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

Bitte ausfüllen:	Matrikelnummer:
Vorname:	
Nachname:	
Für die eindeutige Zuordnung Ihrer Prüfung übertragen Sie bitte Ihre Matrikelnummer gewissenhaft in die dafür vorgesehenen Felder.	

Prüfung – Informationstechnik
Wintersemester 2015/2016

04.03.2016

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 roter 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!
Hinweis: Nur Lösungen innerhalb der Lösungskästen werden gewertet! Lassen Sie etwaige leere Stellen frei (siehe Beispiel).

Beispiel: $(123)_{10}$ = (**1** **2** **3** | | |)$_{10}$

a) $(148)_9$ = (| | | | |)$_{10}$ **$(125)_{10}$**

b) $(128)_{10}$ = (| | | | |)$_{16}$ **$(80)_{16}$**

c) $(21{,}0625)_{10}$ = (| | | | | , | | | | |)$_2$

$(10101{,}0001)_2$

2. IEEE 754 Gleitkommazahlen

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

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

Hinweis: Ergebnisse und Nebenrechnungen außerhalb der dafür vorgesehenen Textblöcke werden nicht bewertet! Bitte nutzen Sie die Lösungskästen.

Vorzeichen

V = | **„-“**

Bias

B = | | | | | | | | | **7**

Biased Exponent

e = | | | | | | | | | **13**

Exponent

E = | | | | | | | | | **6**

Mantisse (Dualzahl und Denormalisiert)

M_2 = (| | | | | | | |)$_2$ **$(1100110)_2$**

Vollständige Dezimalzahl

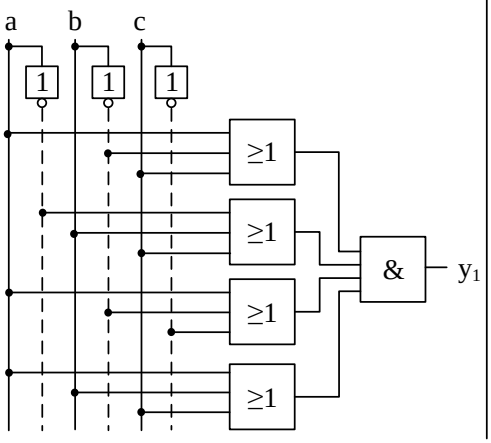
Z = | | | | | | | | | **-102**

3. Logische Schaltungen und Schaltbilder

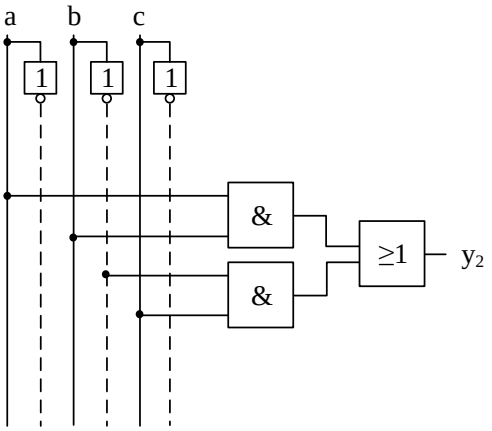
Sie sind zuständig für die Überprüfung der Korrektheit von Schaltungen für die sicherheitstechnische Auslegung einer Anlage. Ein Mitarbeiter legt Ihnen Schaltung 1 und eine minimierte Schaltung 2 vor (siehe unten). Sie müssen überprüfen, ob beide Schaltungen das selbe Schaltungsverhalten haben.

Prüfen Sie das Schaltungsverhalten mit Hilfe der unten angegebenen Wahrheitstabelle! Geben Sie an, ob das Schaltungsverhalten identisch ist oder nicht.

Schaltung 1



Schaltung 2



Wahrheitstabelle

Nr.	a	b	c
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Ergebnis (*Hinweis*: Nur Lösungen innerhalb der Lösungskästen werden gewertet!)

Nr.	y1	y2
1	**0**	**0**
2	**1**	**1**
3	**0**	**0**
4	**0**	**0**
5	**0**	**0**
6	**1**	**1**
7	**1**	**1**
8	**1**	**1**

Ist das Schaltungsverhalten identisch? (*Hinweis*: Bitte nutzen Sie den Lösungskasten.)

Schaltungsverhalten ist identisch!

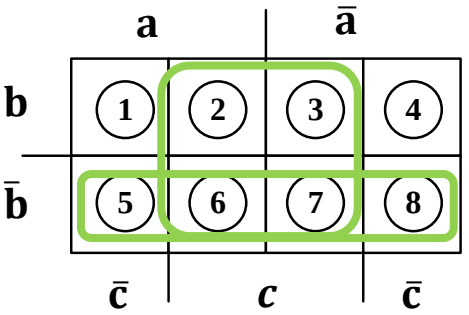
4. Normalformen und Minimierung

Gegeben ist folgende Tabelle sowie das folgende leere KV-Diagramm:

Wahrheitstabelle

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

KV-Diagramm



a) Geben Sie für die Felder 1 bis 8 im KV-Diagramm an, welchen Wert die Ausgangsvariable y aus der Wahrheitstabelle annimmt. Geben Sie die die beiden Terme der DNF *(Disjunktive Normalform)* an. Schreiben Sie ebenfalls die minimierte Funktion in boolescher Algebra auf. Die Ausgänge mit y = „X“ sind don‘t care bits.

Feld	1	2	3	4	5	6	7	8
Wert der Ausgangsvariablen y	0	1	1	0	1	X	X	X

Terme der DNF

$\bar{b}$ c

Minimierte Form in boolescher Algebra:

Formel: $y_{min} = \bar{b} \vee c$

b) Überprüfen Sie Ihre Lösung, in dem Sie die KNF *(Konjunktive Normalform)* lediglich durch boolesche Algebra minimieren; ebenfalls unter Verwendung der don‘t care bits. *Hinweis*: Schreiben Sie alle Zwischenschritte in das Lösungsfeld!

