

Vorname:	
----------	--

Nachname:	
-----------	--

Matrikelnummer:	
-----------------	--

Prüfung – Informationstechnik

Wintersemester 2013/14

07.03.2014

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält 30 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		Σ	Note
erreichte Punkte							
erzielbare Punkte	48	48	48	96		240	



Nachname, Vorname

Matrikelnummer

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 (10101011)₂ = (.....)₁₆ = (.....)₈

2 (6,625)₁₀ = (.....)₂

- b) Rechnen Sie die unten angegebene Zahl (angelehnt an IEEE 754) in eine Dezimalzahl um. Die Größe der Speicherbereiche für Vorzeichen, Mantisse und Exponent entnehmen Sie auch der folgenden Speicherdarstellung.

1	1	0	1	0	1	1	0	0	1	0	0	0
V					e (4 Bit)							
					M (8 Bit)							

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

Vorzeichen: V (+ oder -)

V =

Biased Exponent: e (in Dezimal)

e =

Bias: B; Exponent: E (in Dezimal)

B = E =

Mantisse: M (Denormalisiert)

M =

Dezimalzahl

Z =

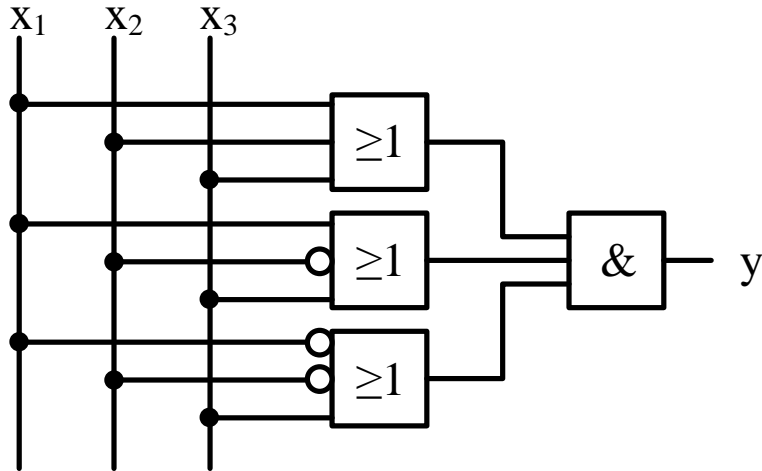


Nachname, Vorname

Matrikelnummer

2. Logische Schaltungen

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



- a) Füllen Sie die Wahrheitstabelle für das Ausgangssignal y aus, bei gegebenen Eingangsbelegungen der Variablen x_1 , x_2 und x_3 .

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

- b) Welcher Normalform entspricht die oben angegebene Schaltung?



Nachname, Vorname

Matrikelnummer

3. Normalformen und Minimierung

Gegeben ist die folgende Wahrheitstabelle.

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

- a) Tragen Sie die Ausgangsvariable y aus der Wahrheitstabelle in das KV-Diagramm ein und minimieren Sie die DNF (*Disjunktive Normalform*) mit Hilfe des KV-Diagramms. Schreiben Sie ebenfalls die minimierte Funktion in boolescher Algebra auf.

	x_1	\bar{x}_1	
x_2			
\bar{x}_2			
	\bar{x}_3	x_3	\bar{x}_3

- b) Minimieren Sie die folgende logische Formel durch Anwendung von *Rechenregeln der booleschen Algebra*. **Hinweis:** Tragen Sie Ihre Lösung in das angegebene Textfeld an! Nebenrechnungen werden nicht bewertet.

$$y = (a \wedge b \wedge c) \vee (a \wedge b \wedge \bar{c}) \vee (a \wedge \bar{b} \wedge c)$$

Lösung: $y =$

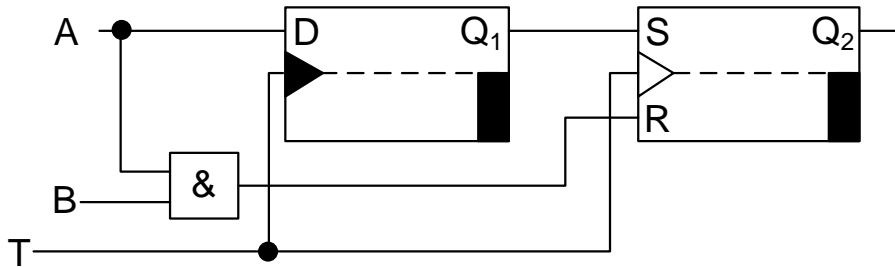


Nachname, Vorname

Matrikelnummer

4. Flip-Flops

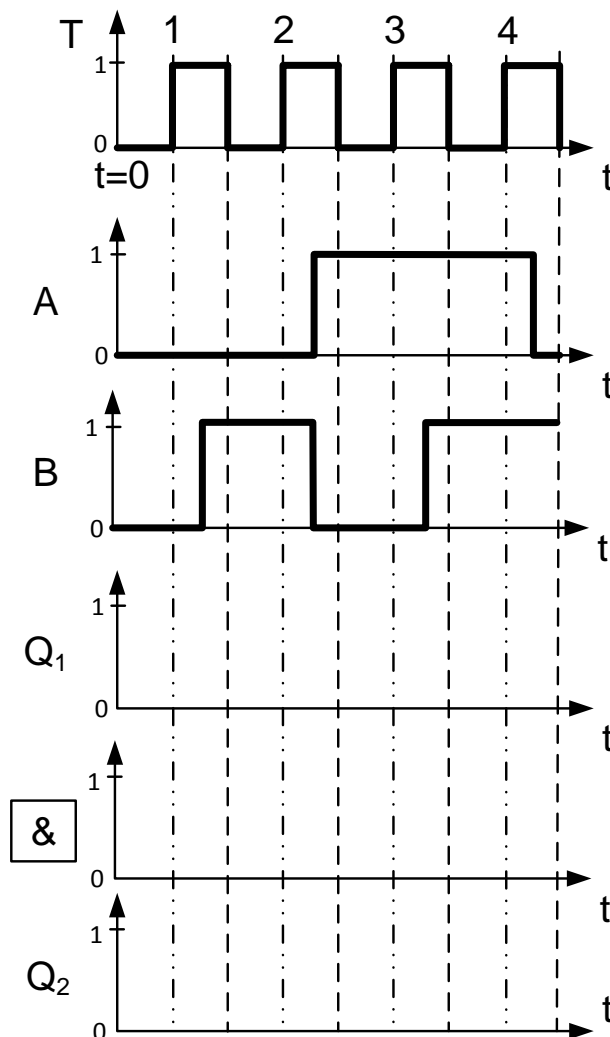
Gegeben sei die folgende Master-Slave-FlipFlop-Schaltung.



Bei $t = 0$ seien alle FlipFlops im folgenden Zustand: $Q_1 = 0$ und $Q_2 = 0$

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.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.





Nachname, Vorname

Matrikelnummer

5. Befehlsabarbeitung mit MMIX-Rechnerarchitektur

Im Registerspeicher eines MMIX-Rechners befinden sich die in Tabelle GL-4.3 (siehe folgende Seite) gegebenen Werte. Es sollen nacheinander die drei Befehle *0xE1 1D 01 B3*, *0x20 1E 1E 1F* und *0x19 1F 20 20* (Tabelle GL-4.4) abgearbeitet werden.

Wandeln Sie zunächst die Maschinenbefehle mit Hilfe der Tabelle GL-4.1 in Assemblersprache um und geben Sie die Bedeutung dieser Befehl an (ein Beispiel finden Sie in Tabelle GL-4.2). Führen Sie dann diese Befehle mit den Werten des Registerspeichers durch (Spalte „Wert vor Befehlsausführung“) und füllen Sie den Registerspeicher nach der Befehlsausführung vollständig aus (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	...
0xA_	STB	STB I		STBU I	STW	
	STT	STT I	...	STTU I	STO	...
...
0xE_	SETH	SETMH		INCH	INCHM	
	ORH	ORMH	...	ANDNH	ANDNMH	...
0xF_	JMP	JMP B		GETA	GETA B	
	POP	RESUME	...	SYNC	SWYM	...

Tabelle GL-4.1: MMIX-Code-Tabelle

Maschinensprache	Assemblersprache	Bedeutung
z. B. <i>0xAD 9B A1 24</i>	<i>STO \$0x9B, \$0xA1, \$0x24</i>	$M[\$0xA1 + \$0x24] = \$0x9B$

Tabelle GL-4.2: Lösungsbeispiel



 Nachname, Vorname

 Matrikelnummer

Registerspeicher		
Adresse	Wert <u>vor</u> Befehlsausführung	Wert <u>nach</u> Befehlsausführung
...
\$0x1D	0x00 00 00 00 00 00 00 A1	0x
\$0x1E	0x00 00 00 00 00 91 BA 0B	0x
\$0x1F	0x00 00 00 00 0B 12 11 05	0x
\$0x20	0x00 00 00 00 00 00 61 00	0x
...

