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

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Sommersemester 2015

28.08.2015

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	

Aufgabe G: Grundlagen

Aufgabe G:
48 Punkte

1. Umrechnung zwischen Zahlensystemen

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

1 (222)$_5$ = (.....)$_{10}$ = (.....)$_{16}$

2 (100111,1010)$_2$ = (.....)$_{10}$

2. IEEE 754 Gleitkommazahlen

Rechnen Sie die gegebene Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) in eine Dezimalzahl um.

1	1	0	1	0	1	1	0	0	1
V	e (4 Bit)				M (5 Bit)				

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

Vorzeichen

V=

Bias und biased Exponent

b= **e =**

Exponent

E =

Mantisse (Dualzahl und Denormalisiert)

M_2=

Vollständige Dezimalzahl Z

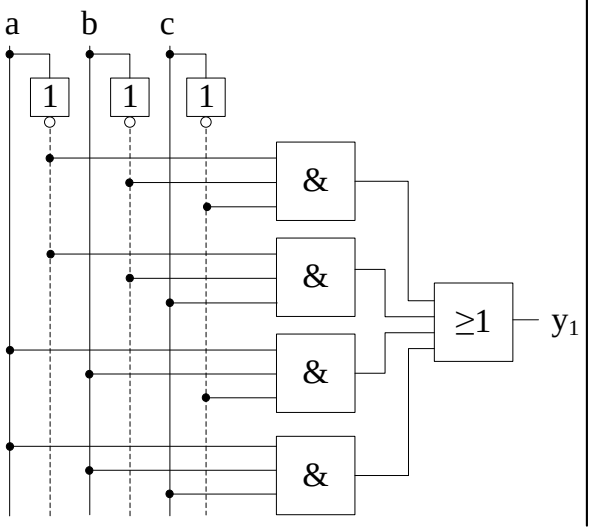
Z=

3. Logische Schaltungen und Schaltbilder

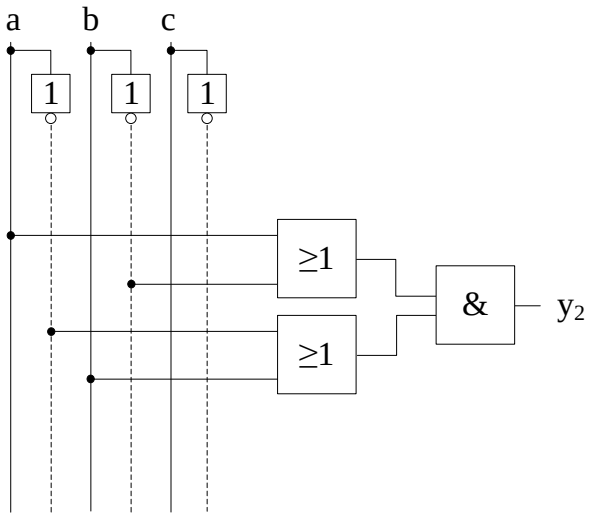
Sie sind zuständig für die Überprüfung der Korrektheit von Schaltungen für die sicherheitstechnische Auslegung einer Anlage. Ein Mitarbeiter legt Ihnen Schaltung 1 und eine minimierte Schaltung 2 vor (siehe unten). Sie müssen überprüfen ob beide Schaltungen das selbe Schaltungsverhalten haben.

Prüfen Sie das Schaltungsverhalten mit Hilfe der unten angegebenen Wahrheitstabelle! Schreiben Sie auf, ob die Schaltungsverhalten identisch sind oder nicht.

Schaltung 1



Schaltung 2



a	**b**	**c**	**y_1**	**y_2**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

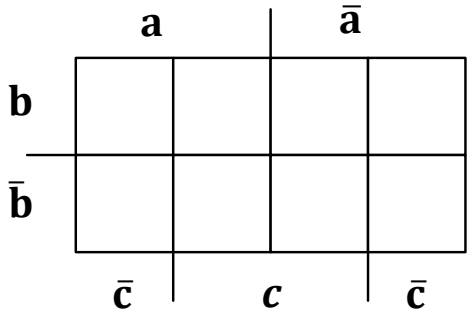
4. Normalformen und Minimierung

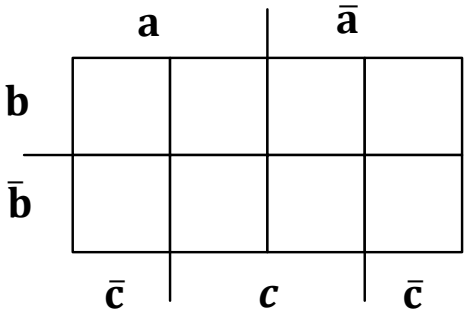
Gegeben ist folgende Tabelle:

a	b	c	y
0	0	0	X
0	0	1	X
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Tragen Sie die Ausgangsvariable y aus der Wahrheitstabelle in das KV-Diagramm und erstellen Sie die minimierte Form in KNF *(Konjunktive Normalform)* Schreibweise. Schreiben Sie ebenfalls die minimierte Funktion in boolescher Algebra auf. Die Ausgänge mit y=„X“ sind don‘t care bits.

Hinweis: Das zweite abgebildete KV-Diagramm ist lediglich redundant, falls Sie sich verzeichnen. Kennzeichnen Sie eindeutig, welches KV-Diagramm bewertet werden soll.

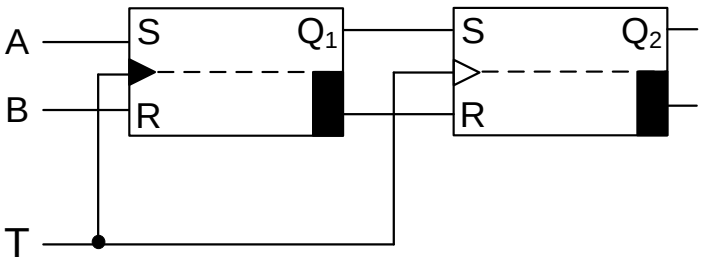




Überprüfen Sie Ihre Lösung, in dem Sie die DNF *(Disjunktive Normalform)* lediglich durch boolesche Algebra minimieren; ebenfalls unter Verwendung der don‘t care bits.

5. Flip-Flops

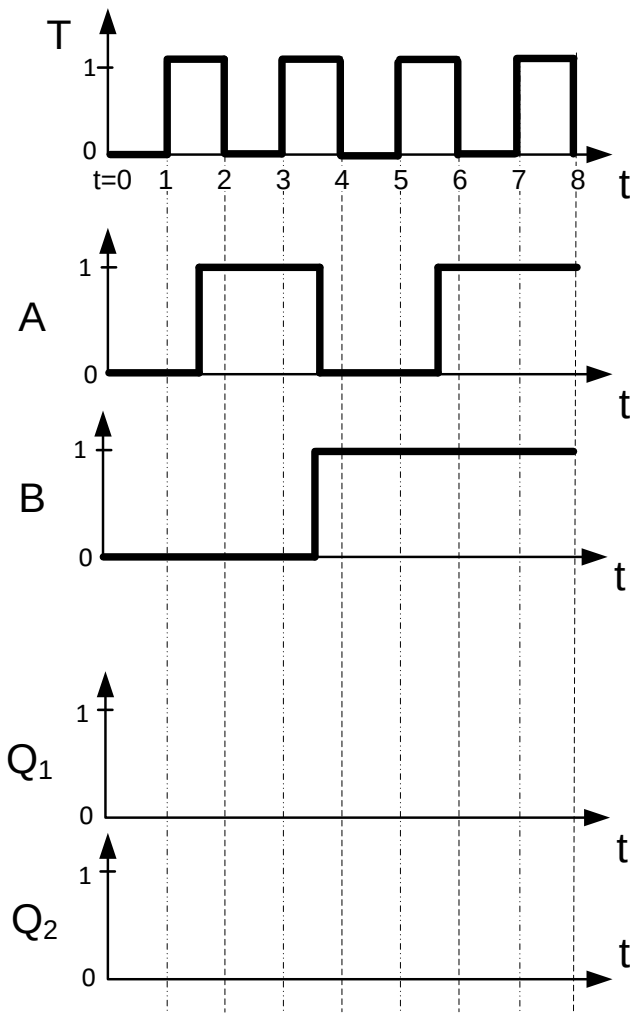
Gegeben ist die folgende Master-Slave Flip-Flop Schaltung (MS-FF)



Bei t = 0 sind die Flip-Flops in folgendem Zustand: $Q_1 = Q_2 = 0$.

Analysieren Sie die Schaltung, indem Sie für die Eingangssignale A, B und T die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme eintragen.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.



6. Befehlsabarbeitung mit MMIX-Rechnerarchitektur

Zum Startzeitpunkt besitzen der Register- und der Datenspeicher eines MMIX-Rechners die in Tabelle GL-5.3 bzw. GL-5.4 (siehe folgende Seite) gegebenen Werte. Es sollen nacheinander drei Befehle ausgeführt und ein Befehl erstellt werden (Tabelle GL-5.5).

