöhe in der Tasse. Somit kann ein Espresso auf das Niveau 1 und Cappuccino auf das Niveau 2 exakt abgefüllt werden.

Weiterhin besitzt die Maschine einen Timer, so dass für den Cappuccino der aus der Milchdüse erzeugte Schaum für exakt 3 Sekunden eingesprüht und dann anschließend Espresso bis auf das Niveau 2 aufgefüllt wird.

Im Folgenden finden Sie eine Übersicht der im Programm benötigten Sensor- und Aktorvariablen, die zu verwenden sind:

Variable	**Bezeichnung**
Aktoren:	
Akt.iEspresso	Startet den Brühvorgang des Espresso
Akt.iMilch	Öffnet die Milchdüse und erzeugt Milchschaum
Sensoren:	
Sen.iWahl_Esp	Startknopf Espresso gedrückt
Sen.iWahl_Cap	Startknopf Cappuccino gedrückt
Sen.iTasse_da	Tasse auf Halter vorhanden/erkannt
Sen.iNiveau1	Abstandssensor erkennt Niveau1 (Espresso)
Sen.iNiveau2	Abstandssensor erkennt Niveau2 (Cappuccino)
vplcZeit	Systemlaufzeit der Steuerung (in Millisekunden)
iCappu	Merkvariable, ob Cappuccino oder Espresso gewählt wurde (Cappuccino=1, Espresso=0)

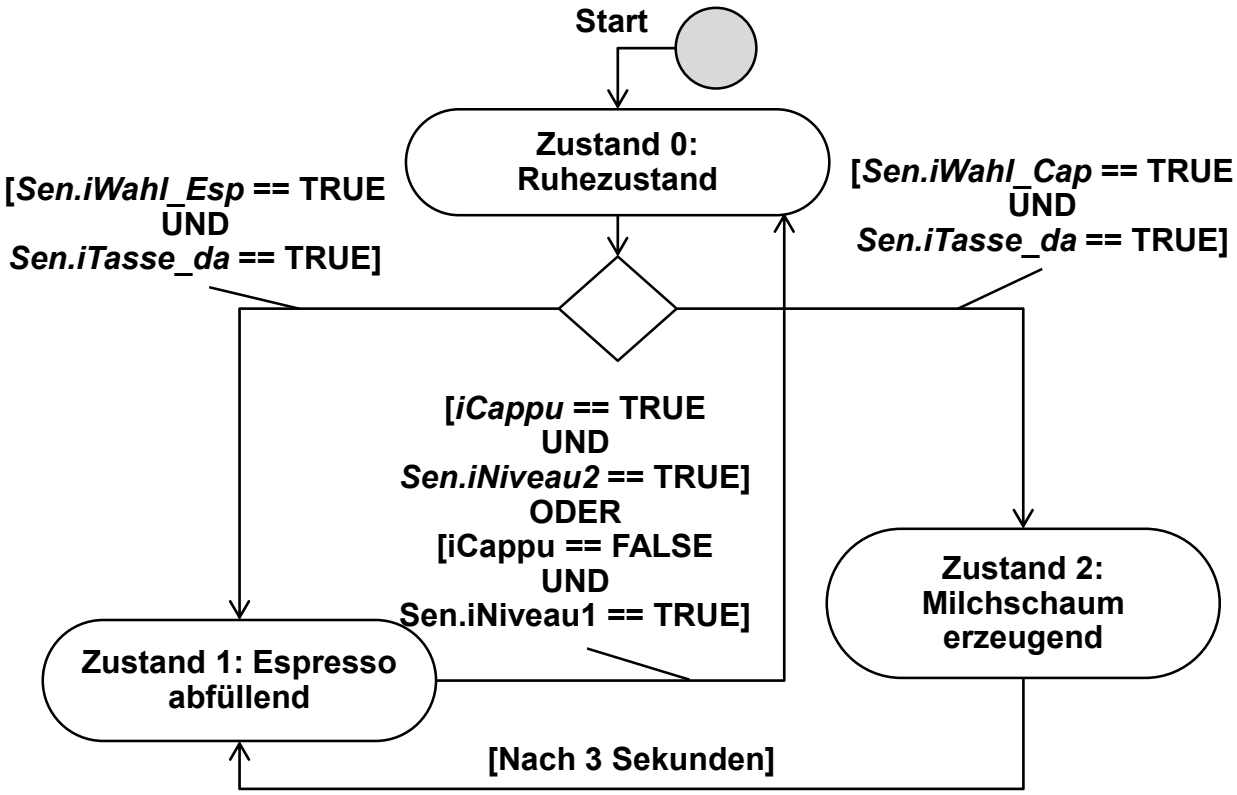
Nachname, Vorname

Matrikelnummer

Punkte

Zur Verdeutlichung des Steuerungsablaufs wurde bereits ein Zustandsdiagramm mit Übergangsbedingungen entsprechend der oben genannten Aufgabenstellung entworfen.