Tabelle GL-4.3: Registerspeicher

Maschinensprache	Assemblersprache	Bedeutung
0xE1 1D 01 B3		
0x20 1E 1E 1F		
0x19 1F 20 20		

Tabelle GL-4.4: Von Maschinensprache zur Funktion



Nachname, Vorname

Matrikelnummer

Aufgabe BS: Betriebssysteme

Aufgabe BS:
48 Punkte

1. Scheduling

Sechs Prozesse (P1 bis P6) sollen mit einem Einkernprozessor abgearbeitet werden. Das Diagramm BS-1.1 zeigt die Zeiten, zu denen die Prozesse am Einkernprozessor eintreffen und die Ausführungszeiten der einzelnen Prozesse. Diese Prozesse sollen zur Laufzeit mit unterschiedlichen Schedulingverfahren eingeplant werden. Alle Schedulingverfahren beginnen zum Zeitpunkt $t = 0T$. Für die Schedulingverfahren, bei denen feste Prioritäten berücksichtigt werden müssen, ist in der Tabelle BS-1.2 die entsprechende Prioritätenverteilung (Prioritäten 1, 2 und 3) gegeben.

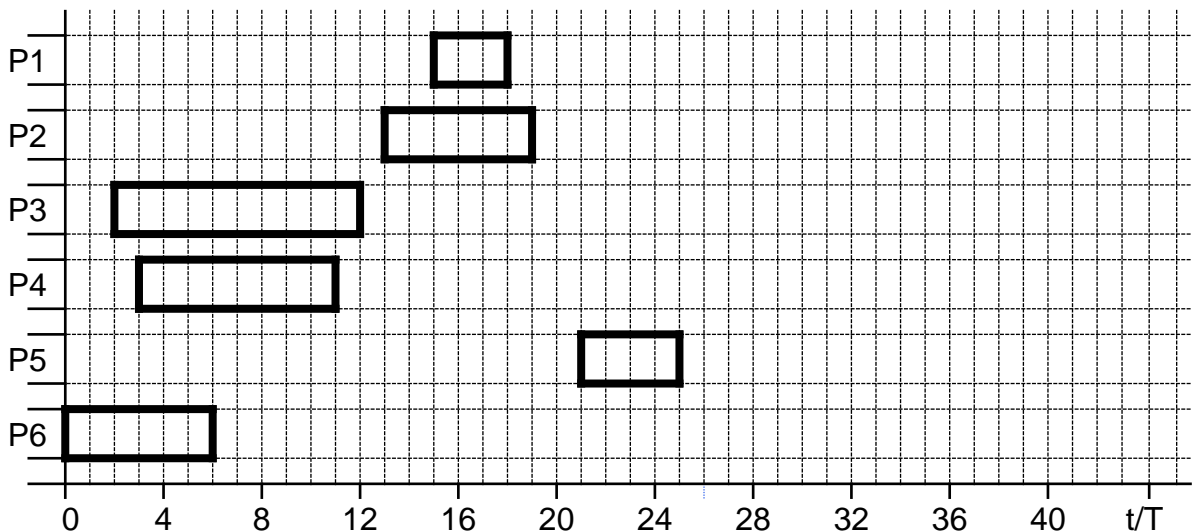


Diagramm BS-1.1: Sollzeitverlauf der Prozesse P1 bis P6

Prozess	P1	P2	P3	P4	P5	P6
Priorität	1 (hoch)	1	2	2	2	3 (niedrig)

Tabelle BS-1.2: Prioritätenverteilung

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

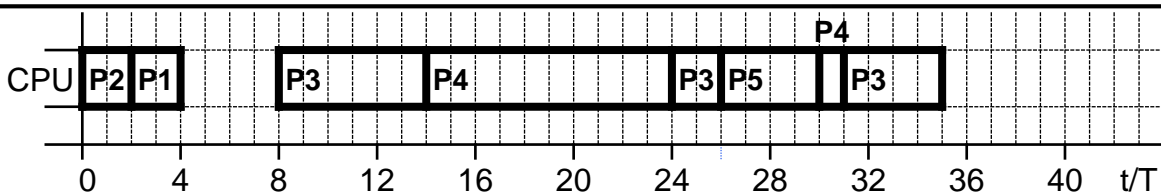


Diagramm BS-1.3: Lösungsbeispiel



Nachname, Vorname

Matrikelnummer

- a) In der ersten Teilaufgabe sollen die sechs Prozesse mit dem Round-Robin-Verfahren eingeplant werden. Neue Prozesse werden nach dem Zeitpunkt ihres Auftretens (RR mit FIFO) auf die Zeitscheibe gelegt. Für die Zeitscheibe soll angenommen werden, dass diese unendlich viele Schlitze besitzt und so zu jedem Zeitpunkt ausreichend freie Schlitze vorhanden sind. Die Zeitschlitze besitzen eine Länge von $4T$. Freie Zeiten können nicht übersprungen werden.

Tragen Sie den Verlauf der Prozessabarbeitung in das Diagramm BS-1.a ein.

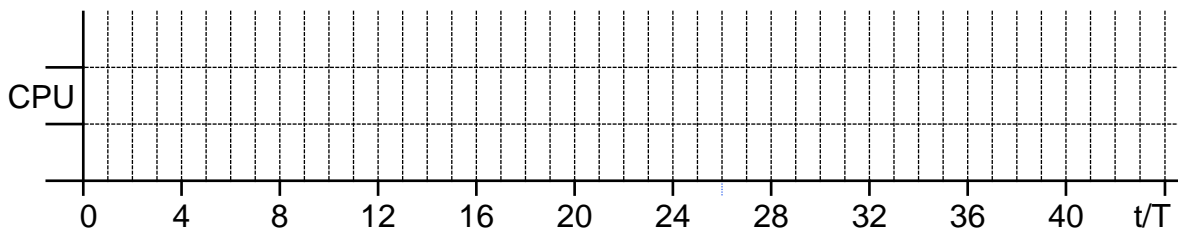


Diagramm BS-1.a: Round-Robin-Verfahren mit First In First Out (RR mit FIFO)

- b) In der zweiten Teilaufgabe soll die Einplanung der sechs Prozesse präemptiv nach ihren Prioritäten erfolgen. Dabei muss für jedes Prioritätslevel ein Round-Robin-Verfahren (Prio mit RR) verwendet werden. Neue Prozesse werden nach dem Zeitpunkt ihres Auftretens (RR mit FIFO) auf die Zeitscheibe gelegt. Für die Zeitscheiben soll angenommen werden, dass diese unendlich viele Schlitze besitzen und so zu jedem Zeitpunkt ausreichend freie Schlitze vorhanden sind. Die Zeitschlitze besitzen eine Länge von $4T$. Eine unterbrochene Zeitscheibe wird, wenn der Grund der Unterbrechung abgearbeitet worden ist, genau an der Stelle der Unterbrechung fortgeführt.

Tragen Sie den Verlauf der Prozessabarbeitung in das Diagramm BS-1.b ein.

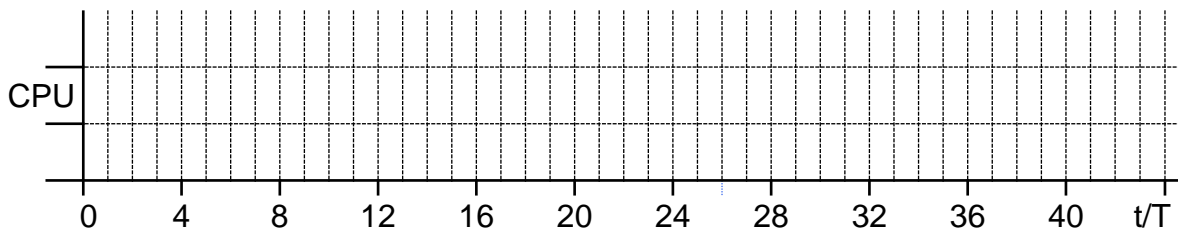


Diagramm BS-1.b: Prioritäten-Verfahren mit Round Robin (Prio mit RR)