Ergänzen Sie zunächst die Befehle der Tabelle GL-5.5, indem Sie die gegebenen Maschinen- oder Assemblerbefehle bzw. Befehlsbeschreibung in die jeweils fehlenden Formen umwandeln (ein Beispiel finden Sie in Tabelle GL-5.2). Führen Sie dann diese Befehle mit den Werten von Register- und Datenspeicher durch (Tabelle GL-5.3 bzw. GL-5.4, „Wert vor Befehlsausführung“) und füllen Sie den Register- und den Datenspeicher für den Zustand nach der Befehlsausführung vollständig aus (Tabelle GL-5.3 bzw. GL-5.4, „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	…	STW	STW I	…
	STT	STT I		STO	STO I	
…	…	…	…	…	…	…
0xE_	SETH	SETMH	…	INCH	INCMH	…
	ORH	ORMH		ANDNH	ANDNMH	
0xF_	JMP	JMP B	…	GETA	GETA B	…
	POP	RESUME		SYNC	SWYM	

Tabelle GL-5.1: MMIX-Code-Tabelle

Maschinensprache	Assemblersprache	Befehlsbeschreibung
z. B. 0x8D 9B A1 24	*LDO $0x9B, $0xA1, $0x24*	*M[$0xA1 + $0x24] = $0x9B oder: „Lädt den Wert der Datenspeicherzelle an der Adresse A1 plus Offset 24 als Octa in die Registerzelle 9B.“*

Tabelle GL-5.2: Lösungsbeispiel

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Wert nach Befehlsausführung
…	…	…
$0x9E	0x00 00 00 00 00 00 AF F8	0x
$0x9F	0xFC B0 00 00 AA AA AA 50	0x
$0xA0	0x01 23 45 67 89 AB CD EF	0x
$0xA1	0x00 00 00 00 00 00 00 0A	0x
…	…	…

Tabelle GL-5.3: Registerspeicher

Datenspeicher																		
Adresse	…	**0x0..0 B0 00**	**0x0..0 B0 01**	**0x0..0 B0 02**	**0x0..0 B0 03**	**0x0..0 B0 04**	**0x0..0 B0 05**	**0x0..0 B0 06**	**0x0..0 B0 07**	**0x0..0 B0 08**	**0x0..0 B0 09**	**0x0..0 B0 0A**	**0x0..0 B0 0B**	**0x0..0 B0 0C**	**0x0..0 B0 0D**	**0x0..0 B0 0E**	**0x0..0 B0 0F**	…
Wert vor Befehls-ausführ.	…	**0x05**	**0x31**	**0x80**	**0x08**	**0x23**	**0x45**	**0x67**	**0x89**	**0xAB**	**0xCD**	**0xAA**	**0x13**	**0x37**	**0x13**	**0x37**	**0x13**	…
Wert nach Befehls-ausführ.	…																	…

Tabelle GL-5.4: Datenspeicher

Maschinensprache	Assemblersprache	Befehlsbeschreibung
0x88 A0 9E A1		
	ADDI $0x9E, $0x9E, 0x12	
		M_2[$0x9E+0x01] = $0x9F
0xE0 9F 18 60		

Tabelle GL-5.5: Maschinensprache – Assemblersprache – Befehlsbeschreibung

Aufgabe BS: Betriebssysteme

Aufgabe BS:
48 Punkte

7. Scheduling

Fünf Prozesse (P1 bis P5) 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 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-1.2 die entsprechende Prioritätenverteilung (Prioritäten 1, 2 und 3) gegeben.

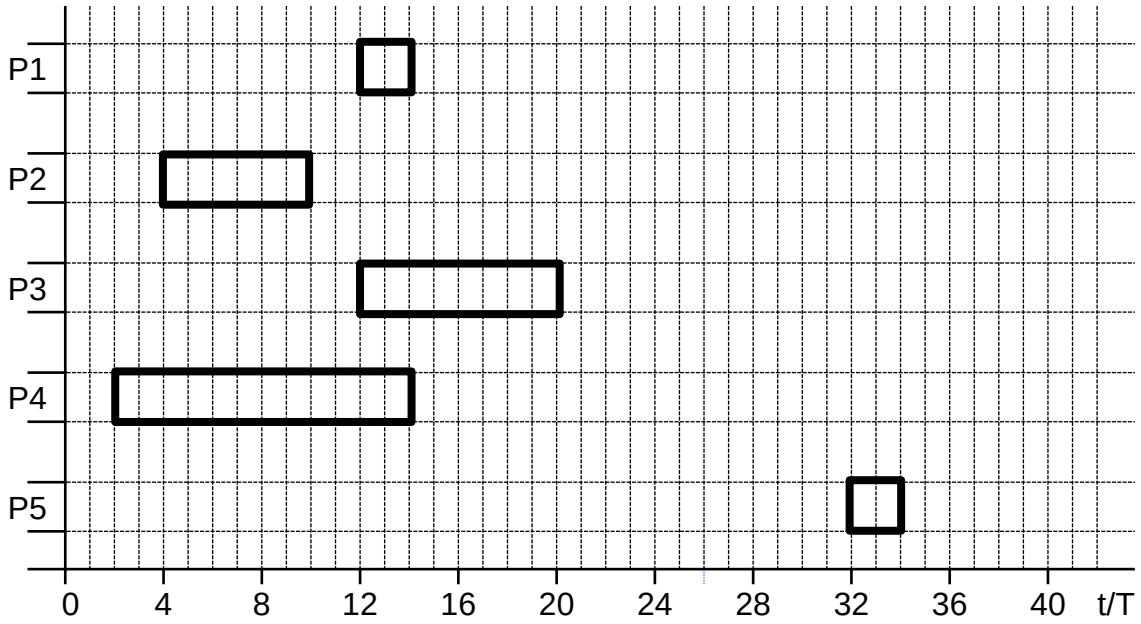


Diagramm BS-1.1: Sollzeitverlauf der Prozesse P1 bis P5

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

Tabelle BS-1.2: Prioritätenverteilung

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

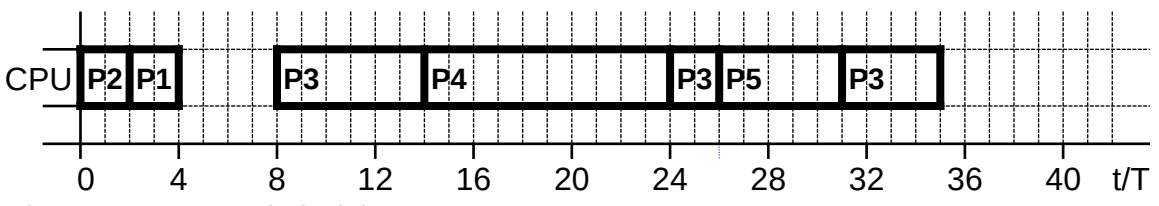


Diagramm: Lösungsbeispiel

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.

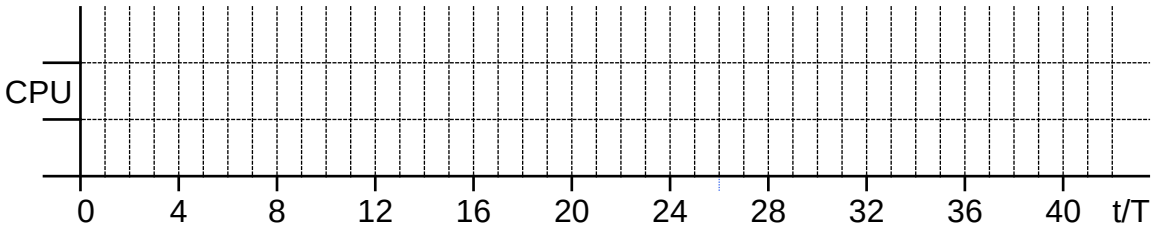


Diagramm: Prioritäten und First In First Out (FIFO)

b) In der zweiten Teilaufgabe soll die Einplanung der Prozesse mit dem Round-Robin-Verfahren erfolgen. Neue Prozesse werden nach ihrer Priorität (RR mit Prio) 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. Restzeiten können nicht übersprungen werden.Tragen Sie den Verlauf der Prozessabarbeitung in das Diagramm BS-1.b ein.

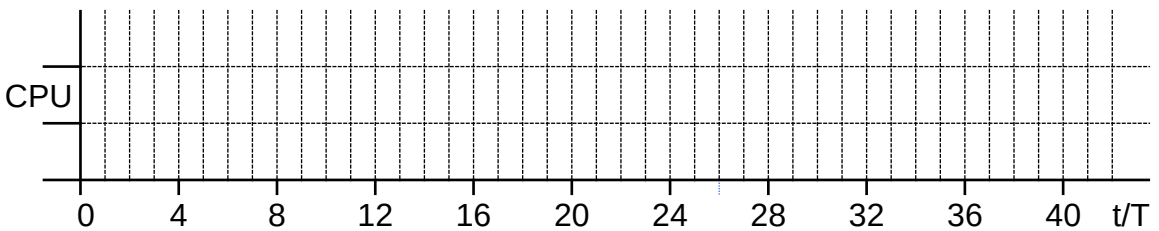
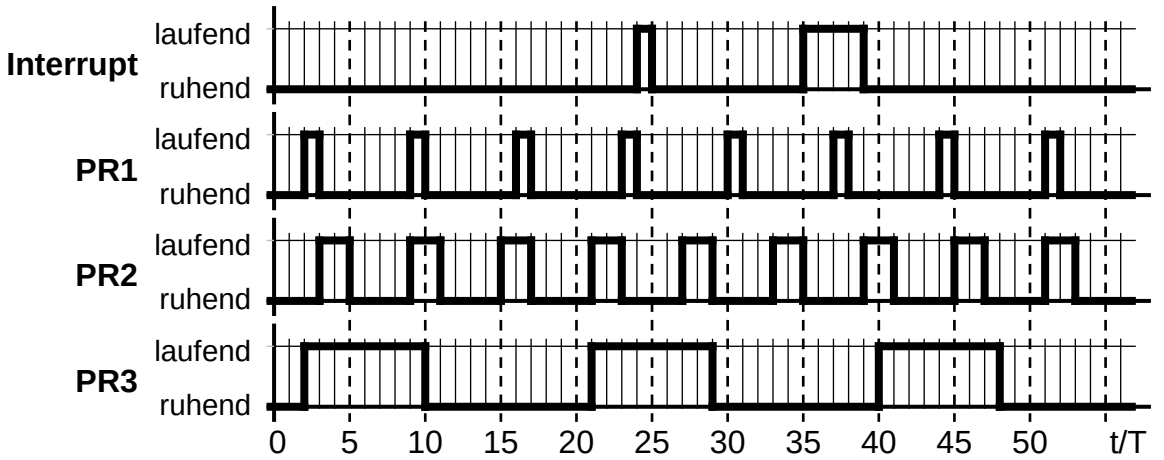


Diagramm: Round Robin (RR) und Prioritäten

8. 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.