Term 1 **Term 2**

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

Term 1 mit Term 2

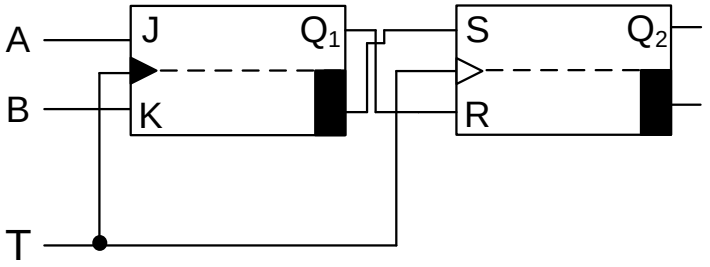
$$y = (\bar{b} \wedge c) \vee (a) \vee (\bar{a})$$

Letzten beiden Terme

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

5. Flip-Flops

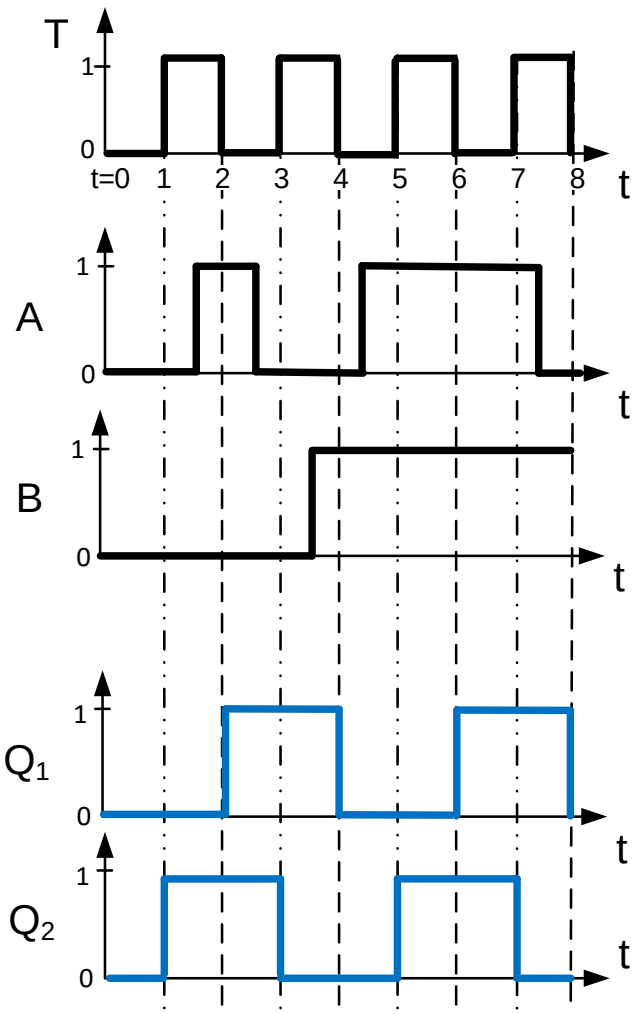
Gegeben ist die folgende Master-Slave Flip-Flop Schaltung (MS-FF):



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, übernächste Seite).

Ergänzen Sie zunächst die Befehle der Tabelle GL-5.5 (übernächste Seite), 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 (unter Tabelle GL-5.3 bzw. GL-5.4, „Wert nach Befehlsausführung“). Beachten Sie, dass Sie in der letzten Zeile der Tabelle GL-5.5 den richtigen Befehl aus dem gegebenen Wert der Registerzelle $0xA1 nach Befehlsausführung ableiten können.

	0x_0	0x_1	0x_2	0x_3	0x_4	0x_5	…
	0x_8	0x_9	0x_A	0x_B	0x_C	0x_D	
0x0_	TRAP	FCMP	FUN	FEQL	FADD	FIX	…
	FLOT	FLOT I	FLOTU	FLOTU I	SFLOT	SFLOT I	
0x1_	FMUL	FCMPE	FUNE	FEQLE	FDIV	FSQRT	…
	MUL	MUL I	MULU	MULU I	DIV	DIV I	
0x2_	ADD	ADD I	ADDU	ADDU I	SUB	SUB I	…
	2ADDU	2ADDU I	4ADDU	4ADDU I	8ADDU	8ADDU I	
…	…	…	…	…	…	…	…
0x8_	LDB	LDB I	LDBU	LDBU I	LDW	LDW I	…
	LDT	LDT I	LDTU	LDTU I	LDO	LDO I	
0x9_	LDSF	LDSF I	LDHT	LDHT I	CSWAP	CSWAP I	…
	LDVTS	LDVTS I	PRELD	PRELD I	PREGO	PREGO I	
0xA_	STB	STB I	STBU	STBU I	STW	STW I	…
	STT	STT I	STTU	STTU I	STO	STO I	
…	…	…	…	…	…	…	…
0xE_	SETH	SETMH	SETML	SETL	INCH	INCMH	…
	ORH	ORMH	ORML	ORL	ANDNH	ANDNMH	
0xF_	JMP	JMP B	PUSHJ	PUSHJ B	GETA	GETA B	…
	POP	RESUME	SAVE	UNSAVE	SYNC	SWYM	

Tabelle GL-5.1: MMIX-Code-Tabelle

Maschinensprache	Assemblersprache	Befehlsbeschreibung
z. B. 0x1C 9B A1 24	*DIV $0x9B, $0xA1, $0x24*	*$0x9B = $0xA1 / $0x24 oder: „Teile den Werte der Zelle mit der Adresse 0xA1 durch den der Zelle 0x24 und speichert das Ergebnis in der Zelle 0x9B.“*

Tabelle GL-5.2: Lösungsbeispiel

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Wert nach Befehlsausführung
…	…	…
$0x9E	0x00 00 00 00 00 00 65 7F	(a)
$0x9F	0x00 00 00 00 99 C0 FF EE	(b)
$0xA0	0x00 00 00 00 00 00 AF F6	(c)
$0xA1	0x00 00 00 00 00 00 0B AD	(d)
…	…	…

Tabelle GL-5.3: Registerspeicher

Füllen Sie die Werte im Registerspeicher nach Befehlsausführung vollständig aus.
Hinweis: Nur Lösungen innerhalb der Lösungskästen werden gewertet!