Nachname, Vorname

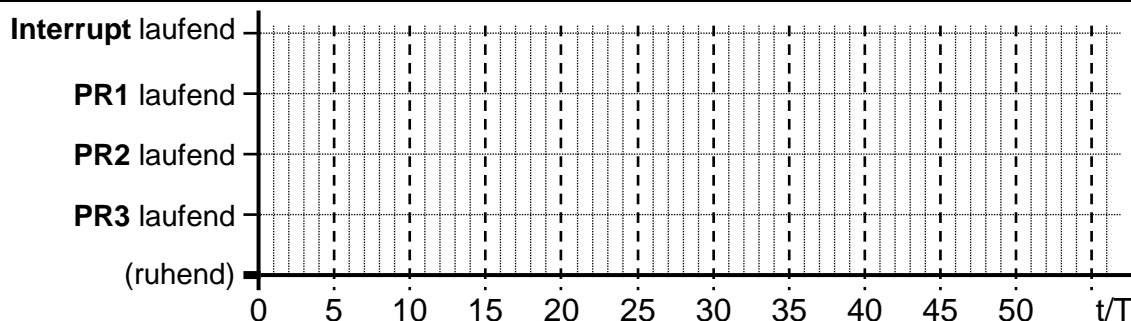
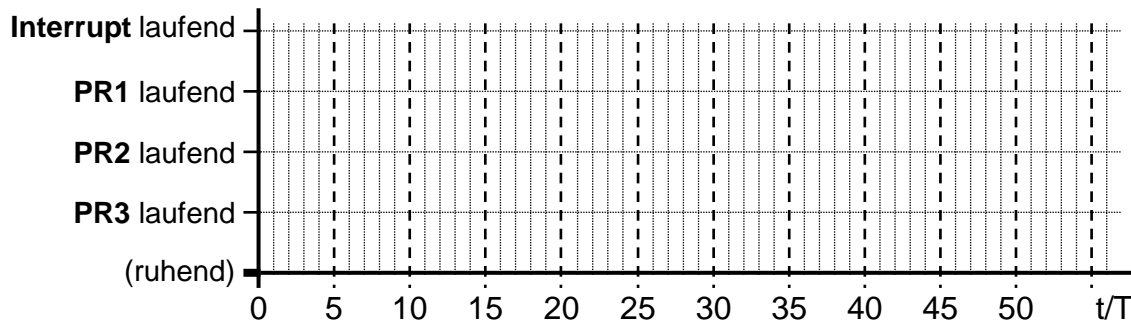
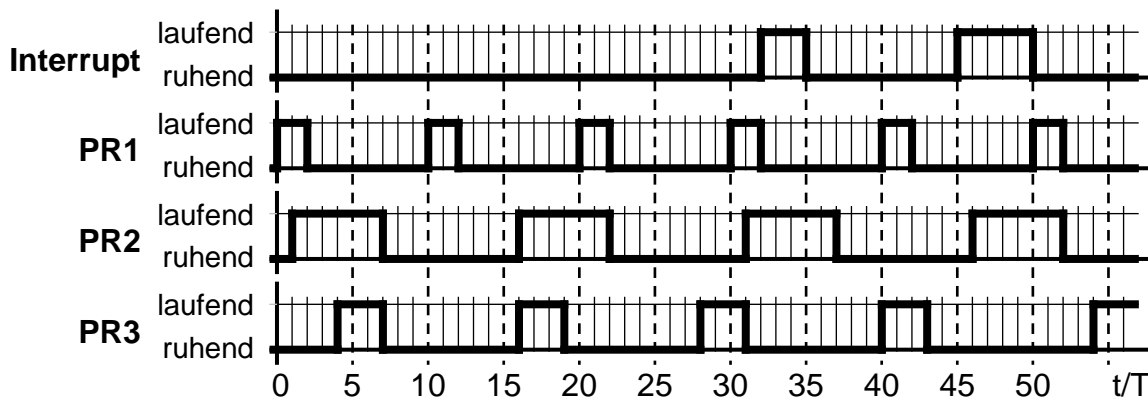
Matrikelnummer

2. Asynchrone Programmierung

Drei periodische Prozesse (PR1 bis PR3) sollen mit dem Verfahren der asynchronen Programmierung nicht-präemptiv auf einem Einkernprozessor eingeplant werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigste Priorität. Die Ausführung wird durch zwei Interrupts unterbrochen. Die Einplanung der Interruptroutinen erfolgt präemptiv. Nach dem Interrupt kehrt der Scheduler an die die zuvor verlassene Stelle zurück.

Tragen Sie in das unten angegebene leere Diagramm den Verlauf der Abarbeitung von Programmen und Interrupts ein.

Hinweis: Bei größeren Korrekturen verwenden Sie bitte das Ersatzfeld und markieren das zu wertende Lösungsfeld. Beide Hälften des Diagramms sowie jede Zeile werden unabhängig gewertet.



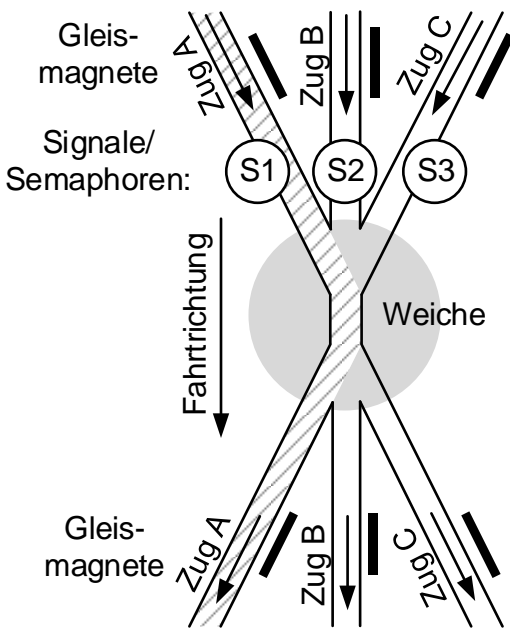


Nachname, Vorname

Matrikelnummer

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: $\overline{A}BC\overline{B}$.



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

Hinweis: Die Taskreihenfolge muss durch die Semaphoroperationen eindeutig festgelegt sein. Die für die Ausführung eines Tasks notwendigen Semaphoren sollen nur im Block verwendet werden. Beispielsweise würde ein Task X mit folgenden Semaphoroperationen

Task	X
Semaphoroperationen	P(S7)
	P(S7)
	P(S8)
	...
	V(S9)

nur starten, wenn alle drei benötigten Semaphoren (S7, S7, S8) gleichzeitig frei sind.

Semaphore:	S1	S2	S3
Initialwert:			
Task:	A	B	C
Semaphoroperationen			

sich wiederholende Folge der Zugklassen: $\overline{A}BC\overline{B}$



Nachname, Vorname

Matrikelnummer

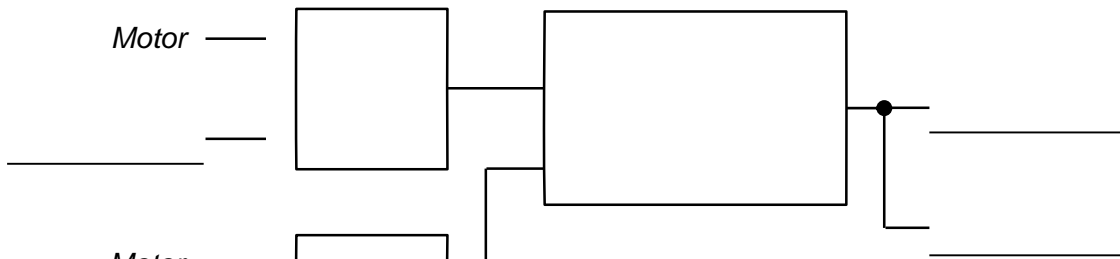
4. IEC 61131-3: Funktionsbausteinsprache (FBS)

Eine Deckenventilator soll durch einen *Seilzug* gesteuert werden. Mit dem *Seilzug* kann der *Motor* geschaltet und die *Drehrichtung* geändert werden. Wird der *Seilzug* betätigt (*Seilzug* = 1), während der *Motor* aus (*Motor* = 0) ist, schaltet der *Motor* ein. Wird der *Seilzug* betätigt, während der *Motor* an ist, wird dieser ausgeschaltet. Die Steuerung reagiert auf den *Seilzug* nur im Moment des Anziehens. Wenn der *Motor* läuft, soll auch das *Licht* eingeschaltet sein (*Licht* = 1). Die *Drehrichtung* wird im Moment des Abschaltens des Motors geändert. Wird der *Motor* durch den *Seilzug* abgeschaltet und die *Drehrichtung* war „1“ (*Drehrichtung* = 1: Rechtslauf), wird die *Drehrichtung* auf „0“ (*Drehrichtung* = 0: Linkslauf) gesetzt. Ist die *Drehrichtung* im Moment des Abschaltens „0“, wird die *Drehrichtung* auf „1“ gesetzt.

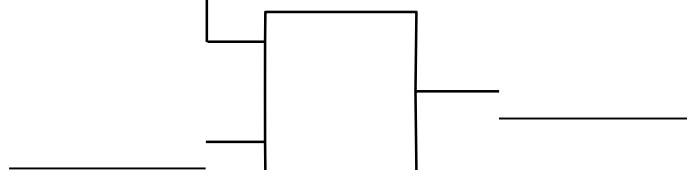
Vervollständigen Sie den untenstehenden Funktionsbausteinplan mit den fehlenden Variablennamen, Funktionsblockbezeichnungen und Verbindungen.

Hinweis: Um den Block für die *Drehrichtung* leichter bestimmen zu können, können Sie sich eine Wahrheitstabelle für diesen Block skizzieren.