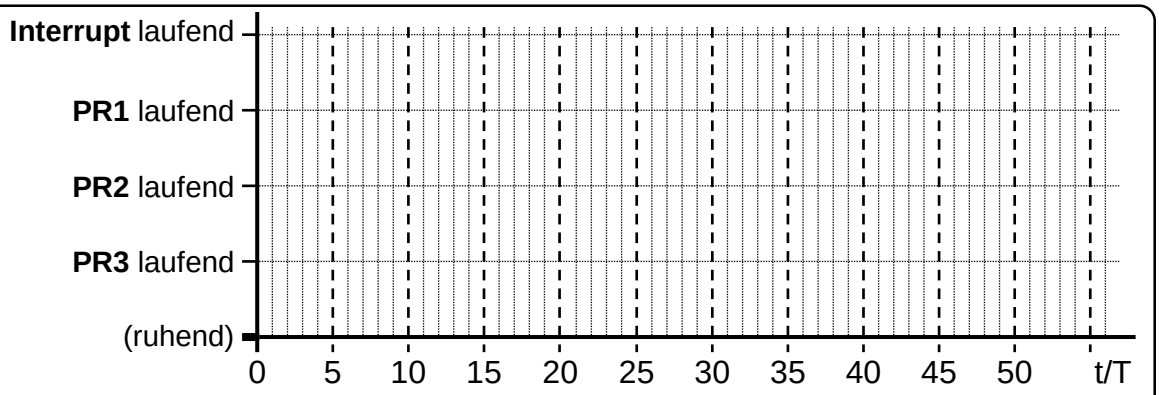


Diagramm: asynchrone Programmierung

☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*

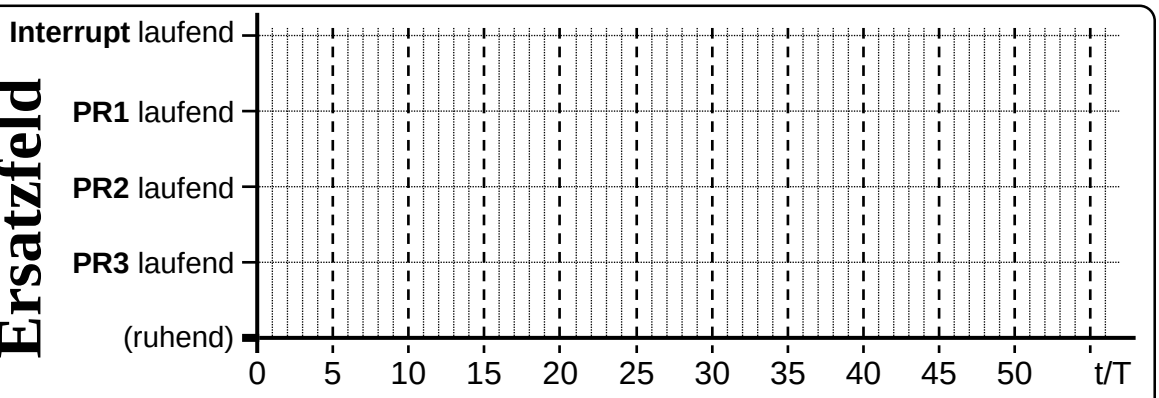
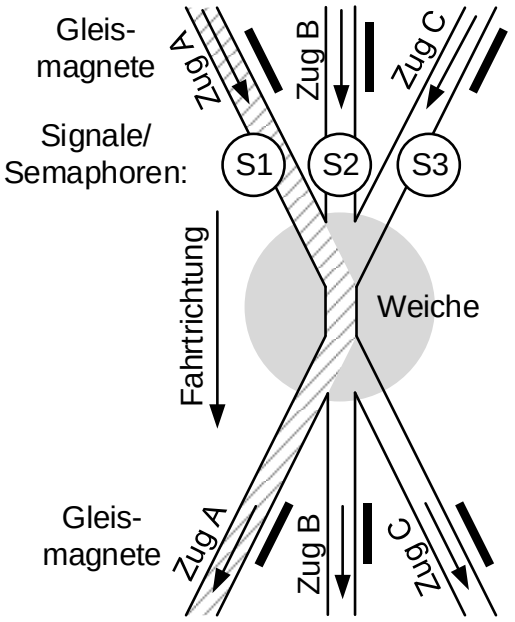


Diagramm: asynchrone Programmierung – **Ersatzfeld, falls oberes Feld falsch**

☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*

9. Semaphoren

Eine Weiche soll die Durchfahrt von drei Zugklassen (Zug A, Zug B, Zug C) durch eine Engstelle regeln. Da die Zugklassen unterschiedlich häufig verkehren, soll die Weiche die Züge in folgender Reihenfolge passieren lassen: $\overline{ABBCB}$.



Entwickeln Sie mit Hilfe der drei Semaphoren S1, S2 und S3 für jeden 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
Semaphoren-operationen ↓	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**
Semaphor-operationen ↓			

zu definierende Folge der Zugklassen: $\overline{ABBCB}$

10. PEARL

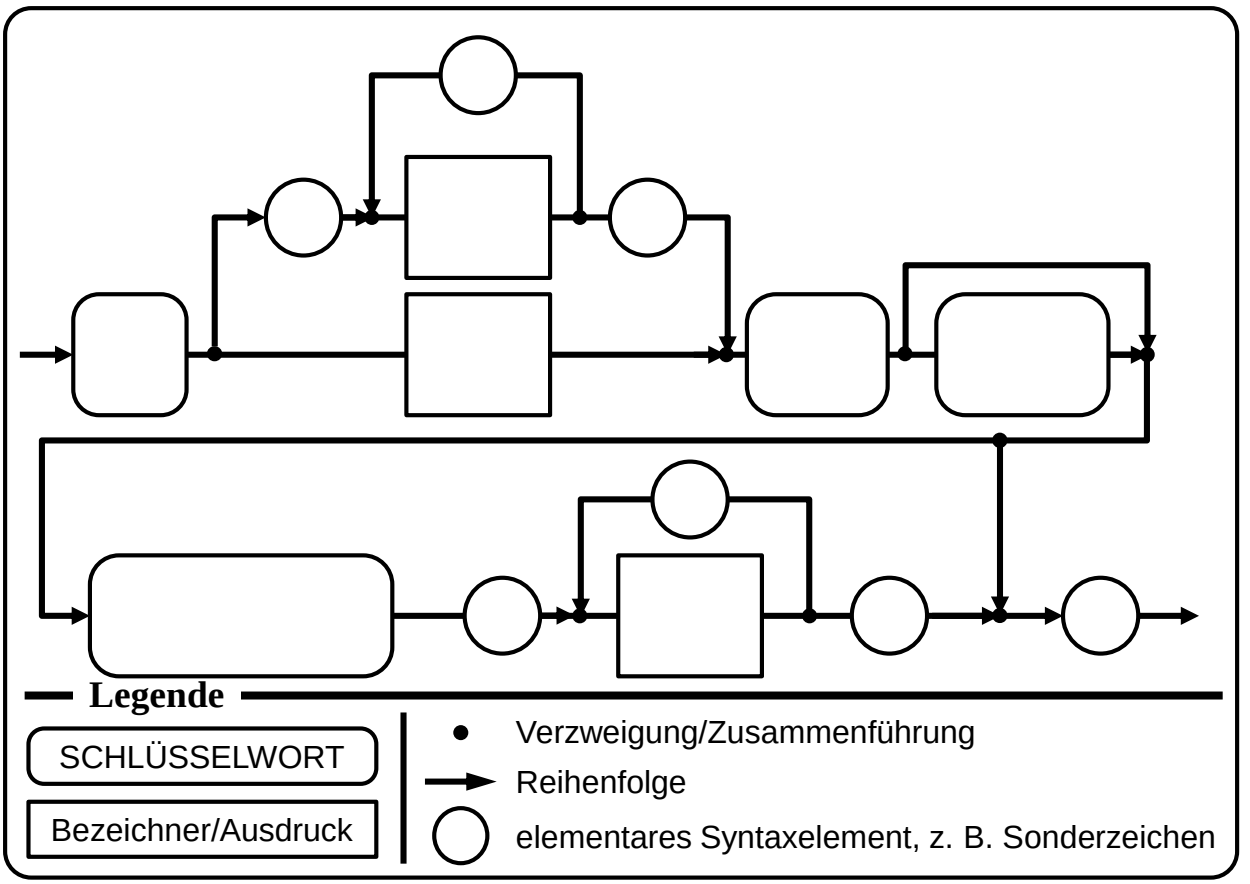
Eine Steuerung deklariert die folgenden PEARL-Semaphoren.