(a) 0x	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**C**	**A**	**F**	**E**
(b) 0x	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**9**	**9**	**C**	**0**	**F**	**F**	**E**	**E**
(c) 0x	**0**	**0**	**0**	**0**	**B**	**E**	**E**	**F**	**0**	**0**	**0**	**0**	**A**	**F**	**F**	**6**
(d) 0x	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**0**	**0**	**D**

Datenspeicher																			
Adresse	…	0x0..0 AF F0	0x0..0 AF F1	0x0..0 AF F2	0x0..0 AF F3	0x0..0 AF F4	0x0..0 AF F5	0x0..0 AF F6	0x0..0 AF F7	0x0..0 AF F8	0x0..0 AF F9	0x0..0 AF FA	0x0..0 AF FB	0x0..0 AF FC	0x0..0 AF FD	0x0..0 AF FE	0x0..0 AF FF	…
Wert vor Befehls-ausführ.	…	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	…
Wert nach Befehls-ausführ.	…	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	…

Tabelle GL-5.4: Datenspeicher

Füllen Sie die Werte im Datenspeicher nach Befehlsausführung vollständig aus.
Hinweis: Nur Lösungen innerhalb der Lösungskästen werden gewertet!

(1) 0x	**0**	**0**	(5) 0x	**9**	**9**	(9) 0x	**0**	**0**	(13) 0x	**0**	**0**
(2) 0x	**0**	**0**	(6) 0x	**C**	**0**	(10) 0x	**0**	**0**	(14) 0x	**0**	**0**
(3) 0x	**0**	**0**	(7) 0x	**F**	**F**	(11) 0x	**0**	**0**	(15) 0x	**0**	**0**
(4) 0x	**0**	**0**	(8) 0x	**E**	**E**	(12) 0x	**0**	**0**	(16) 0x	**0**	**0**

Maschinensprache	Assemblersprache	Befehlsbeschreibung
0x19 9E 9E 02	**MULI $0x9E, $0x9E, 0x02**	**$0x9E = $0x9E * 0x02**
0xA9 9F A0 01	**STTI $0x9F, $0xA0, 0x01**	**M_4[$0xA0+0x01] = $0x9F**
0xE5 A0 BE EF	**INCMH $0xA0, 0xBE, 0xEF**	**Addiere 0xBEEF zu den Bits 32..47 des Registerinhalts von 0xA0.**
Füllen Sie die folgende Tabellenzeile so, dass nach dieser Befehlsausführung der oben angegebene Inhalt der Registerspeicherzelle $0xA1 erreicht wird!		
0x E3 A1 60 0D	**SETL $0xA1, 0x60, 0x0D**	**$0xA1 = 0x 0..00 60 0D**

Tabelle GL-5.5: Maschinensprache – Assemblersprache – Befehlsbeschreibung

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. Die Prozesse sollen zur Laufzeit mit unterschiedlichen Schedulingverfahren eingeplant werden. Alle Schedulingverfahren beginnen zum Zeitpunkt t = 0T. Für die Schedulingverfahren, bei denen Prioritäten berücksichtigt werden müssen, ist in der Tabelle BS-1.2 die entsprechende Prioritätenverteilung (Prioritäten 1 bis 3) gegeben.

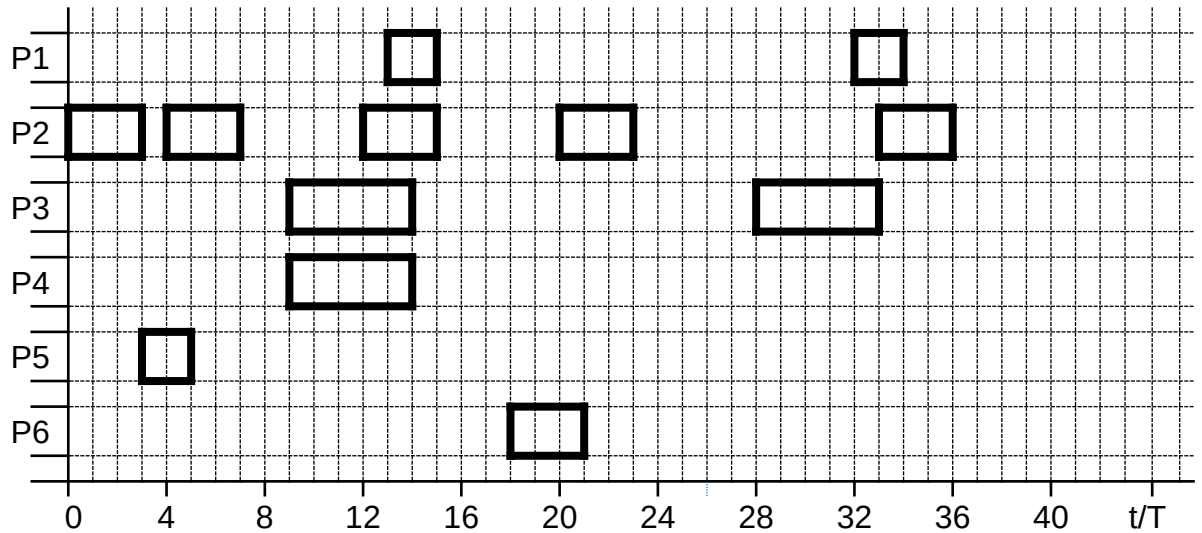


Diagramm BS-1.1: Sollzeitverlauf der Prozesse P1 bis P6

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

Tabelle BS-1.2: Prioritätenverteilung

Hinweis: Füllen Sie die nachfolgenden Aufgaben nach dem folgenden Schema aus (Lösung außerhalb der Lösungskästen werden nicht gewertet):

t/T	**0**			**4**				**8**				**12**				**16**				**20**				
Prozess	**2**	**2**	**1**	**1**					**3**	**3**	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**

t/T	**24**			**28**				**32**				**36**				**40**				**44**				
Prozess	**3**	**3**	**5**	**5**	**5**	**5**	**4**	**3**	**3**	**3**	**3**													

a) In der ersten Teilaufgabe sollen die Prozesse nicht-präemptiv nach ihren Prioritäten eingeplant werden. Wenn keine andere Regel greift, gilt das Prinzip FIFO.