Motor- und Lichtsteuerung



Richtungssteuerung





Nachname, Vorname

Matrikelnummer

Aufgabe MSE Teil 1: Automaten**Aufgabe MSE1:**
24 Punkte**1. Theorie**

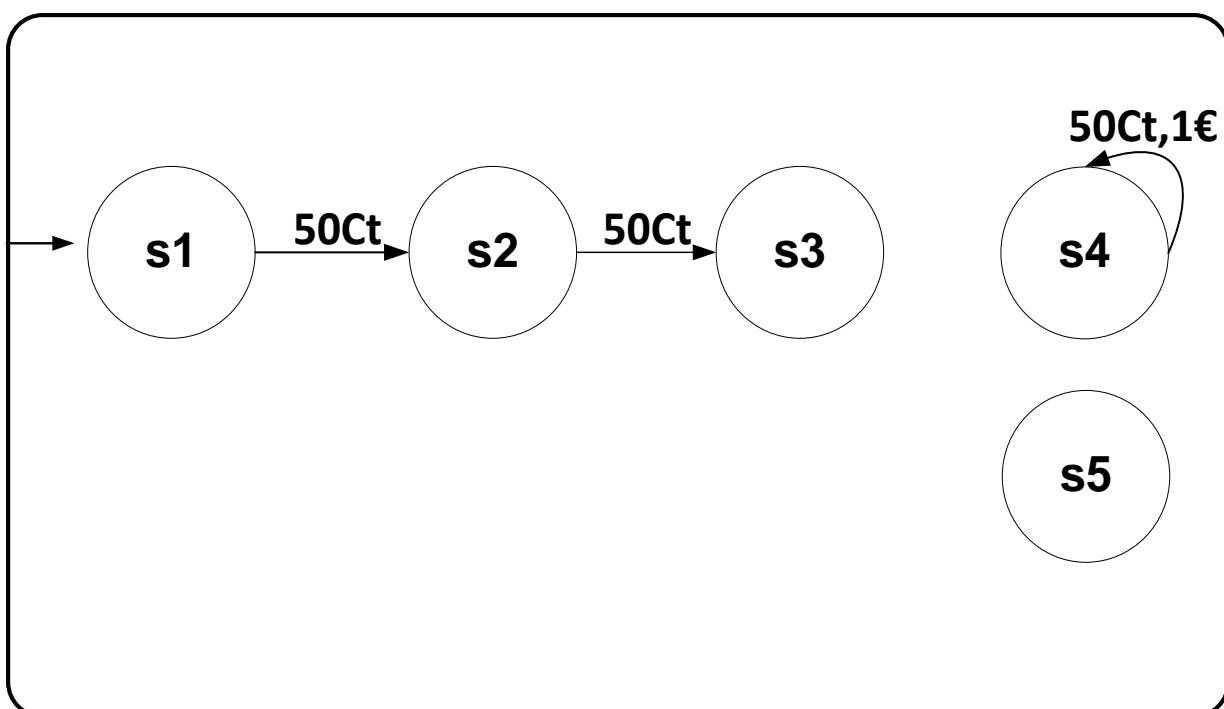
Nennen Sie zwei Darstellungsformen für Automaten

2. Akzeptor

Gegeben sei folgender unvollständiger Automat für einen Münzeinwurf. Ergänzen Sie den Automaten so, dass er nur Eingaben akzeptiert, deren Summe genau den Wert 1,50€ haben.

Hinweise:

- Als Eingaben werden nur 50Ct-Stücke (50Ct) und 1-Euro Stücke (1€) betrachtet.
- Fügen Sie Zustandsübergänge hinzu, soweit es für die Aufgabenstellung nötig ist.
- Fügen Sie keine neuen Zustände ein, bzw. streichen Sie keine bestehenden Zustände.
- Markieren Sie den Endzustand entsprechend der Notation.





Nachname, Vorname

Matrikelnummer

3. Turing-Maschine

Gegeben ist die Anweisungsliste, sowie das Ein/Ausgabe-Band einer Turing-Maschine. Zu Beginn des Programmablaufs befindet sich der Schreib-/Lese-Kopf an der markierten Position. **Hinweis:** Der Programmablauf beginnt mit Anweisung 1.

	Lese	Schreibe	Gehe nach	Nächste Anweisung
Anweisung 1	0	0	Rechts	1
	1	0	Rechts	2
Anweisung 2	0	1	Links	0
	1	0	Rechts	2

0	0	1	1	0	0
---	---	---	---	---	---



Geben Sie in den vorgegebenen Kästen die Besetzung des Ein/Ausgabe-Bandes nach dem Ausführen der jeweiligen Anweisung an, und markieren Sie an welcher Stelle sich der Schreib-/Lese-Kopf befindet.

--	--	--	--	--	--

--	--	--	--	--	--

--	--	--	--	--	--

--	--	--	--	--	--

--



Nachname, Vorname

Matrikelnummer

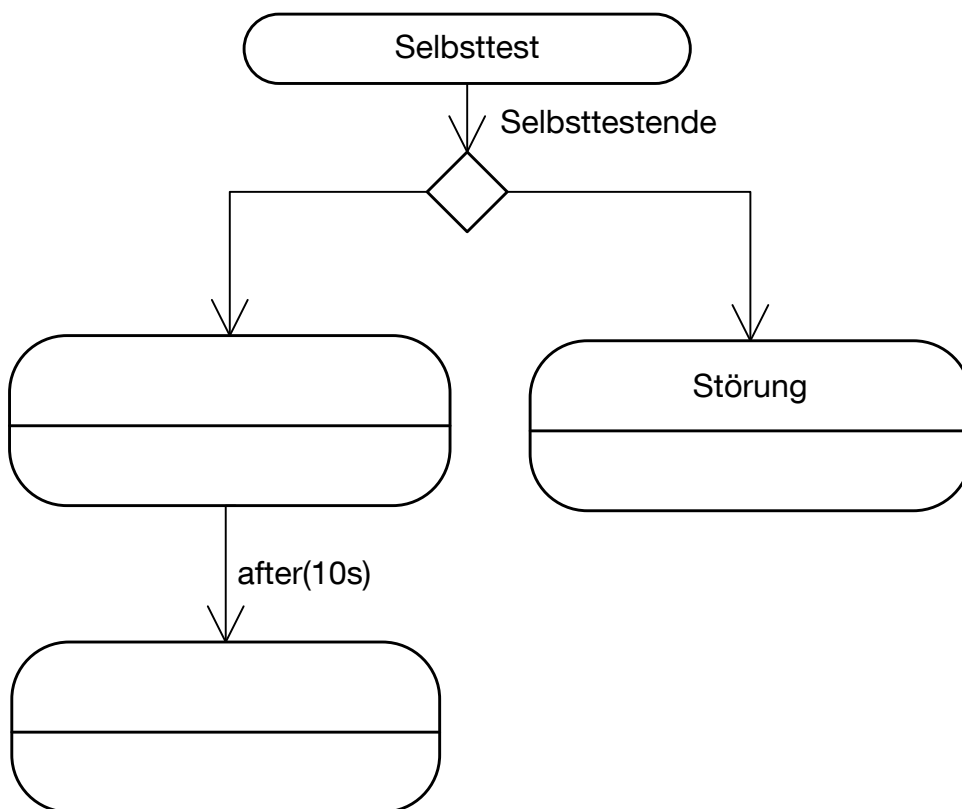
4. Zustandsdiagramm

Der Hochlauf einer Maschine soll als Zustandsdiagramm modelliert werden. Dafür wird das System als erstes in den Zustand *Selbsttest* versetzt. Nachdem das Selbsttestende erreicht wurde, wird überprüft ob das Ergebnis (Variable *Result*) des Selbsttests wahr oder falsch war.

Falls das Ergebnis falsch war, wechselt das System in den Zustand *Störung*. Beim Betreten des Zustandes soll dabei die Aktion *Störungsmeldung* ausgeführt werden, danach geht das System in den Endzustand über.

Falls der Selbsttest ein wahres Ergebnis hatte, soll in Zustand *Grundstellung* übergegangen werden. Während dieser Zustand aktiv ist, soll die Aktion *ZylinderEinfahren* durchgeführt werden. Nach 10 Sekunden soll das System dann in den Zustand *Betriebsbereit* wechseln und bei Zustandseintritt ein Protokoll senden (Aktion *ProtokollSenden*). Das System geht dann ebenfalls in den Endzustand über und der Hochlauf ist abgeschlossen.