Übertragen Sie die Zustände, sowie deren Implementierung der Aktoransteuerung und die jeweiligen Zustandsübergänge in den lückenhaften C-Code auf der folgenden Seite. Benutzen Sie dafür die Sensor-/Aktorvariablen auf der vorherigen Seite. Beachten Sie, dass der Code zum Betrieb des Espressoautomaten zyklisch in einer Steuerung ausgeführt wird. Der Kopf des Programms sowie die Deklaration der Sensor- und Aktorvariablen sind bereits unten dargestellt und müssen nicht realisiert werden. Achten Sie darauf, alle Aktorvariablen im Ruhezustand geeignet zu initialisieren.




```
#include "eavar_kaffee2000.h"
struct SData Sen; //Sensorvariablen
struct AData Akt; //Aktorvariablen

unsigned int vplcZeit=0; // 1000 entspricht 1 Sek.
int iCappu=0; // Merkvariable ob Cappuccino oder nicht
unsigned int t=0; // Merkvariable zum Speichern der Laufzeit
int state=0;  // Merkvariable für den aktuellen Zustand
(...)
```

Nachname, Vorname _____ Matrikelnummer _____

Punkte

```
int plcZyklus()
{
    switch(state)
    {
     _________________ // Zustand 0: Warten auf Eingabe
     iCappu=0;       // Init Merkvariable und Aktoren

     ____________________________

     ____________________________

    if(_____________________________________)state=1;

    if(_____________________________________)state=2;
    break;

    _______________// Zustand 1: Espresso brühen

    ____________________________

    if(_____________________________________

       _____________________________________)state=0;

    break;

    _______________// Zustand 2: Milch aufschäumen

    ____________________________

    ____________________________

  if(!t) _____________________________ // Timer

  else if(_____________________________________)
        {
         _____________________________
          t=0; //Timer zurücksetzen
          state = 1;
        }
  break;
  } (...)
```

Nachname, Vorname _____ Matrikelnummer _____

Punkte

4. Datenbank mit einem zugehörigen Hochregallager für Kaffeebohnen

a) Für die Lagerung von Kaffeebohnen existiert bei der Firma ISA GmbH ein Hochregallager. Das Lager besitzt 5 Stockwerke (0.-4.) und je 8 abgetrennte Regalplätze (0.-7.). Zum Abspeichern der Information, welche Sorte in welchem Regalplatz gelagert ist, wird jedem Bohnentyp eine Zahl von 0 bis 127 zugeordnet. *Deklarieren* Sie im Folgenden ein Array, welches diese IDs so effizient wie möglich speichern kann, um die Kaffeebohnen bei Bedarf auffindbar zu machen. *Initialisieren* Sie weiterhin: 1. Stock, 2. Regalplatz enthält Bohnen mit ID 58.

b) Zusätzlich existiert eine weitere Datenbank, welche alle vorhandenen Bohnentypen enthält und für verschiedene Funktionen der Lagerhaltung folgende Informationen pro Bohnensorte bereitstellt:

- Stärke der Bohne (unterteilt in STARK und SCHWACH)
- Gewicht der Bohne (als Integer-Zahl)
- Name der Bohne (max. 50 Buchstaben)
- ID im Hochregallager (Zahl von 0-127)

Zur vereinfachten Handhabung soll zunächst ein neuer Datentyp „STAERKE“ definiert werden, welcher die Werte ‚SCHWACH‘ und ‚STARK‘ enthalten kann:

c) Weiterhin soll nun die Struktur für den Datentyp „BOHNE“ definiert werden, welche die oben genannten Daten für die jeweiligen Bohnen speichern kann:

Nachname, Vorname _____ Matrikelnummer _____

Punkte

d) Im Folgenden soll nun ein Array deklariert werden, welches zu Testzwecken mit den Nutzdaten von nur einem Bohnentyp initialisiert wird. Geben Sie dem Array den Namen ‚bohne' und verwenden Sie den oben angelegten Datentyp „BOHNE". Deklarieren Sie das Array mit der Länge 20. Initialisieren Sie das erste Feld mit folgenden Daten:

Nr.	**0**
Stärke	STARK
Gewicht [g]	25
Name	Columbia
ID	47

e) Um die Arraygröße bei der Auslegung des Codes beliebig zu halten, soll diese mit Hilfe eines geeigneten Präprozessor-Befehls am Anfang des Codes festgelegt werden. Verwenden Sie als Keyword MAX und legen Sie als maximale Arraygröße für den Testcode zunächst zwanzig Plätze fest.