Tragen Sie den Verlauf der Prozessabarbeitung im Folgenden ein.
Hinweis: Nur Lösungen innerhalb der Lösungskästen werden gewertet!

t/T	0			4				8				12				16				20				
Prozess	2	2	2	5	5	2	2	2		3	3	3	3	3	1	1	2	2	2	4	4	4	4	4

t/T	24			28				32				36				40				44				
Prozess	2	2	2	6	6	6	3	3	3	3	3	1	1	2	2	2								

b) In der zweiten Teilaufgabe soll die Einplanung der Prozesse präemptiv mit festen Prioritäten erfolgen. Für jedes Prioritätslevel soll dabei ein eigenes Round-Robin-Verfahren angewendet werden. 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 **3T**. Restzeiten können nicht übersprungen werden.

Tragen Sie den Verlauf der Prozessabarbeitung im Folgenden ein.
Hinweis: Nur Lösungen innerhalb der Lösungskästen werden gewertet!

t/T	0			4				8				12				16				20				
Prozess	2	2	2	5	2	2	2	5		3	3	3	2	1	1		2	2	3	3		2	2	2

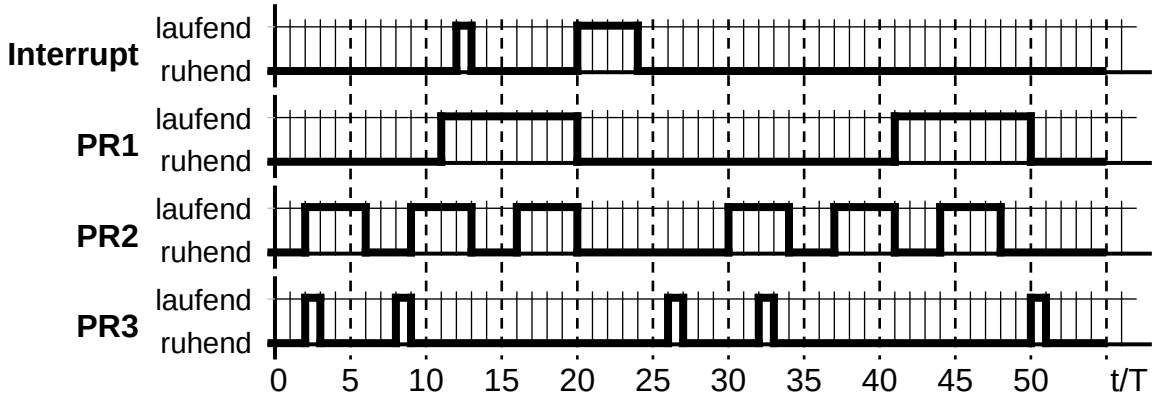
t/T	24			28				32				36				40				44				
Prozess	4	4	4	6	3	3	3	3	1	1		3		2	2	2	6	6	4	4				

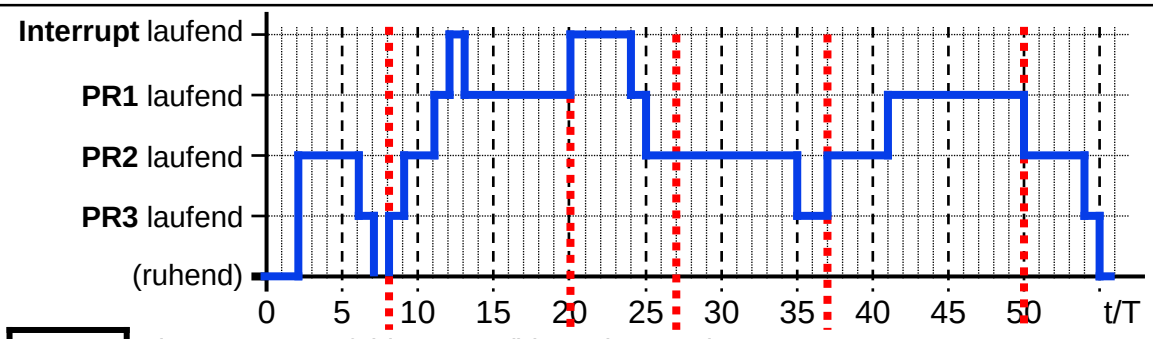
2. Asynchrone Programmierung

Die zwei periodischen Prozesse PR2 und PR3 sowie der asynchrone Prozess PR1 sollen mit dem Verfahren der asynchronen Programmierung präemptiv auf einem Einkernprozessor eingeplant werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigste Priorität. Die Ausführung wird durch einen Interrupts unterbrochen.