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





Nachname, Vorname

Matrikelnummer





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

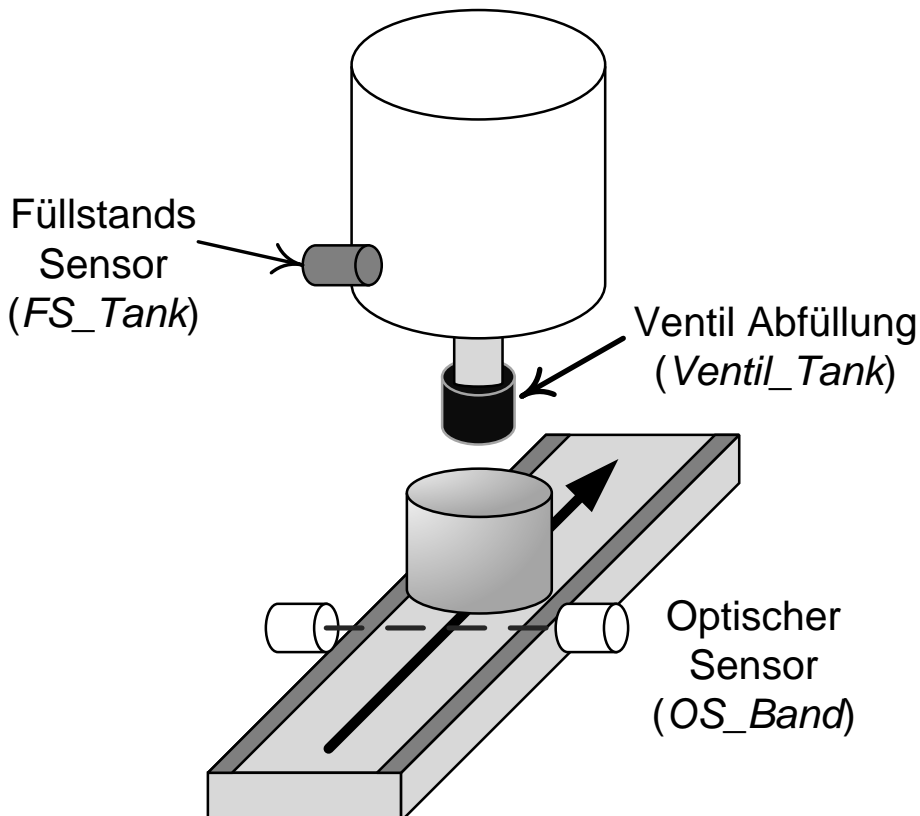
Aufgabe MSE2:
24 Punkte

Gegeben ist ein Abfüllsystem, in welchem *Behälter* befüllt werden sollen. Auf einem Transportband erkennt ein optischer Sensor (*OS_Band*), ob ein Behälter zum Abfüllen vorhanden ist. In dem Tank, in dem die Flüssigkeit gelagert ist, überprüft ein Füllstands-Sensor (*FS_Tank*), ob genügend Flüssigkeit vorhanden ist. Wenn dies der Fall ist, wird das Abfüll-Ventil (*Ventil_Tank*) geöffnet und die Flüssigkeit in den Behälter abgefüllt.

Das Abfüllsystem soll in den folgenden Aufgaben mittels Strukturierter Analyse/ Real-Time (SA/RT) modelliert werden.

Legende

-  Optischer Sensor
-  Füllstands-Sensor
-  Ventil
-  Behälter





Nachname, Vorname

Matrikelnummer

1. Kontextdiagramm

Es soll das Kontextdiagramm (Datenkontext- und Steuerkontextdiagramm in einem Diagramm) des Prozesses *Behälter befüllen* modelliert werden.

Der *Anlagenbediener* erteilt *Kommandos* zur *Prozesssteuerung*. Die *SPS* (Speicherprogrammierbare Steuerung) liefert dem Prozess die zur Berechnung notwendigen *Sensorwerte* und empfängt die berechneten *Aktorwerte*. Ein *Nebensystem* (ein dem Abfüllvorgang nachgelagertes Systemmodul) empfängt *Steuervorgaben* vom Prozess.





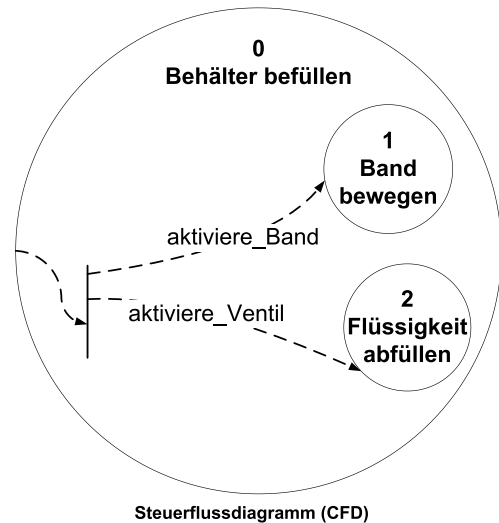
Nachname, Vorname

Matrikelnummer

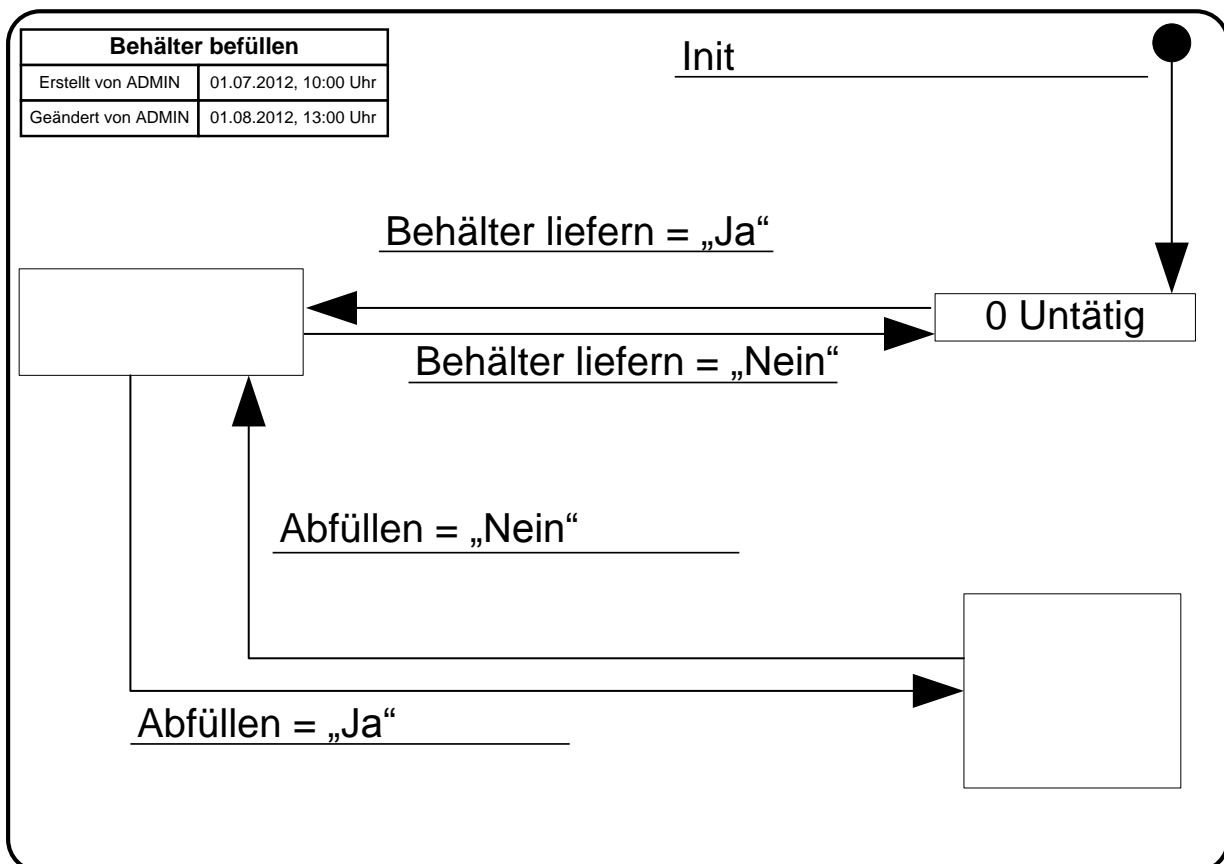
2. Steuerspezifikation (CSPEC)

Für den Prozess *Behälter befüllen* besteht nebenstehendes Steuerflussdiagramm (CFD). Zu diesem Steuerflussdiagramm ist die unvollständige Steuerspezifikation (CSPEC) gegeben.

Am Anfang (*Init*) werden alle Prozesse deaktiviert (= *false*) und das System geht in den Zustand *Untätig*. Wenn ein Behälter an das Abfüllsystem geliefert werden soll, (*Behälter liefern* = „Ja“) muss das Band aktiviert (= *true*) werden. Wenn der Behälter an der Abfüllstation ist und der Abfüllvorgang beginnen soll (*Abfüllen* = „Ja“), müssen das Band deaktiviert und das Ventil aktiviert werden. Wenn der Abfüllvorgang beendet werden soll (*Abfüllen* = „Nein“), müssen das Band aktiviert und das Ventil deaktiviert werden. Wenn sich das Band bewegt und es sollen keine weiteren Behälter geliefert werden (*Behälter liefern* = „Nein“) muss das Band deaktiviert werden und das System geht in den Zustand *Untätig*.



Ergänzen Sie in der CSPEC die Zustände und Aktionen. Verwenden Sie dabei die Bezeichnungen aus dem Steuerflussdiagramm. Zeichnen Sie keine zusätzlichen Zustände ein.