DCL (S1, S2) **SEMA;**

DCL S3 **SEMA GLOBAL PRESET** (5)**;**

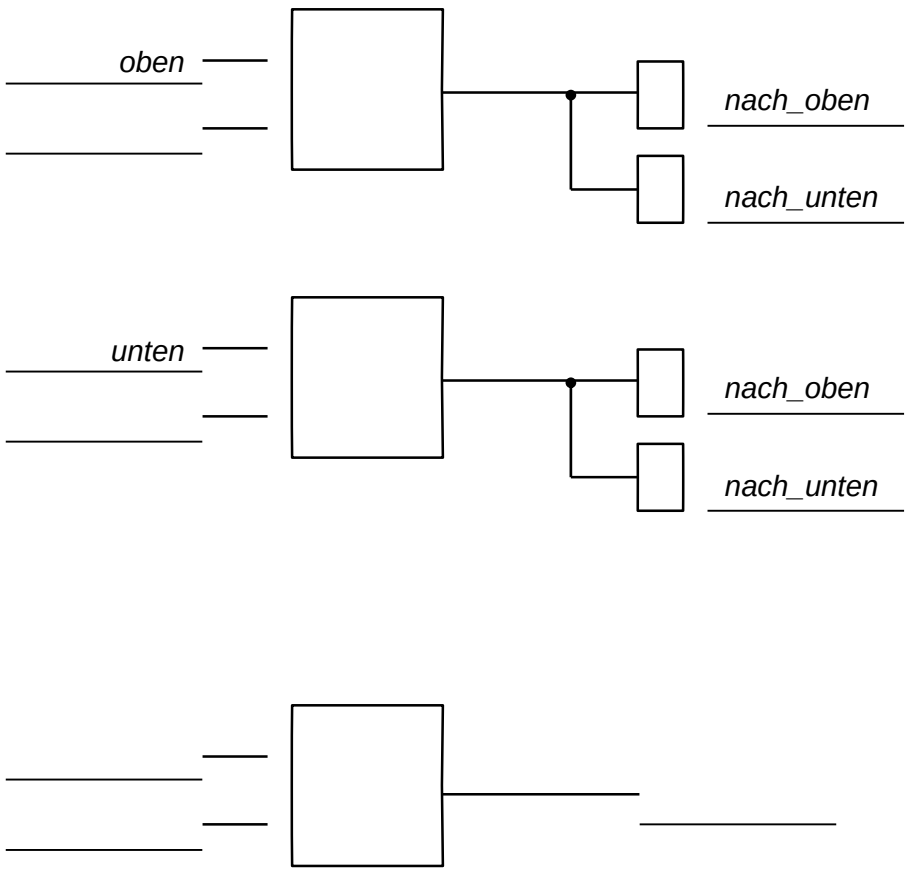
DCL (S4, S5, S6) **SEMA PRESET** (1,0,0)**;**

Leiten Sie aus dem gegebenen Programmausschnitt die Schlüsselwörter, Bezeichner und Syntaxelemente des Syntaxgraphen für die Deklaration von Semaphor-Variablen in der Programmiersprache PEARL ab und tragen Sie diese in die untenstehenden Platzhalter ein.



11. IEC 61131-3: Funktionsbausteinsprache (FBS)

Eine Schrankensteuerung soll mittels Funktionsbausteinsprache (FBS) realisiert werden. Die Schranke besitzt zwei binäre Sensoren *oben* und *unten*, welche angeben, ob sich die Schranke an den Endpunkten befindet. Der Schrankenmotor kann durch die beiden Signale *nach_unten* und *nach_oben* angesteuert werden: Die Schranke soll *nach_unten* fahren, wenn sich die Schranke *oben* befindet und der *Taster* kurzzeitig gedrückt wird. Analog soll sie *nach_oben* fahren, wenn sie sich *unten* befindet und der *Taster* kurzzeitig gedrückt wird. Ist die Schranke in Bewegung, so soll ein *Signal* gegeben werden.



Aufgabe MSE Teil 1: Automaten und Zustandsdiagramm

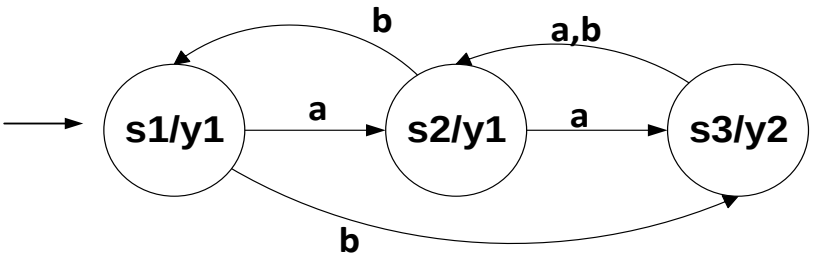
Aufgabe MSE1:
24 Punkte

12. Automaten

a) In welcher Phase des V-Modells der Entwicklung werden Automaten üblicherweise eingesetzt?

Phase: _____

Gegeben sei folgender Automat:



b) Geben Sie an, ob es sich um einen Moore- oder einen Mealy-Automaten handelt. Geben Sie zudem jeweils eine Eingabesequenz von 3 bzw. 4 Eingaben an mit der Sie, ausgehend von Zustand s1, wieder zum Zustand s1 kommen.

Art des Automaten: _____

Eingabesequenz (3 Eingaben): __ __ __

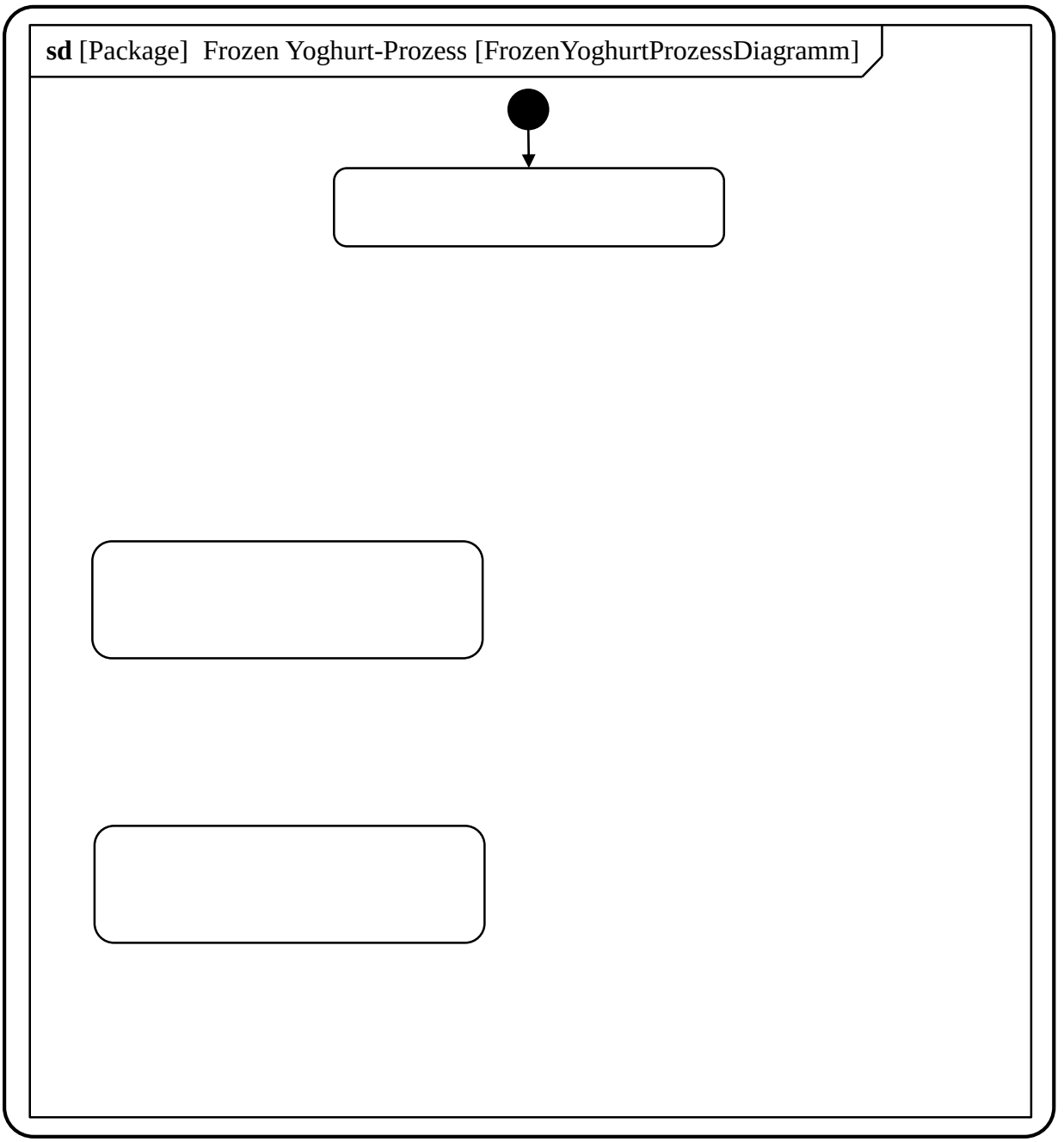
Eingabesequenz (4 Eingaben): __ __ __ __

c) Wandeln Sie den Automaten in die äquivalente andere Form um (Moore → Mealy bzw. Mealy → Moore).