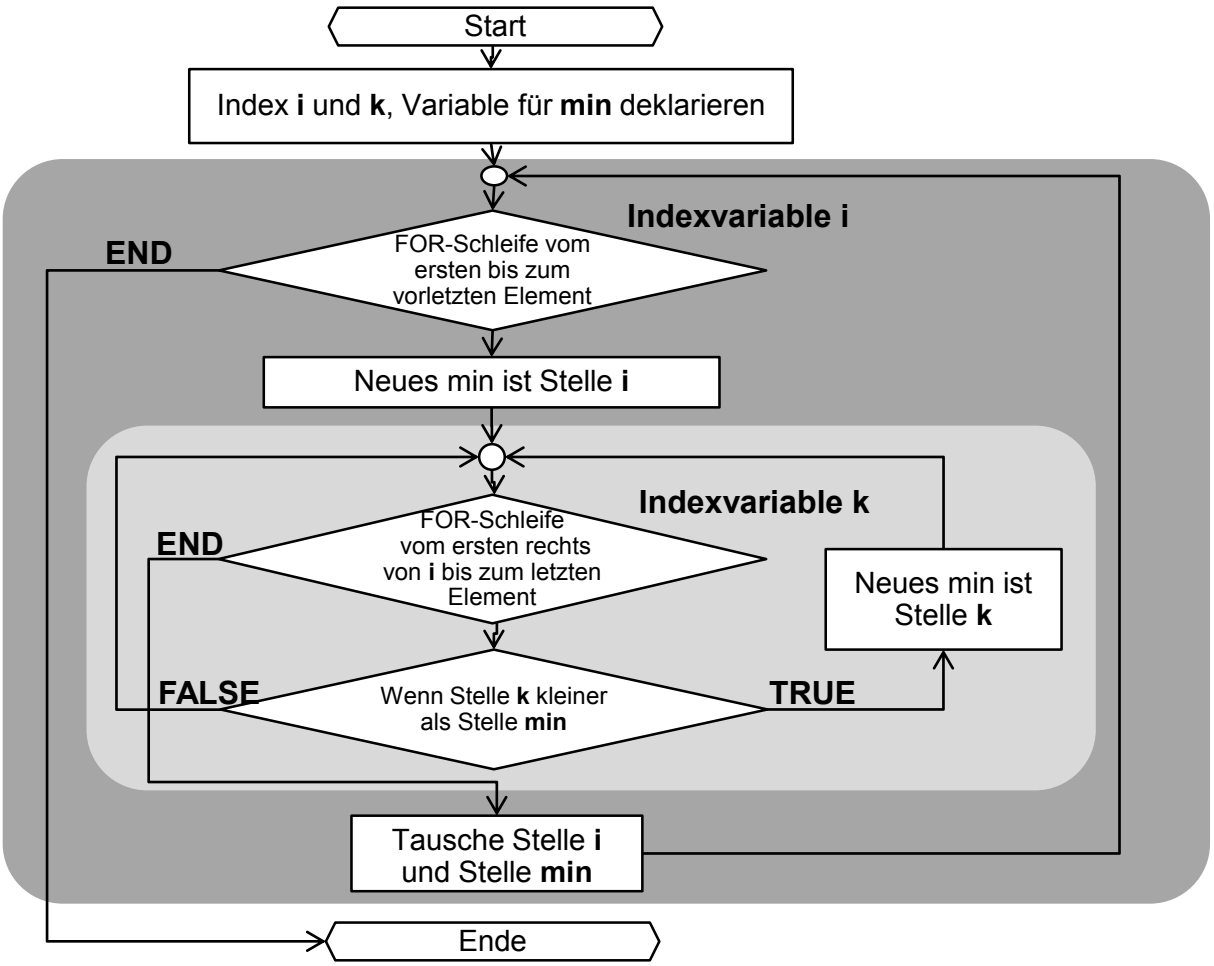
f) Zur Vorbereitung eines Sortier-Algorithmus soll im Folgenden nun eine Tauschfunktion mit dem Bezeichner „tausche" geschrieben werden. Der Aufruf dieser erfolgt per Übergabe zweier Zeigervariablen pbA und pbB, welche vom Typ BOHNE sind und auf die zu tauschenden Arrayplätze hinzeigen.

```
_______tausche(_____________________________)
{
 ____________________________ //Zwischenspeichern
 ____________________________
 ____________________________
 ____________________________
}
```