Nachname, Vorname

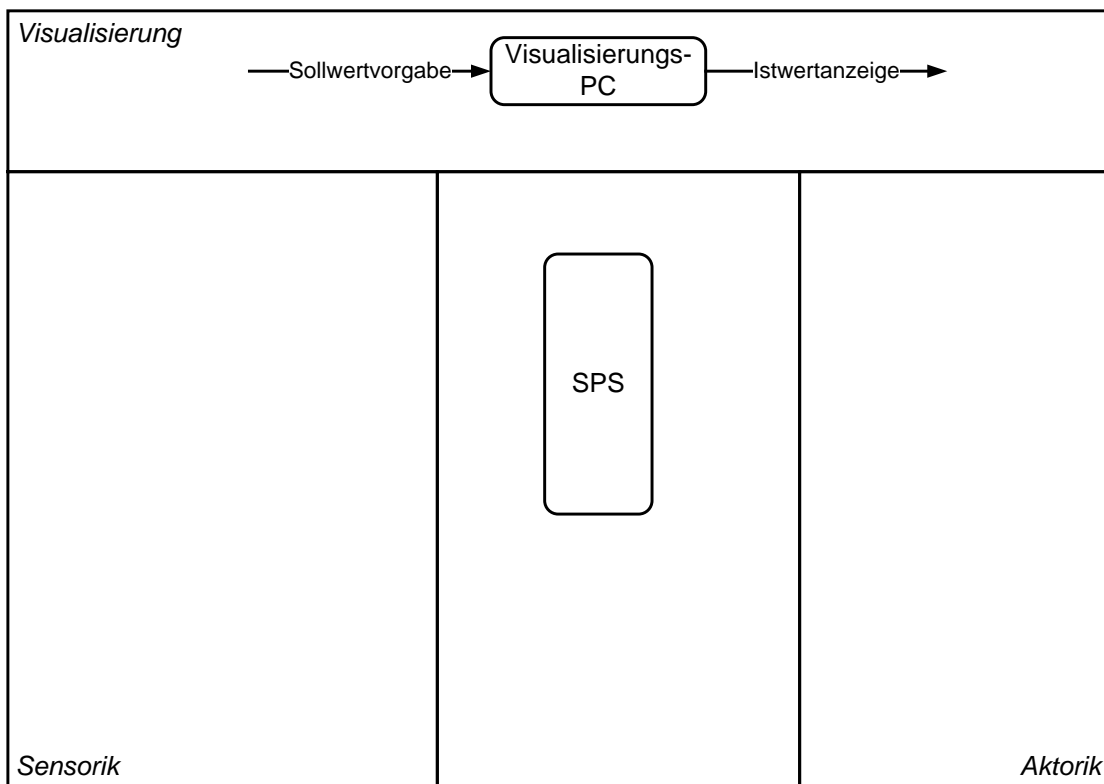
Matrikelnummer

3. Architekturverbindungsdiagramm

Das Abfüllsystem wird von einer einzelnen *SPS* gesteuert. Hierzu soll das Architekturverbindungsdiagramm modelliert werden, um zu zeigen, wie die einzelnen Elemente des Systems miteinander verbunden sind. Fügen Sie in das angegebene, unvollständige Architekturverbindungsdiagramm die folgenden Aktorik-/Sensorik-Module hinzu sowie die (physikalischen) Verbindungen zwischen den Architekturmodulen und der *SPS* (Flüsse von der Umgebung zu den Sensoren bzw. von den Aktoren zu der Umgebung müssen nicht modelliert werden):

- **Sensorik:** *OS_Band*, *FS_Tank*
- **Aktorik:** *Ventil_Tank*

Die *Sensorik* liefert jeweils einen *Istwert (IW)* an die *SPS*. Die *Aktorik* empfängt jeweils einen *Stellwert (SW)*. Die *SPS* tauscht die *Soll- und Istwerte (SW/IW)* mit der *Sensorik* und *Aktorik* über eine *Profinet*-Verbindung aus. Zusätzlich ist die *SPS* über eine *Ethernet*-Verbindung mit dem Visualisierungs-PC verbunden und tauscht mit diesem die *Soll- und Istwerte (SW/IW)* aus. Zeichnen und beschriften Sie die jeweiligen Verbindungen der Module entsprechend.





Nachname, Vorname

Matrikelnummer

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;`).

_____ __Farben_____ = {'g','y','r'};

_____ __Dauer_____ = 10.5;

_____ __Anzahl_____ = 3;

- b) Die Gleitkommazahl **Dauer** soll eingelesen werden und zur Überprüfung mit einer Dezimalstelle ausgegeben werden. Ergänzen Sie den Quelltext.

_____ __Dauer = 0;

_____ ("Bitte geben Sie ein Zeit in Sekunden ein: ");

if(_____ (_____) == 0)
 return 1; // Beendet die Funktion bei unzulässiger Eingabe

_____ ("\nDie eingestellte Dauer ist: _____ ", _____);

- c) Werten Sie folgende Ausdrücke nach den Regeln der booleschen Algebra aus. Geben Sie das Ergebnis als Dezimalzahl an. Verwendete Variablen:

Int-Variablen: X = 9; Y = 3;

Float-Variable: Z = 0.75;

Char-Variable: A = 0x12;

$(X Y) == (A - 7)$	
$((X \& A) \& X)$	
$(A \& (\text{char}) Y)$	
$((\text{float})(Y/Z) == 3)$	



Nachname, Vorname

Matrikelnummer

2. Kontrollstrukturen

- a) Das Programm zur Steuerung einer Ampel an einer Kreuzung von zwei Straßen, Straße A kreuzt Straße B, soll um einen intelligenten Umschaltmechanismus erweitert werden. Zugrunde gelegt wird dabei der Quotient der Verkehrsdichte beider Straßen ($fDichteQ = fDichte_A / fDichte_B$):

- Bei einer Verkehrsdichte von mehr als 200 Fahrzeugen pro Stunde auf einer der beiden Straßen soll die Dauer der Grün-Phase von Straße A mit Verkehrsdichtenquotienten ($fDichteQ$) und Straße B mit dem inversen Verkehrsdichtenquotienten ($1/fDichteQ$) multipliziert werden.
- Sofern der erste Fall nicht zutrifft und die Verkehrsdichte der Straßen jeweils unter 50 Fahrzeuge pro Stunde fällt, soll die Dauer der Grün-Phase der jeweiligen Straße auf 50% reduziert werden.

Gehen Sie davon aus, dass der Code zum Einlesen der Verkehrsdichte der beiden Straßen bereits realisiert wurde und die zugehörigen Variablen belegt wurden. Manipulieren Sie die Variable für die Dauer der Grün-Phase $fDauer_A$ und $fDauer_B$ für die genannten Fälle.

Nutzen Sie hierfür eine *IF-ELSE* Abfrage.

```
#include <stdio.h>
enum STRASSE { A , B };
void main(){
    float fDauer_A = 50, fDauer_B = 75;
    float fDichte_A = einlesenVerkehrsdichte( A );
    float fDichte_B = einlesenVerkehrsdichte( B );
    float fDichteQ = fDichte_A / fDichte_B;

    //Erweiterte Grün-Phasen-Dauer-Berechnung mit IF-ELSE
```

```
    return;
```

```
}
```



Nachname, Vorname

Matrikelnummer

- b) Für die Berechnung der durchschnittlichen Verkehrsdichte in einem gewünschten Intervall müssen aus der Datenbank die letzten `iAnzahl` Messwerte (Anzahl Fahrzeuge pro Minute) mit einer FOR-Schleife nacheinander mit der Funktion `leseAnzahlFahrzeuge(int iMesswert)` gelesen und summiert werden. Die Funktion gibt bei Übergabe von „0“ den letzten (bei „1“ den vorletzten, etc.) gemessenen Wert zurück. Im Anschluss muss die Summe der Fahrzeuge durch die Anzahl der Messwerte geteilt und mit 60 multipliziert werden, um die durchschnittliche Verkehrsdichte in Fahrzeuge pro Stunde zu erhalten.

Ergänzen Sie hierfür die folgende Funktion `berechneVerkehrsdichte()`:

```
#include <stdio.h>
```

```
int leseAnzahlFahrzeuge(int iMesswert)
{ ... } // Anzahl Fahrzeuge für jeweilige Minute
```

```
float berechneVerkehrsdichte(int iAnzahl)
{
    // Berechnung der Verkehrsdichte für iAnzahl
```

```
}
```

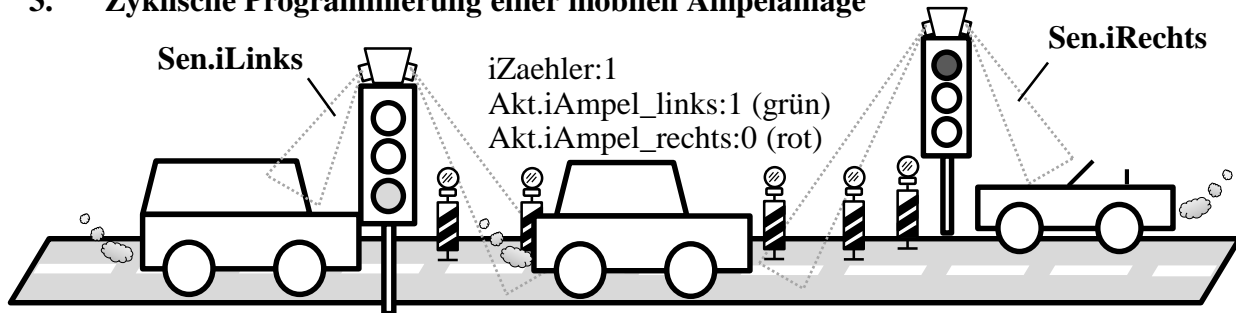
```
void main(void) // Main-Programm
{
    // Beispiel fuer Mittelung
    // über die letzten 5 Messwerte
    float fDichte = berechneVerkehrsdichte(5);
}
```



Nachname, Vorname

Matrikelnummer

3. Zyklische Programmierung einer mobilen Ampelanlage



Die Firma ISA GmbH stellt eine Ampelanlage her, welche für Straßen genutzt wird, die auf eine Spur verengt sind. Die Ampelsteuerung kann dazu die beiden Ampeln auf „grün“ (=1) und „rot“ (=0) schalten, um den Verkehr zu regeln. Um die Fahrzeuge links, in der Verengung und rechts davon zu erkennen, ist Sensorik angebracht. Jeweils ein Sensor erkennt, ob links oder rechts ein Auto wartet. Die Sensorwerte im verengten Bereich werden nicht direkt verwendet, da eine hier nicht betrachtete Funktion die Anzahl der Fahrzeuge in der Verengung zyklisch aktualisiert und in die Variable *iZaehler* speichert. Die Ampelanlage soll im Haltzustand auf beiden Seiten rot sein.