13. Zustandsdiagramm

Der Funktionsablauf einer FrozenYoghurt-Maschine soll als Zustandsdiagramm modelliert werden. Nach dem Start geht die Maschine in den Zustand *Betriebsbereit* über. Beim Trigger *Freeze* (Benutzer möchte FrozenYoghurt herstellen) überprüft die Maschine, ob Joghurt vorhanden ist oder nicht (Variable *Yoghurt* ist *TRUE bzw. FALSE*). Wenn kein Joghurt vorhanden ist, wechselt die Maschine wieder in den Zustand *Betriebsbereit*. Wenn Joghurt vorhanden ist, wird in den Zustand *Gefriervorgang* gewechselt und dort über die gesamte Dauer des Gefrievorgangs die Aktion *gefrieren* durchgeführt. Wenn die Variable *Temperatur* unter 3°C sinkt, wechselt die Maschine in den Zustand *Topping*. In diesem Zustand wird die Aktion *Schokosauce* durchgeführt. Nach 5 Sekunden wird der Zustand verlassen und das Zustandsdiagramm beendet

Vervollständigen Sie das Zustandsdiagramm entsprechend den Angaben.



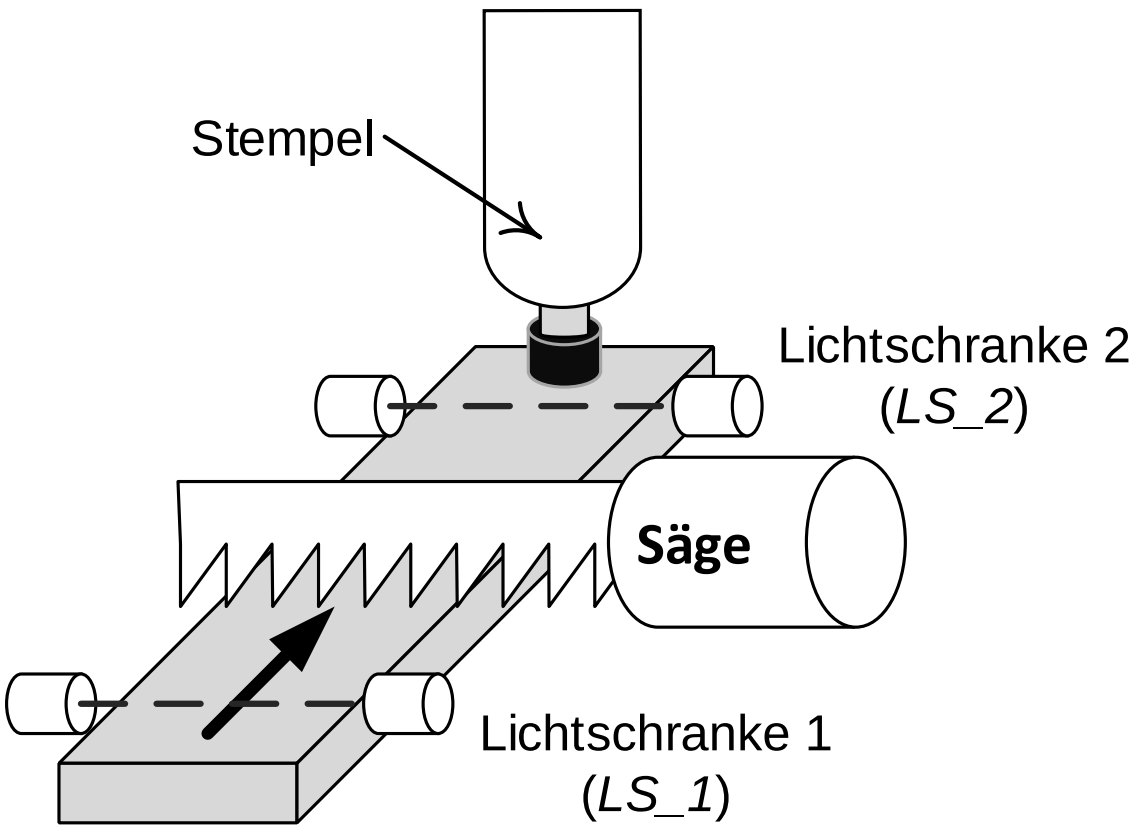
Aufgabe MSE Teil 2: Strukturierte Analyse/ Real-Time (SA/RT)

Aufgabe MSE2:
24 Punkte

14. SA/RT

Gegeben ist folgendes (Teil-)System zur Holzverarbeitung, in welchem Holz zu Brettern gesägt und diese mit dem aktuellen Datum gestempelt werden. Ankommendes Holz wird von einer Lichtschranke (*LS_1*) erkannt. Daraufhin wird es durch die *Säge* abgesägt. Eine zweite Lichtschranke (*LS_2*) erkennt das Brett vor dem *Stempel*. Der *Stempel* versieht das Brett mit dem aktuellen Datum.

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



15. Datenflussdiagramm

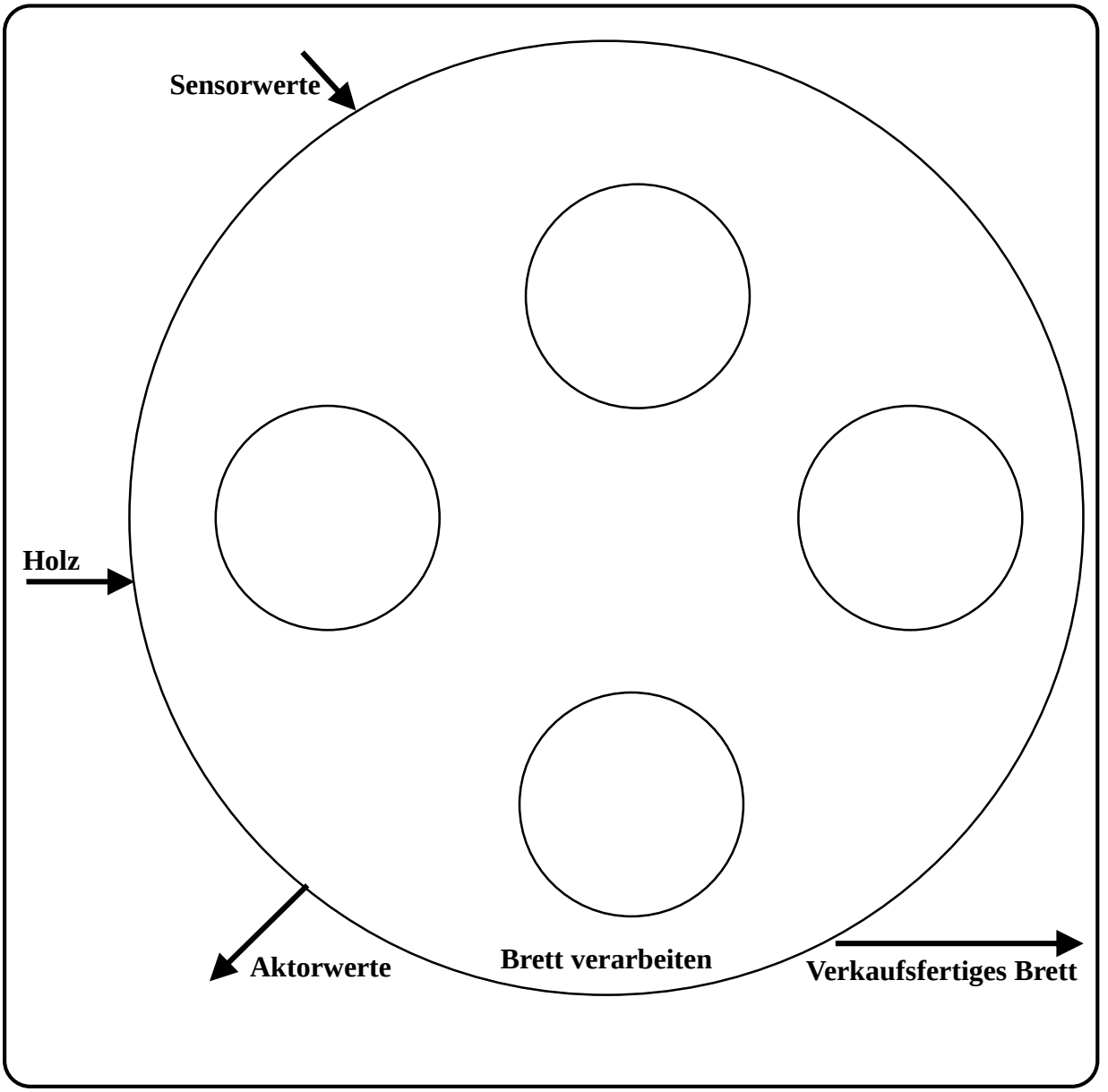
Es soll das Datenflussdiagramm des Prozesses *Brett verarbeiten* modelliert werden. Der Prozess besteht aus den Subprozessen *Holz erkennen*, *Brett sägen*, *Brett erkennen* und *Brett stempeln*. Folgende Flüsse sollen dabei betrachtet werden:

Materialfluss: Am Anfang des Prozesses befinden sich Holz auf dem Band (*Holz auf Band*). Das *erkannte Holz* wird dann gesägt. Das *gesägte Brett* wird daraufhin erkannt und das *erkannte Brett* gestempelt. Das *gestempelte Brett* ist somit verkaufsfertig.

Sensordaten: Von dem übergeordneten Prozess fließen die *Sensordaten der Lichtschranke LS_1* (notwendig um das Holz zu erkennen) und *der Lichtschranke LS_2* (notwendig um das gesägte Brett zu erkennen) zu den entsprechenden Subprozessen.