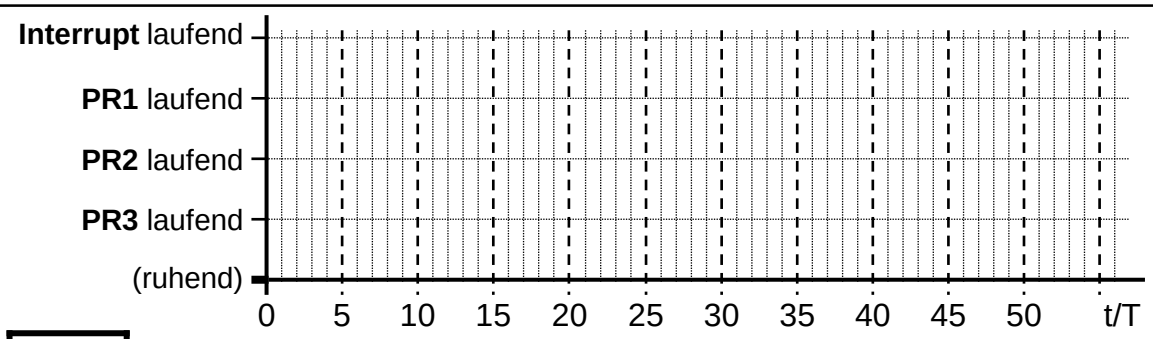
Tragen Sie in das unten angegebene leere Diagramm den Verlauf der Abarbeitung von Programmen und Interrupts ein.

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





☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*



☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*

☐

3. Semaphoren

Die vier Tasks A, B, C und D sollen mit Hilfe von Semaphoren in die Abfolgesequenz $ABA\overline{CD}$ (ABACDCDCD…) gebracht werden. Entwickeln Sie für die dafür gegebenen vier Semaphoren S1, S2, S3 und S4 eine passende Anordnung von Semaphoroperationen. Ein Initialwert ist in diesem Fall schon vorgegeben. Ergänzen Sie fehlende Initialwerte!

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

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

Semaphore:	S1	S2	S3	S4
Initialwert:	*3*	**1**	**0**	**0**

Task:	A	B	C	D
Semaphor-operationen ↓	**P(S1)** **P(S1)** **V(S2)** **V(S3)** **Alternative:** **P(S1)** **P(S1)** **P(S1)** **V(S2)** **V(S3)**	**P(S2)** **P(S2)** **V(S1)** **V(S3)** **P(S2)** **P(S2)** **V(S1)** **V(S1)** **V(S1)**	**P(S3)** **P(S3)** **P(S3)** **V(S4)** **P(S3)** **P(S3)** **V(S4)**	**P(S4)** **V(S3)** **V(S3)** **V(S3)** **P(S4)** **V(S3)** **V(S3)**
	Weitere Alternativen nicht ausgeschlossen			

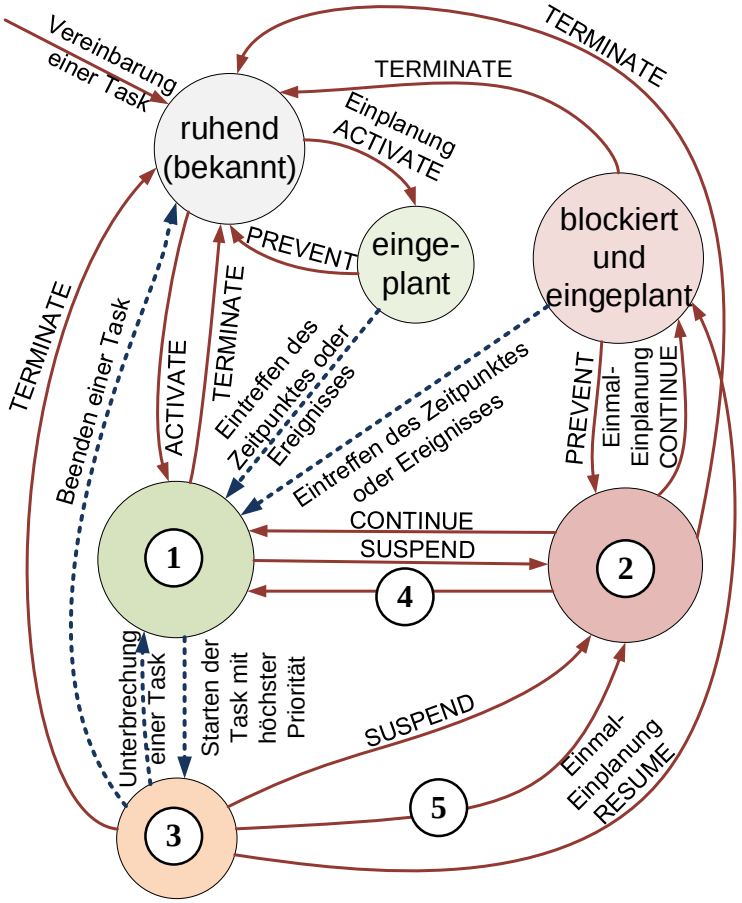
zu definierende Taskfolge: $\boldsymbol{ABA\overline{CD}}$

Platz für Nebenrechnungen:

4. PEARL

Im Folgenden sehen Sie das erweiterte Zustandsdiagramm einer Task mit fünf Grundzuständen. Dieses weist allerdings Lücken auf hinsichtlich der Bezeichnung der Zustände und ihrer Übergänge.

a) Füllen Sie die nummerierten Lücken des erweiterten Taskmodells.



1	**bereit / lauffähig**
2	**blockiert / angehalten**
3	**laufend**
4	**RELEASE**
5	**(erfolgloses) REQUEST**

b) Der eine Teil der Übergänge (Pfeile) zwischen den Zuständen ist gestrichelt, der andere Teil mit durchgängiger Linie eingezeichnet. Wodurch werden die Übergänge jeweils gesteuert?