Übergangsbedingung von (0) „Alle Anhalten“ zu (1) „Links fahren“/(2) „Rechts fahren“: Falls ein Fahrzeug erkannt (I) und die verengte Fahrbahn frei ist (II), soll die entsprechende Seite grün schalten. Zusätzlich soll mit Hilfe der Merkvariable „*iWechseln*“ („links“=1, „rechts“=2) stets zwischen den Seiten gewechselt werden (III). Direkt nach Einschalten der Ampel wird *iWechseln*=1 (links) initialisiert (siehe Kopf des Codes), um bei Fahrzeugen links und rechts gleichzeitig keinen Konflikt zu erzeugen. Für den Fall, dass nach dem Einschalten nur rechts Fahrzeuge warten, muss daher im Übergang 0→2 (rechts) noch die Bedingung, dass auf der linken Seite kein Auto wartet (IV) durch ODER-Verknüpfung mit „*iWechseln*“ geprüft werden, um daraufhin trotzdem rechts grün zu schalten.

Übergangsbedingung von (1) „Links fahren“/(2) „Rechts fahren“ zu (0) „Alle Anhalten“: Fahren schließlich die Autos auf einer Seite, dann soll nach Ankunft eines Fahrzeugs auf der anderen Seite und wenn schon mindestens 40 Sek. grün war, gewechselt werden. Dazu muss immer erst in den Haltzustand geschaltet werden.

Aktoren:	
Akt.iAmpel_links	Schaltet links „grün“=1 bzw. „rot“=0
Akt.iAmpel_rechts	Schaltet rechts „grün“=1 bzw. „rot“=0
Sensoren/Merkvariablen:	
Sen.iLinks	Erkennt ein stehendes Auto links (=1)
Sen.iRechts	Erkennt ein stehendes Auto rechts (=1)
iZaehler	Enthält die Anzahl der Fahrzeuge in der Strecke
vplcZeit	Systemlaufzeit der Steuerung (in ms)
t	Variable für eine Zeit (z.B. für einen Timer)
state	Speichert den aktuellen Zustand
iWechseln	Merkt welche Seite als nächstes kommt („links“=1, „rechts“=2)



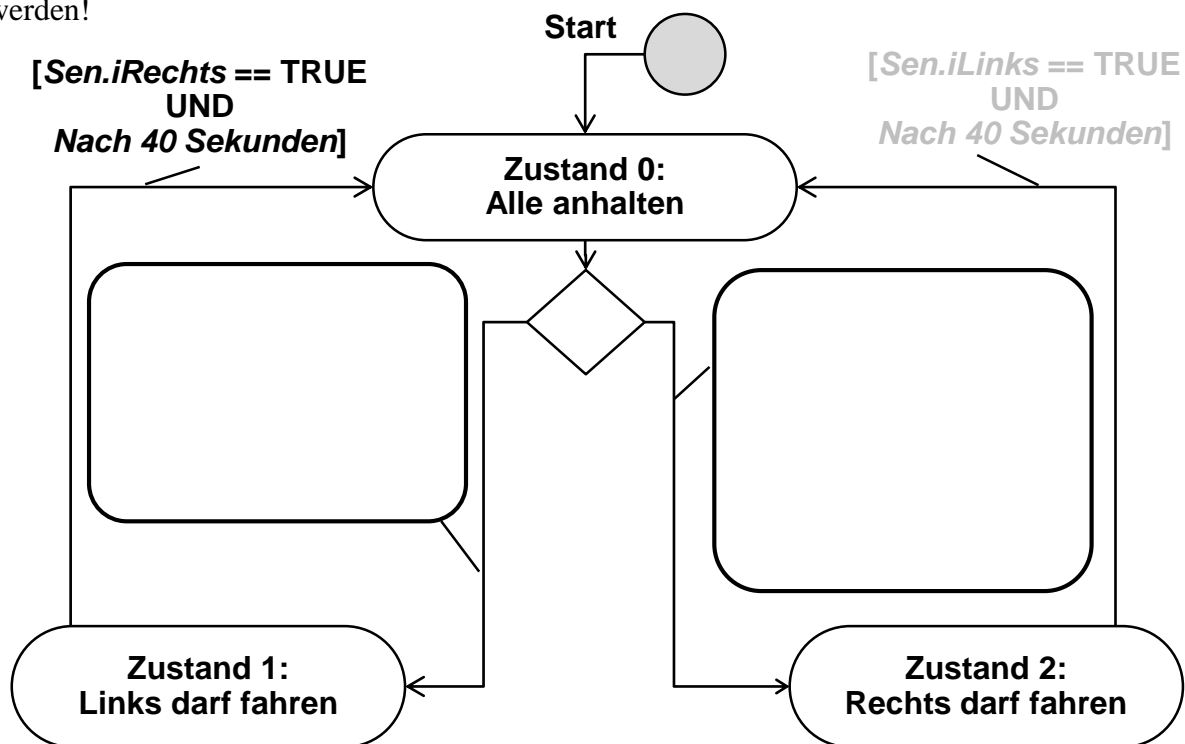
Nachname, Vorname

Matrikelnummer

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

- Vervollständigen Sie zunächst in diesem die Übergangsbedingungen von $0 \rightarrow 1$ und $0 \rightarrow 2$ entsprechend der oben beschriebenen Funktionsweise.
- Übertragen Sie das Zustandsdiagramm mit Übergangsbedingungen und Implementierung der beschriebenen Funktionalität in den C-Code auf der folgenden Seite. Benutzen Sie dafür die Variablen auf der vorherigen Seite. Beachten Sie, dass der Code zyklisch ausgeführt wird. Der Kopf des Programms mit Deklaration und Initialisierung aller Variablen ist unten dargestellt und muss nicht realisiert werden.

Hinweis: Die Transition $2 \rightarrow 0$ ist im Code nicht dargestellt und muss nicht realisiert werden!



```
#include "eavar_mobile_ampel.h"
```

```

struct SData Sen;           // Sensorvariablen
struct AData Akt;           // Aktorvariablen
unsigned int vplcZeit=0;    // 1000 entspricht 1 Sek.
int iWechseln=1;            // Merkvariable nächste Seite
                             // (nach Einschalten links bevorzugt)
unsigned int t=0;           // Merkvariable zum Speichern der
                             // Laufzeit
int state=0;                // Merkvariable für den aktuellen
                             // Zustand
int iZaehler=0;             // Aktuelle Anzahl der Fahrzeuge in
                             // der Strecke
(...)
  
```


_____
Nachname, Vorname_____
Matrikelnummer

```
int plcZyklus()
{
    switch(state)
    {
        _____// Zustand 0: Alles anhalten
        _____
        _____

    if( _____
        _____) state=1;

    if( _____
        _____) state=2;
        break;

        _____// Zustand 1: Links fahren
        _____ // Aktor setzen
        _____ // Wechseln setzen

    if(!t) _____ // Timer

        else if( _____)
        {
            t=0; //Timer zurücksetzen
            state = 0; // Transition 1→0
        }
        break;

        _____// Zustand 2: Rechts fahren
        _____ // Aktor setzen
        _____ // Wechseln setzen

    (...) // Transition von 2→0 (hier nicht ausgeführt)
```



 Nachname, Vorname

 Matrikelnummer

4. Datenbank mit Abflugzeiten an einem Flughafen

Für Fluglotsen betreibt der Flughafen München eine hochaktuelle Datenbank mit den Abflugdaten in den nächsten 60 Minuten. Dies ist notwendig, um die Flugzeuge sicher auf eine freie Flugbahn lotsen zu können.

Die Datenbank benötigt daher einen speziellen Datentypen, welcher exakt die benötigten Informationen in einem Array speichern kann.

Airline	Ziel	Plan	Bahn
LH	Frankfurt	1002	C
AB	Hamburg	1005	A
GW	Amsterdam	0955	B
...