Aktordaten: Die Aktordaten setzen sich aus den *Aktordaten der Säge* und den *Aktordaten des Stempels* zusammen.

Vervollständigen Sie das untenstehende Datenflussdiagramm entsprechend der Beschreibung.



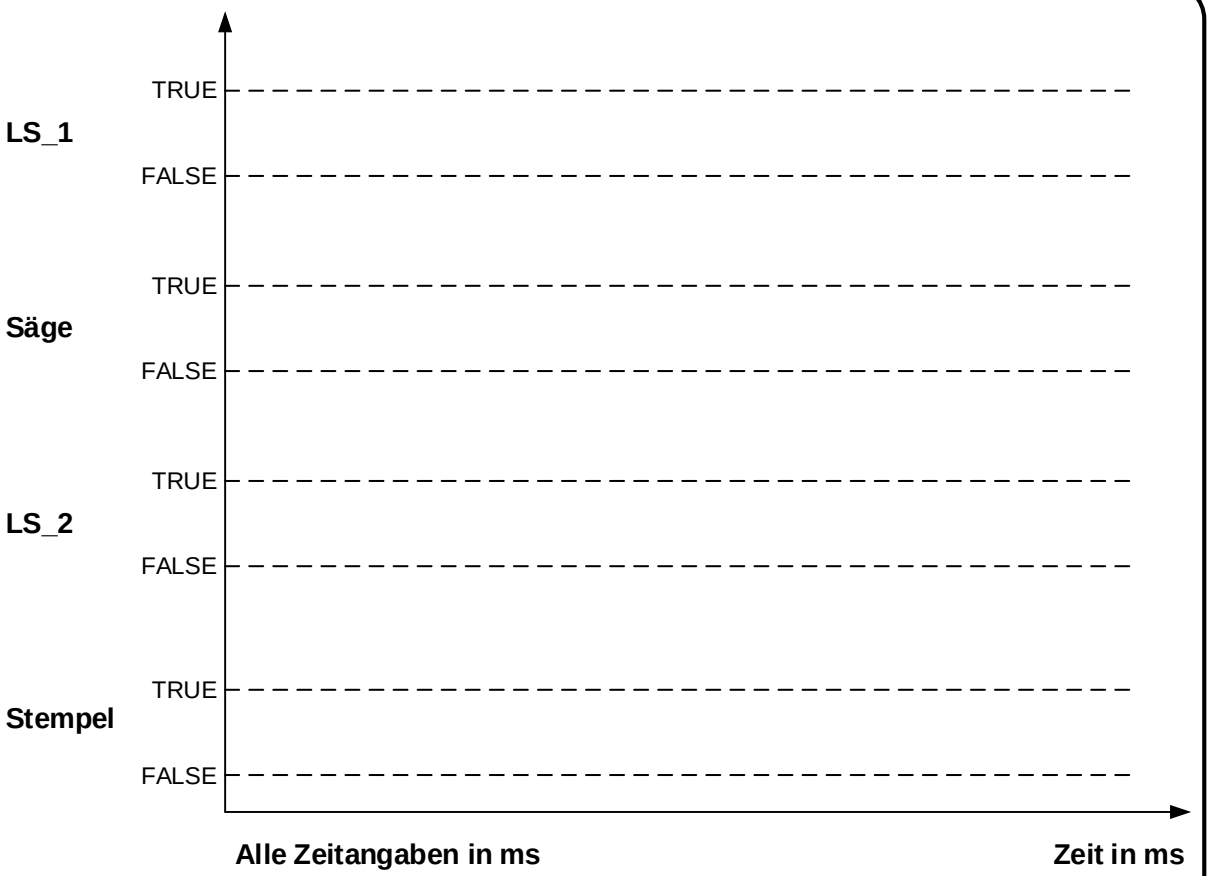
16. Antwortzeitspezifikation: Timing-Diagramm

Zur Verfeinerung des Prozesses *Brett verarbeiten* soll eine Antwortzeitspezifikation in Form eines Timing-Diagramms durchgeführt werden, um sicherzustellen, dass der Prozess korrekt durchgeführt wird.

Die Werte der Sensoren bzw. Aktoren können jeweils *TRUE* (z. B. Brett erkannt) oder *FALSE* (z. B. Brett nicht erkannt) sein.

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

- Sobald Holz auf dem Band erkannt wird, liefert die Lichtschranke *LS_1* für 100 +/− 5 ms den Wert TRUE.
- 200 +/− 10 ms nach Beginn der Holzerkennung durch LS_1 fängt die Säge an, das Holz zu zersägen. Die Säge benötigt dafür 250 +/− 50 ms und schaltet sich danach wieder ab.
- 90 +/− 10 ms nach Beendigung des Sägens soll die Lichtschranke LS_2 das Brett erkennen und wiederum für 100 +/− 5 ms den Wert TRUE liefern.
- 150 +/− 10 ms nach Beginn der Erkennung durch LS_2 wird der Stempel angeschaltet. Wenn kein Brett erkannt wird (z. B. durch einen Fehler der Lichtschranke) bleibt der Stempel ausgeschaltet.



Aufgabe C: Programmieren in C

Aufgabe C:
96 Punkte

17. Datentypen, Ein-/Ausgabe und boolesche Algebra

a) Erstellen Sie die Definition der folgenden Variablen mit Datentypen, um durch das Ablegen **so wenig** Speicher wie möglich zu nutzen. Wählen Sie aussagekräftige Variablenbezeichnungen, um Daten über einen Nutzer speichern zu können.

- Datum als Anzahl der Sekunden seit 1970 (10-stellige positive Zahl, max. jedoch 4.294.967.295)
- Größe in Zentimeter (zwischen 0 und 220)
- Gewicht in Kilogramm auf zwei Nachkommastellen genau

Geben Sie zudem das Gewicht **in Kilogramm** und die Größe **in Meter** mittels formatierter Ausgabe jeweils auf zwei Nachkommastellen aus.

```
_____________________________________________

_____________________________________________

_____________________________________________

// Formatierte Ausgabe des Gewichts

_________(__Gewicht: _____, Größe: ____________________

_____________________)__
```

b) Konstruieren Sie einen booleschen Ausdruck (kein ablauffähiger Code benötigt) welcher die Variable *zahl* darauf prüft, ob diese restlos durch 13 und 17 oder alternativ nicht restlos durch 23 teilbar ist.

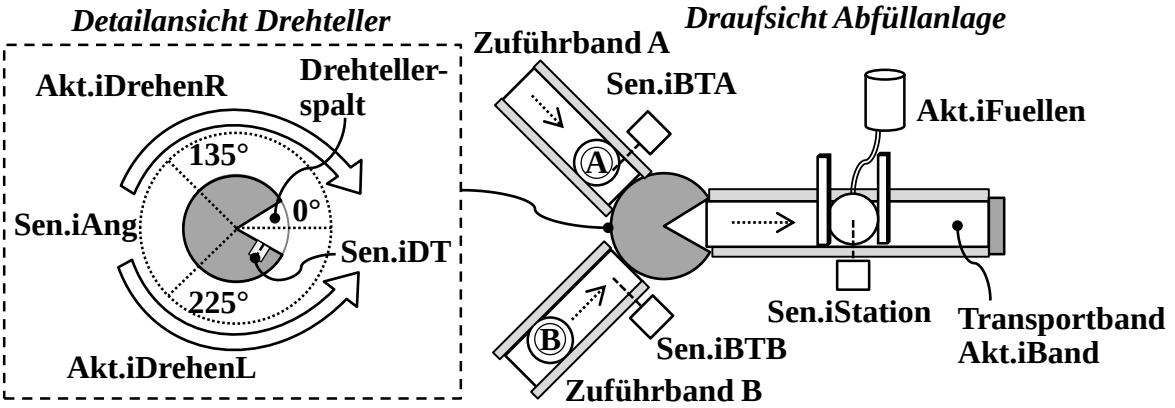
18. Kontrollstrukturen

Ein C-Programm zur Qualitätssicherung soll die Werkstückqualität prüfen, indem mittels einer einfachen Dichteberechnung nach möglichen Materialschwankungen während des Herstellungsprozesses gesucht wird. Dazu wird zu *NUM* Zeitpunkten das Werkstückgewicht (Array *gewicht*) ermittelt. Die Volumen der Werkstücke werden als konstant (Variable *volumen*) angenommen.

Schreiben Sie nun ein C-Programm welches das Durchschnittsgewicht des Werkstücks über alle Messzeitpunkte hinweg ermittelt. Brechen sie mit einer Fehlermeldung (Ausgabe in Konsole) ab, wenn ein Messwert den Toleranzwert „3.1“ überschreitet und geben Sie den Werte „-1“ an die Hauptfunktion zurück. Ignorieren Sie in der Durchschnittsberechnung weiterhin fehlerhafte Messwerte, welche negativ sind. Geben Sie zum Schluss die gemittelte Dichte (Durchschnittsgewicht/Volumen) als Wert zurück.

```
#include <stdio.h>
#define NUM 8


float QualitaetsMessung()
{
    float gewicht[NUM]  = {2.1,2.1,2.2,2.3,2.3,-9,2.3,3.0};
    float volumen = 1.45;
    float sum = 0.0;
    int i=0, j=0;

}
```

19. Zyklische Programmierung einer Abfüllanlage



Die Firma ISA GmbH stellt die Pudding-Abfüllanlage „myPudding“ her, welche zwei Größen von Puddingbecher befüllen kann. Die Anlage erkennt einen neuen Becher auf den beiden durchgehend laufenden Zuführbändern (Größe A und B), vereinzelt und positioniert ihn dann unter einer Abfüllstation. Der Ablauf und die Übergangsbedingungen der zugehörigen Steuerung ist auf der folgenden Seite in einem Zustandsdiagramm beschrieben.