Durchgezogen:

gesteuert durch PEARL

Gestrichelt:

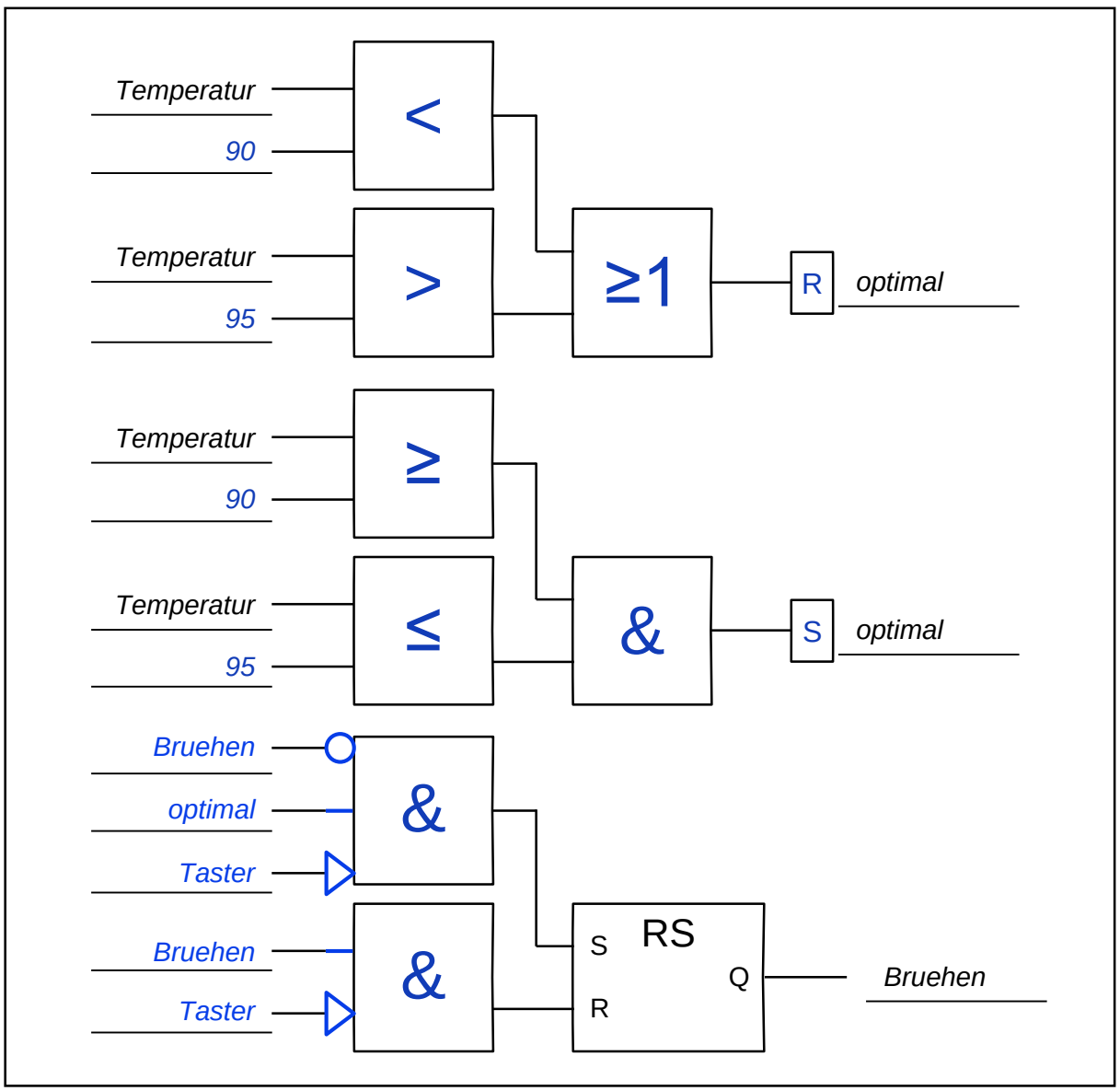
Gesteuert durch das Betriebssystem

5. IEC 61131-3: Funktionsbausteinsprache (FBS)

Eine Kaffeemaschinensteuerung soll mittels Funktionsbausteinsprache (FBS) realisiert werden. Das zu implementierende Programm soll dabei folgenden Ablauf verfolgen: Der optimale Temperaturbereich (Variable *optimal*) liegt zwischen 90 °C und 95 °C (Grenzwerte jeweils eingeschlossen). Wird, wenn die optimale Temperatur erreicht wurde und der Brühvorgang noch nicht läuft (Aktor *Bruehen*), der Taster gedrückt (Sensor *Taster*), so wird der Brühvorgang gestartet (Aktor *Bruehen*). Wird der Taster während des Brühvorgangs wieder gedrückt, so wird der Brühvorgang gestoppt.

Nutzen Sie zur Realisierung des Ablaufs der Steuerung die Vorlage im Lösungsfeld.

Name	Typ	Datentyp	Beschreibung
Temperatur	Sensor	INT	Temperatur in °C
optimal	Variable	BOOL	Optimaler Bereich erreicht (true) oder nicht (false)
Taster	Sensor	BOOL	Taster gedrückt (true) oder nicht (false)
Bruehen	Aktor	BOOL	Brühvorgang läuft (true) oder nicht (false)