Der Datentyp soll dabei folgende Spezifikationen speichern können:

- Airline (im betrachteten Terminal-Abschnitt starten nur die Airlines mit den Kürzeln LH, AB und GW)
- Ziel (max. 50 Buchstaben)
- Geplanter Abflug (4-stellige Zahl, 1002 entspricht 10 Uhr und 2 Minuten)
- Startbahn (Buchstaben von A-F möglich)

- a) Zur vereinfachten Handhabung soll zunächst ein neuer Datentyp „AIRLINE“ definiert werden, welcher die Werte ‚LH‘, ‚AB‘ und ‚GW‘ enthalten kann:

- b) Weiterhin soll nun die Struktur für den Datentyp „Flug“ definiert werden, welche die oben genannten Daten für die jeweiligen Flüge speichern kann:



Nachname, Vorname

Matrikelnummer

- c) Der Datentyp Flug kann nun sowohl für Ankünfte und Abflüge verwendet werden. Um diese zu speichern, wird nun ein Array benötigt, welches jeweils 30 Einträge für Ankünfte in der ersten Zeile und Abflüge in der zweiten Zeile speichern kann. Initialisieren Sie das Array mit der Bezeichnung „AlleFluege“ korrekt und geben Sie dem Abflug an der Stelle [0][0] im Array folgende Nutzdaten:

Airline	LH
Ziel	New York
Plan	1230
Bahn	C

- d) Für die Bearbeitung der Nutzdaten sind verschiedene Algorithmen vorgesehen. Um die Größe weiterer Arrays 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 zunächst vier Plätze fest.

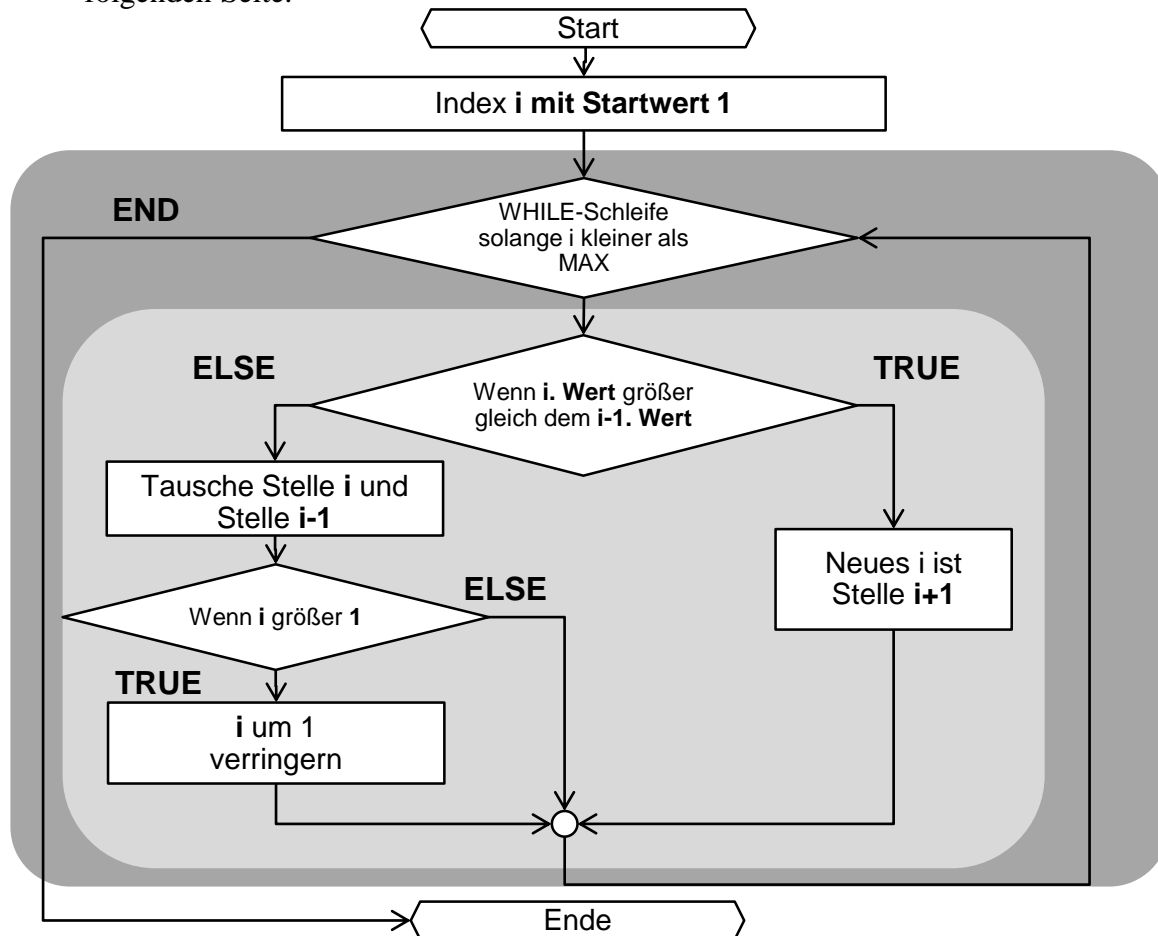


Nachname, Vorname

Matrikelnummer

- e) Abschließend soll eine Funktion implementiert werden, welche ein Array vom Typ Flug mit der Arraygröße „MAX“ sortieren kann. Hierbei sollen die Flüge ihrer geplanten Abflugzeit nach *aufsteigend* (also der nächste Flug zuerst) sortiert werden. Hierfür soll der sog. *Gnome-Sort-Algorithmus* implementiert werden, welcher im Folgenden ausführlich in einem Flussdiagramm beschrieben wird.

Hinweis: Der zu vervollständigende Lückentext für den Code befindet sich auf der folgenden Seite.



Zum weiteren Verständnis der Wirkungsweise des Gnome-Sort-Algorithmus ist im Folgenden ein Beispiel für die Zahlenreihe 1-4-3-2 (MAX ist 4) durchgeführt. Die Plätze im Kästchen werden im aktuellen Schritt verglichen:

1 4 3 2 (i=1), 4 größer 1 ? → Ja, also i=i+1
 1 4 3 2 (i=2), 3 größer 4 ? → Nein, also tauschen! → 1 3 4 2 → i=i-1
 1 3 4 2 (i=1), 3 größer 1 ? → Ja, also i=i+1
 1 3 4 2 (i=2), 4 größer 3 ? → Ja, also i=i+1
 1 3 4 2 (i=3), 2 größer 4 ? → Nein, also tauschen! → 1 3 2 4 → i=i-1
 1 3 2 4 (i=2), 2 größer 3 ? → Nein, also tauschen! → Ergebnis: 1 2 3 4 → i=i-1 → (i=1) ...

Da das Array nun sortiert ist, wird im ersten inneren Vergleich immer TRUE durchlaufen und i läuft bis zu i=4, wodurch die Abbruchbedingung der WHILE-Schleife erfüllt ist.

_____
Nachname, Vorname_____
Matrikelnummer

Vervollständigen Sie nun unten die Funktion „gnomeSort“ entsprechend der zuvor im Flussdiagramm beschriebenen Funktionsweise. Die Funktion wird mittels *call-by-reference* aufgerufen und bekommt einen Zeiger auf das Array übergeben. Sie sortiert also direkt im Array vom Typ *Flug* und gibt keinen Rückgabewert zurück.

Die verwendete Schleife benötigt die Größe des Arrays, welche durch den Präprozessor-Bezeichner „*MAX*“ bereits definiert wurde.

Die Arrayelemente sollen nach der Struct-Variable für die geplante Abflugzeit *aufsteigend* sortiert werden (z.B. *iPlan*).

Es sei des Weiteren bereits eine Funktion „*tausche*“ implementiert, die nicht dargestellt ist. Die Funktion kann zwei Arrayelemente vom Typ *Flug* vertauschen. Verwenden Sie diese 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.

```
_____ gnomeSort(_____)
{
    _____ // Variablen initialisieren

    _____ // Schleife
    {
        // Vergleich:

        _____

        _____
    else
    {
        // Tauschen und ggf. i erniedrigen:

        _____

        _____

        _____
    }
}
```



 Nachname, Vorname

 Matrikelnummer

5. Zeiger

- a) Gegeben sei ein kurzes C-Programm, welches ein Array mit drei Elementen und die Zeigervariable *piZeiger* initialisiert. Vervollständigen Sie in der dritten Zeile von unten den angegebenen printf-Befehl zur Bildschirm-Ausgabe des Elementes, welches sich eins rechts vom Zeiger befindet.
- b) 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.

Hinweis: Aufgabe a) und b) können unabhängig voneinander gelöst werden.

```
#include <stdio.h>
int main ()
{
    int *piZeiger=NULL;
    int iAFeld[3]={5,2,9};

    piZeiger=&iAFeld[1];
    *(piZeiger++)=3;
    *(piZeiger--)+=*(piZeiger-1)*2;
    * (--piZeiger)=iAFeld[0]|7;

    printf("\nWert rechts des Zeigers: _____")

    return 0;
}
```

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