Zustand 0: Der Drehteller befindet sich initial bei Stellung 0°. Falls bereits ein Becher im Drehtellerspalt ist, wird das Transportband gestartet und der Becher abtransportiert (Zustand 4). Andernfalls wird gewartet bis entweder Sen.iBTA oder Sen.iBTB zuerst auslöst, der Typ mittels „iBT“ gespeichert und entsprechend links- oder rechts gedreht (Zustand 1 und 2).

Zustand 1: Der Drehteller wird nach **links** gedreht bis er genau bei 135° steht und ein Becher A aufgenommen wird (Wechsel 1→3). Wurde zuvor ein Becher B aufgenommen, wird bis genau 0° gedreht um diesen auszugeben (Wechsel 1→0).

Zustand 2: Der Drehteller wird nach **rechts** gedreht bis er genau bei 225° steht und ein Becher B aufgenommen wird (Wechsel 2→3). Wurde zuvor ein Becher A aufgenommen, wird wieder bis genau 0° gedreht um diesen auszugeben (Wechsel 2→0).

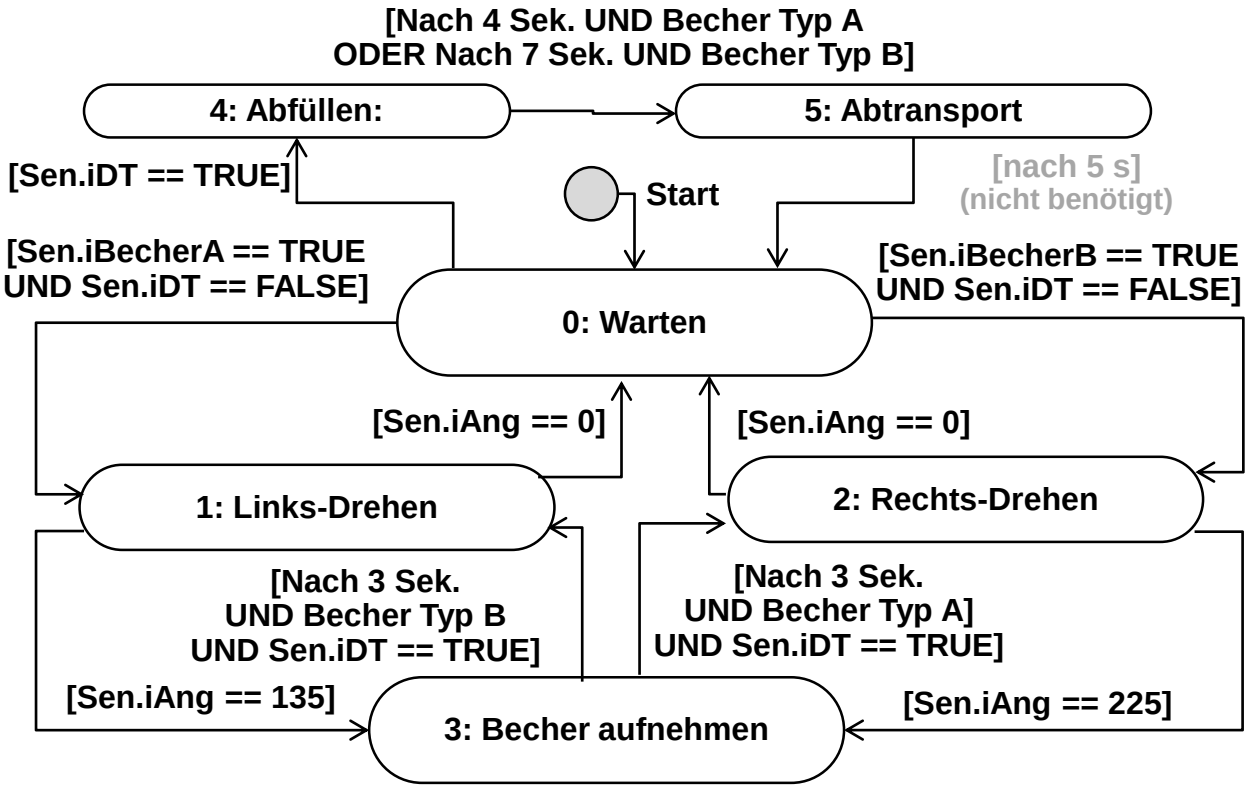
Aktoren:	
Akt.iDrehenL	Dreht den Drehteller links herum (Gegen den Uhrzeigersinn)
Akt.iDrehenR	Dreht den Drehteller rechts herum (Mit dem Uhrzeigersinn)
Akt.iBand	Aktiviert das Transportband zur Abfüllstation
Akt.iFuellen	Aktiviert die Zufuhr von Pudding
Sensoren/Merkvariablen:	
Sen.iAng	Sensorwert der aktuellen Stellung des Drehteller (in Grad)
Sen.iDT	Sensor erkennt Vorhandensein eines Bechers im Drehteller-Spalt
Sen.iBTA / Sen.iBTB	Sensor erkennt Vorhandensein eines Bechers auf Zuführband A bzw. B
Sen.iStation	Sensor erkennt einen Becher an der Abfüllstation
t	Speichert eine Zeit (in ms) für die Verwendung als Timer
iBT	Merkvariable Bechertyp bzw. welche Größe befüllt wird (A=1, B=2)
plcZeit	Systemlaufzeit der Steuerung (in ms)
state	Speichert den aktuellen Zustand

Zustand 3: Um Verklemmen zu vermeiden wird 3 Sekunden gewartet und gleichzeitig geprüft ob ein Becher im Drehtellerspalt ist. Anschließend wird für die Größe A rechts herum gedreht (Zustand 2) und für die Größe B links herum gedreht (Zustand 3).

Zustand 4: Sobald der Becher bei der Abfüllstation angekommen ist wird das Transportband angehalten und Pudding zugeführt. Für die Größe A (iBT=1) wird 4 Sek. lange abgefüllt, für die Größe B (iBT=2) 7 Sek. Danach wird das Band gestartet und zu 5 gewechselt.

Zustand 5: Dieser Zustand regelt den Abtransport, wird jedoch nicht näher betrachtet und nicht implementiert. iBT wird auf 0 zurückgesetzt und schließlich nach 0 gewechselt.

Vorgehen: Übertragen Sie das Zustandsdiagramm und die oben beschriebene Funktionalität in den C-Code auf den folgenden Seiten. Benutzen Sie 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 bereits unten gegeben.



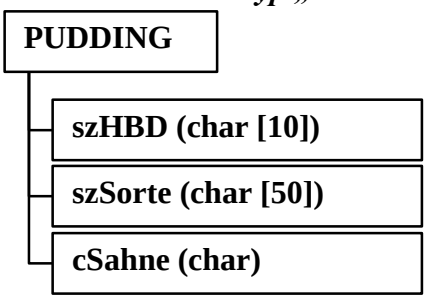

```
#include "eavar_mypudding.h"

struct SData Sen;          // Sensorvariablen
struct AData Akt;          // Aktorvariablen
unsigned int state=0;      // Zustandsvariable (Start in 0)
unsigned int plcZeit=0;    // in ms (1000 entspricht 1 Sek.)

int iBT;   // Merkvariable Bechertyp
int t;     // Speicher für Timer-Messung (in ms)
```

```
switch(state)
{
 ___________ // Zustand 0: Warten

      ____________________________________ // Bed. 0->1

      {_________;_________} //Merken und Statewechsel

  else ____________________________________//Bed. 0->2

      {_________;_________} //Merken und Statewechsel

  else ____________________ // Bed. 0->4

      {_____________;_________}//Band u. Statewechsel

      _____________

  ___________ // Zustand 1: Links drehen

      ___________________________________// Aktor setzen

      ___________________{_____________;________}//1->3

      ___________________{_____________;________}//1->0

      _____________

  ___________ // Zustand 2: Rechts drehen

      ___________________________________// Aktor setzen

      ___________________{_____________;________}//2->3

      ___________________{_____________;________}//2->0

      _____________

// (Fortsetzung naechste Seite)
```

```
__________ // Zustand 3: 3s warten, Becher aufnehmen

      _____________________ // Timer

      ______________________________________
         {
                  _____________________ //3->2

                  _____________________ //3->1

                  t=0;         }
      _________
___________ // Zustand 4: Abfuellen

      _____________________ // Pruefen ob Angekommen

        {
            _____________________

            _____________________  // Abfuellen

            _____________________ // Timer

            //Bedingung 4->5

             ____________________________________

             ____________________________________
              {
                  _____________________

                  _____________________; t=0; // 4->5
              }

            _________________
        }

___________// Zustand 5: Abtransport
        (...) //Implementierung nicht dargestellt
        state=0; // 5->0
        break;
 }
```