Nachname, Vorname _____ Matrikelnummer _____

Punkte

g) Abschließend soll eine Funktion implementiert werden, welche ein Array vom Typ BOHNE mit der Arraygröße „MAX“ sortieren kann. Hierbei sollen die Bohnen ihrem Gewicht nach *aufsteigend* (also leichteste Bohne zuerst) sortiert werden. Hierfür soll der sog. *Selection-Sort-Algorithmus* implementiert werden, welcher im Folgenden ausführlich in einem Flussdiagram beschrieben wird.
Hinweis: Der zu vervollständigende Lückentext für den Code ist auf der folgenden Seite.



Zum weiteren Verständnis der Wirkungsweise des Selection-Sort-Algorithmus, ist im Folgenden ein Beispiel für eine einfache Zahlenreihe durchgeführt. Die unterstrichenen Plätze sind endgültig und die Zahlen verbleiben an der Stelle:

Ausgangszustand:

3 9 2 7 1 (i=0) Startwert für min ist 3, Vergleich mit allen rechtsseitigen Zahlen
=> min = 1 => Tausche 3 und 1

Sortiervorgänge:

1 9 2 7 3 (i=1) 9 ist neues min, Vergleiche 9 mit allen Rechtsseitigen => Tausche 9 und 2
1 2 9 7 3 (i=2) Vergleiche 9 mit allen Rechtsseitigen => Tausche 9 und 3
1 2 3 7 9 (i=3) Vergleiche 7 mit allen Rechtsseitigen => 7 bleibt min, Selbsttausch

Endergebnis: **1 2 3 7 9**

Nachname, Vorname _____ Matrikelnummer _____

Punkte

Vervollständigen Sie nun unten die Funktion „selectionSort“ entsprechend der zuvor im Flussdiagramm beschriebenen Funktionsweise. Die Funktion wird mittels *call-by-reference* aufgerufen. Sie sortiert also direkt im Ziel-Array vom Typ BOHNE und gibt keinen Rückgabewert zurück.

Die verwendeten Schleifen benötigen die Größe des Arrays, welche durch den Präprozessor-Bezeichner „*MAX*“ bereits definiert wurde.

Die Arrayelemente sollen nach der Struct-Variable *iGewicht* aufsteigend sortiert werden.

Verwenden Sie zum Vertauschen die zuvor in Teilaufgabe f) angelegte Funktion „*tausche*“ durch einen korrekten Funktionsaufruf mittels *call-by-reference*, indem Sie ihr die beiden Tauschpartner (also die entsprechenden Arraystellen) als Adresse übergeben.

Hinweis: Beachten Sie die zugehörigen Kommentare im Code zur Vervollständigung.

```
_____ selectionSort(_____________________________)
{
    _____________________//Variablen deklarieren
    //Aeussere Schleife, Index i
    ____________________________________
    {
        ___________________________//Merken
        //Innere Schleife, Index k
        ___________________________
        {
         ____________________________________
         //Vergleich
        ____________________________//Merken
         }
      ___________________________________//Tauschen
    }
 }
```

Nachname, Vorname

Matrikelnummer

Punkte

5. Zeiger

Gegeben sei ein kurzes C-Programm, welches ein Array mit vier Elementen und die Zeigervariable *piZeiger* initialisiert. Die darauf folgenden Befehle führen jeweils zu einer Manipulation des Zeigers sowie der Werte im Array. Geben Sie die veränderten Werte des Arrays nach dem Ausführen des Codes an. Kreuzen Sie in der mittleren Spalte an, wo sich der Zeiger nach der Ausführung des Codes befindet.

```
#include <stdio.h>
int main ()
{
   int *piZeiger=NULL;
   int iAFeld[4]={7,1,6,2};

   piZeiger=&iAFeld[0];
   *piZeiger=6;
   piZeiger=(iAFeld+2);
   *(piZeiger)+=*(piZeiger-1)*2;
   *(piZeiger-1)=iAFeld[0]&10;
   *(piZeiger++)=*(piZeiger)+3;
return 0;
}
```

Array-Element	Zeiger	Wert des Elementes
iAFeld[0]		
iAFeld[1]		
iAFeld[2]		
iAFeld[3]		