20. Speichern von Produktionsdaten in eine Datei

Datenstruktur Datentyp „PUDDING“

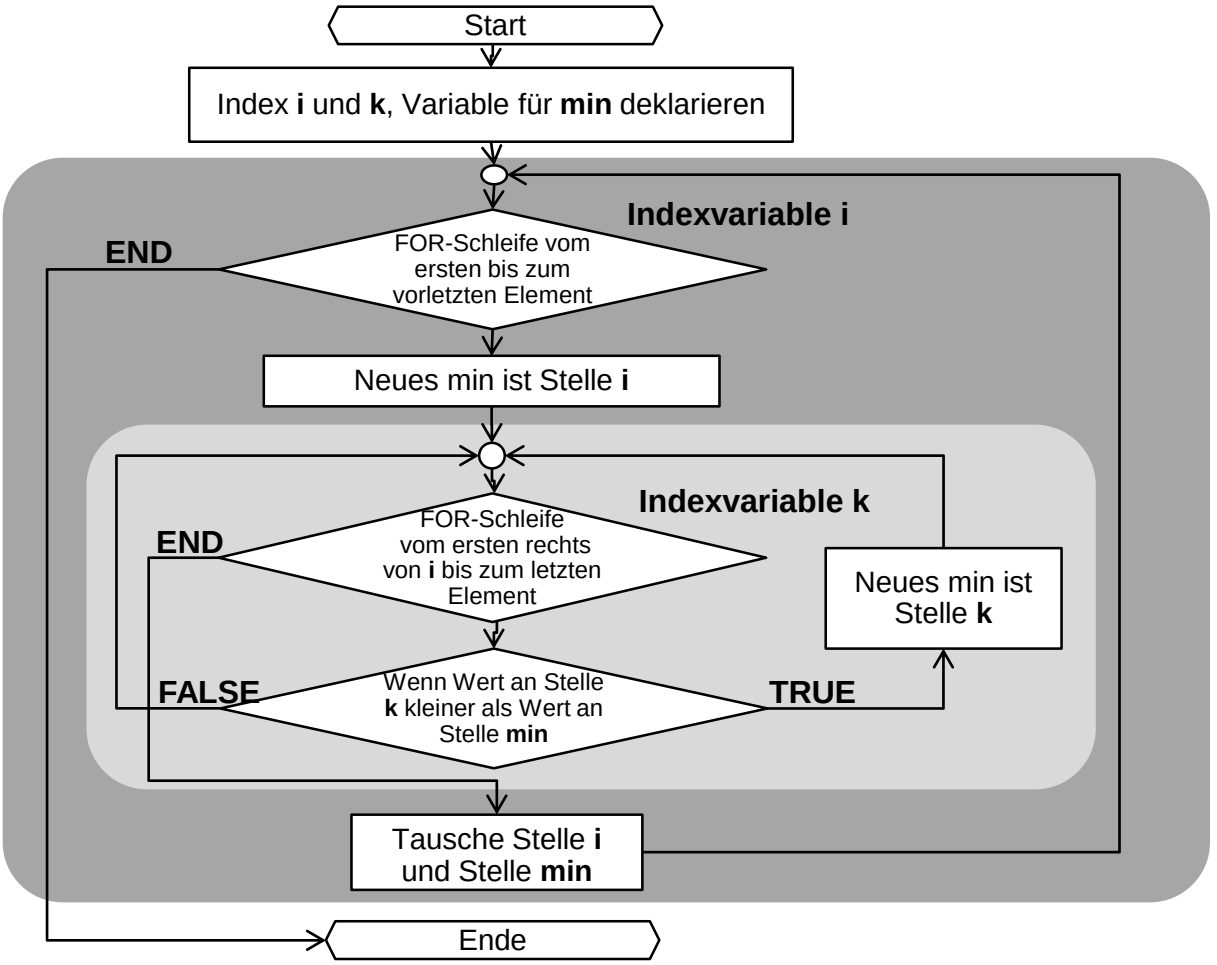


a) Die zuvor mit Hilfe der Abfüllanlage produzierten Pudding-Becher sollen nun an einer weiteren Station geprüft und die Daten in eine Datei zur Qualitätssicherung übernommen werden. Um eine hohe Nachverfolgbarkeit zu gewährleisten, speichert die Steuerung daher alle Daten über den Pudding in die Textdatei "produktion.txt".

Vervollständigen Sie nun ein Programm zum Abspeichern der Daten. Legen Sie dabei ein Array vom Datentyp PUDDING an (oben erläutert), welches 20 Einträge zwischenspeichern kann. Greifen Sie anschließend auf die Datei „produktion.txt“ mittels eines FILE-Handle mit der Bezeichnung „pSpeichern“ so zu, dass die Daten vom Programm **am Ende der Datei** angehängt werden. Um alle Daten abzuspeichern, ist eine Schleife notwendig, die alle Array-Einträge einzeln übergibt. Verwenden Sie in der Schleife einen geeigneten Befehl, um jeweils in einer neuen Zeile den String „szHBD“ (Haltbarkeitsdatum), die Pudding-Sorte „szSorte“ und ob der Pudding eine Sahnetopping hat (0/1) durch Leerzeichen getrennt, abzuspeichern. Schließen Sie schlussendlich den Dateizugriff wieder.

```
int main()
{   int i=0, j=0;
    //Variablendeklaration:

    _____________________________________________
     pDB=readall(); // Einlesen der Daten in die DB
    //Öffnen der Datei produktion.txt:

    ____________________________________________________________
    //Schleife vom Anfang bis Ende des Array:

    ____________________________________________________
    { // Befehl zum Beschreiben

      ____________________________________________________

      ____________________________________________________;
    } //Schließen der Datei:

    ____________________________________
    return 0;  }
```

b) Abschließend soll eine Funktion implementiert werden, welche ein Array vom Typ PUDDING mit der Arraygröße „MAX“ sortieren kann. Hierbei sollen die Pudding-Becher ihrer Sorte nach (erster Buchstabe zählt) *aufsteigend* (also A zuerst) sortiert werden. Hierfür soll der sog. *Selection-Sort-Algorithmus* implementiert werden, welcher im Folgenden ausführlich in einem Flussdiagram beschrieben wird. Die Variable „min“ dient hierbei als Merker für die Stelle des aktuell am niedrigsten bekannten Wertes bzw. Buchstaben.



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 0 (Inhalt: 3) , Vergleich mit allen rechtsseitigen Zahlen
=> min = 4 (Inhalt: 1) => Tausche 3 und 1

Sortiervorgänge:

1 9 2 7 3 (i=1) 1 ist neues min (Inhalt: 9), Vergleich rechtsseitigen Werten => Tausche 9 u. 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**

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 PUDDING und gibt keinen Rückgabewert zurück.

Die verwendeten Schleifen sollen genau für ein Array mit 20 Einträgen ausgelegt sein.

Die Arrayelemente sollen nach dem ersten Buchstaben in der Struct-Variable *szSorte aufsteigend* sortiert werden.

Es sei des Weiteren bereits eine Funktion „*tausche*“ implementiert, die nicht dargestellt ist. Die Funktion kann zwei Arrayelemente vom Typ PUDDING vertauschen. Verwenden Sie diese durch einen korrekten Funktionsaufruf mittels *call-by-reference*, indem Sie ihr die beiden Arrayelemente als Adresse übergeben.

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

21. Verkettete Liste

a) Für das Oktoberfest 2015 soll ein Programm zum Erfassen der Personaldaten der Arbeitskräfte im „Schottenhammel“-Zelt mit Hilfe einer verketteten Liste erstellt werden. Erstellen Sie hierzu zunächst alle für das Problem und die Liste benötigten Datenstrukturen. Der Datentyp für eine Person enthält zusätzlich den Typ der Arbeitsstelle. Legen Sie für diesen zuerst einen ENUM-Datentyp mit den angegebenen vier Unterscheidungen an. Weiterhin wird ein Datentyp für das Listenelement und schließlich den Listenkopf benötigt.

Folgende Personendaten sind zu erfassen:

- *Name* (Nachname) der Arbeitskraft (max. 50 Zeichen)
- *Alter* der Arbeitskraft
- *Stelle* der Arbeitskraft (entweder *Kassenpersonal*, *Schankwirt*, *Koch* oder *Bedienung*)

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


// Typdefinition Stelle



// Typdefinition Personeneintrag PERSON









// Typdefinition Listenelement ELEMENT



// Typdefinition Listenkopf LISTE



(Fortsetzung nächste Seite)
```

b) In der folgenden Aufgabe soll nun die Funktion „insertfront“ zum Einlesen eines neuen Personeneintrags am Anfang der Liste, sowie deren Aufruf in der main-Funktion für das Oktoberfest-Datenerfassungsprogramms vervollständigt werden.

- Vervollständigen Sie dazu zunächst die Deklaration der „insertfront“ Funktion indem Sie dieser die Liste als Adresse übergeben.
- Reservieren Sie weiterhin einen neuen Speicherplatz an der ersten Stelle in der Liste mit dem Typ des Listenelements und hängen Sie das ehemals erste und bereits gesicherte Element mit Hilfe des Zeigers pTemp wieder nach dem neuen ersten Element ein.
- Vervollständigen Sie die scanf-Funktion um vom Benutzer einen String für den Namen des neuen Personeneintrags an der ersten Stelle einzulesen.
- Geben Sie in der „main“-Funktion mit Hilfe des printf-Befehls den Namen der ersten Person in der Liste aus.

```
_________ insertfront(_____________) //Insert-Funktion
{
  ELEMENT *pTemp=liste->pFirst; //Sichern erstes Element
  // Neuen Speicherplatz reservieren
  ____________________________________________________
  // Altes erstes Element anheften
  ____________________________________________________
printf("Geben Sie den Namen der neuen Person ein:");
scanf("%s",_____________________________________);
}
int main()
{
    LISTE liste;
    liste.pFirst=NULL;
    insertfront(&liste);
    printf("Name der ersten Person in der Liste: %s",
          ____________________________________________);
    return 0;
}
```