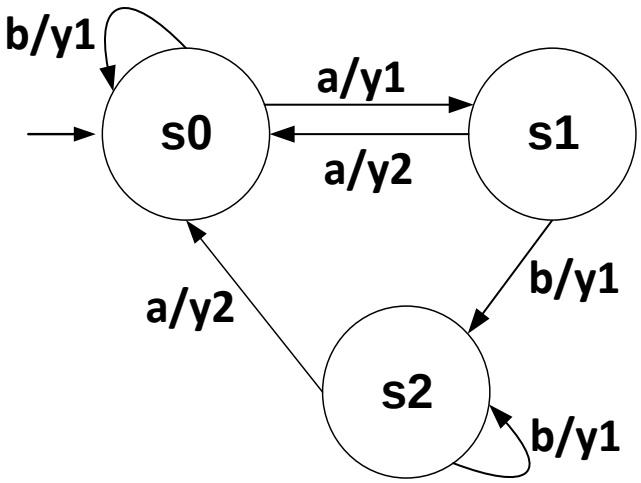


Aufgabe MSE Teil 1: Automaten und Zustandsdiagramm

Aufgabe MSE1: 24 Punkte

1. Moore-/Mealy-Automat

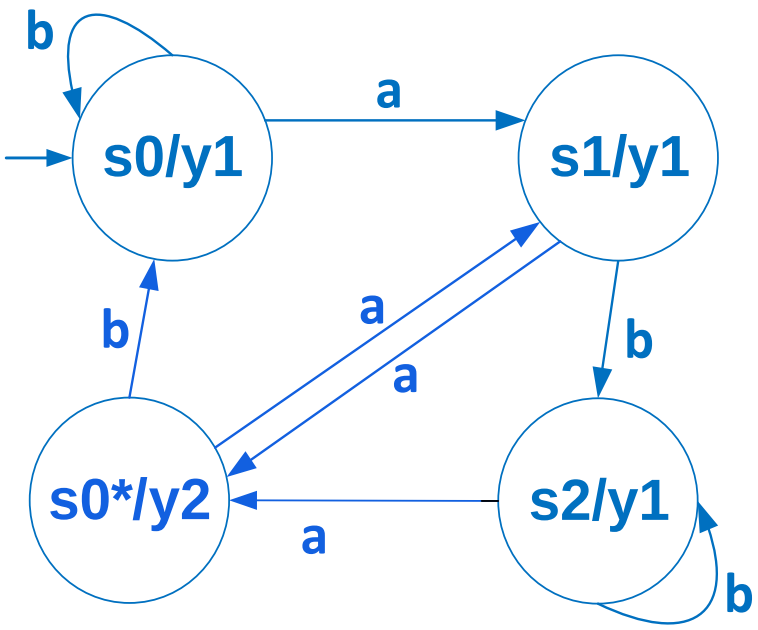
Gegeben ist folgender Automat. Bestimmen Sie, ob es sich um einen Moore- oder einen Mealy-Automaten handelt und wandeln Sie den Automaten in die äquivalente andere Form um (Moore → Mealy bzw. Mealy → Moore; Ergänzen Sie bei Bedarf zusätzliche Zustände).



Art des Automaten:

Mealy-Automat

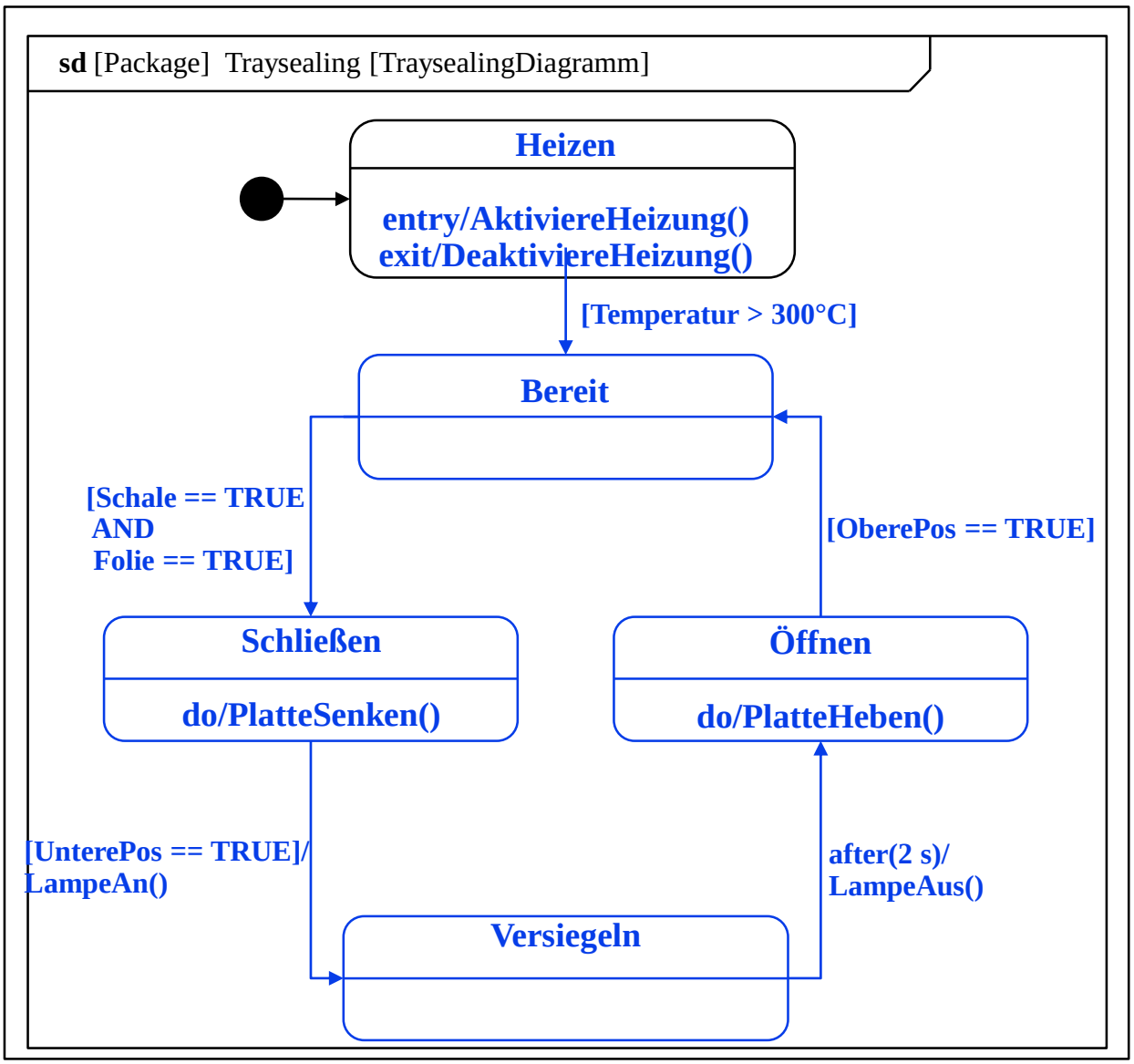
Umwandlung:



2. Zustandsdiagramm

Im Folgenden soll der Funktionsablauf für die Verschweißung von Plastikschalen mit einer Folie als Zustandsdiagramm modelliert werden. Nach dem Start geht die Maschine in den Zustand *Heizen* über, bei dessen Betreten die Aktion *AktiviereHeizung* ausgeführt wird. Übersteigt der Sensor *Temperatur* den Wert von 300°C, wird der Zustand wieder verlassen. Dabei wird noch im Zustand *Heizen* die Heizung durch die Aktion *DeaktiviereHeizung* wieder abgeschaltet und die Maschine wechselt in den Zustand *Bereit*. Ist eine Schale zur Verschweißung vorhanden (Sensorwert *Schale* ist TRUE) und auch mit Folie überspannt (Senorwert *Folie* ist TRUE), wird in den Zustand *Schließen* übergegangen, der die Aktion *PlatteSenken* besitzt. Erreicht die Heizplatte ihre vorgesehene Position (Sensor *UnterePos* ist TRUE), wird die Aktion *LampeAn* durchgeführt und die Maschine wechselt in den Zustand *Versiegeln*. Dieser wird nach 2 Sekunden wieder verlassen und die Lampe wieder ausgeschaltet (Aktion *LampeAus*). Danach wird im Zustand *Öffnen* die Heizplatte mit der Aktion *PlatteHeben* wieder angehoben, bis ihre obere Position erreicht ist (Sensorwert *OberePos* gleich TRUE). Daraufhin geht die Maschine wieder in den Zustand *Bereit* über.

Vervollständigen Sie das Zustandsdiagramm entsprechend der Angaben.



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

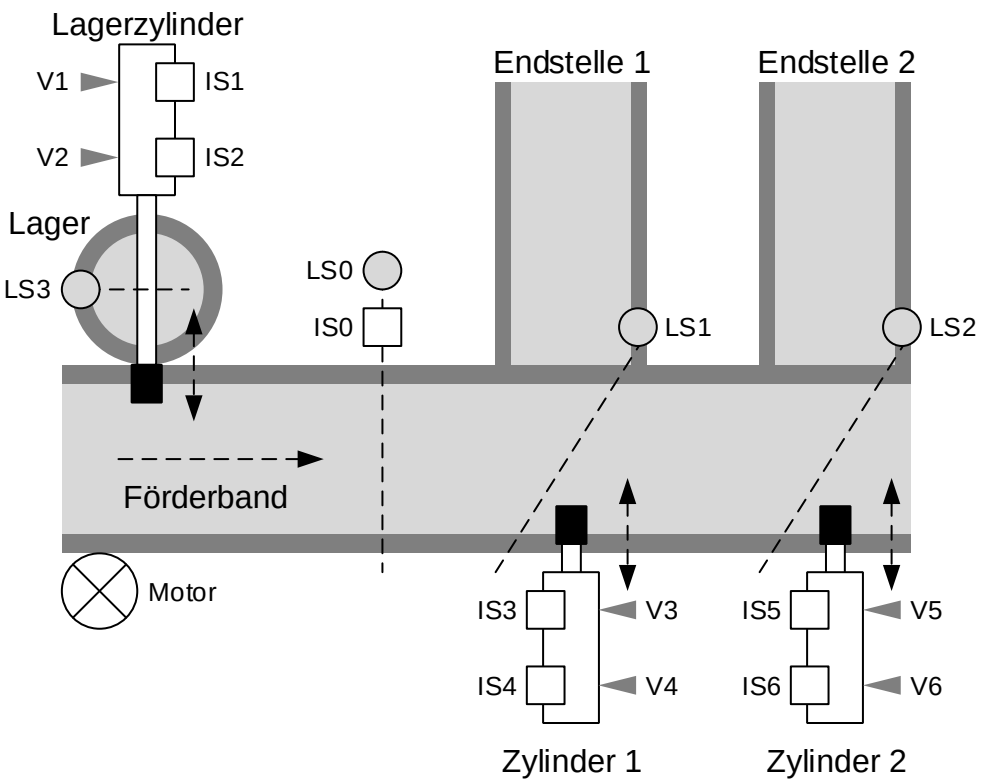
Aufgabe MSE2: 24 Punkte

Für die folgenden Teilaufgaben ist eine Sortieranlage (siehe Abbildungen unten) gegeben.

Aus einem *Lager* (Werkstückerkennung durch Lichtschranke *LS3*) werden Werkstücke mittels eines *Lagerzylinders* vereinzelt und abschließend durch ein *Förderband* transportiert. Das *Förderband* transportiert die Werkstücke – je nachdem ob sie metallisch sind oder nicht – zu den *Endstellen 1* bzw. *2*. Die Erkennung des Werkstücktyps erfolgt dabei durch die Lichtschranke *LS0* und den Induktiven Sensor *IS0*, die an dem Band befestigt sind. Vor den jeweiligen *Endstellen* befinden sich wiederum die *Zylinder 1* bzw. *Zylinder 2*, welche die Werkstücke an den *Endstellen* ausstoßen, wenn sie an den entsprechenden Lichtschranken *LS1* bzw. *LS2* erkannt werden. Metallische Werkstücke sollen sich nach dem Transportiervorgang in *Endstelle 1* befinden; nichtmetallische Werkstücke in *Endstelle 2*.

Alle im System verbauten Zylinder (*Lagerzylinder*, *Zylinder 1* und *Zylinder 2*) sind bistabile Zylinder, d. h. sie beinhalten zwei *Ventilanschlüsse* (V_i) zum Ein- bzw. Ausfahren und zwei *Sensoren* (IS_j) zur Erkennung der Endlagen der Zylinder.

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



Symbol	Bezeichnung	Funktion
○	Lichtschranke (LS)	Erkennt, ob ein Werkstück vorhanden ist oder nicht.
□	Induktiver Sensor (IS)	Erkannt, ob ein Werkstück metallisch ist oder nicht.

1. Flussdiagramm: Werkstücke verarbeiten

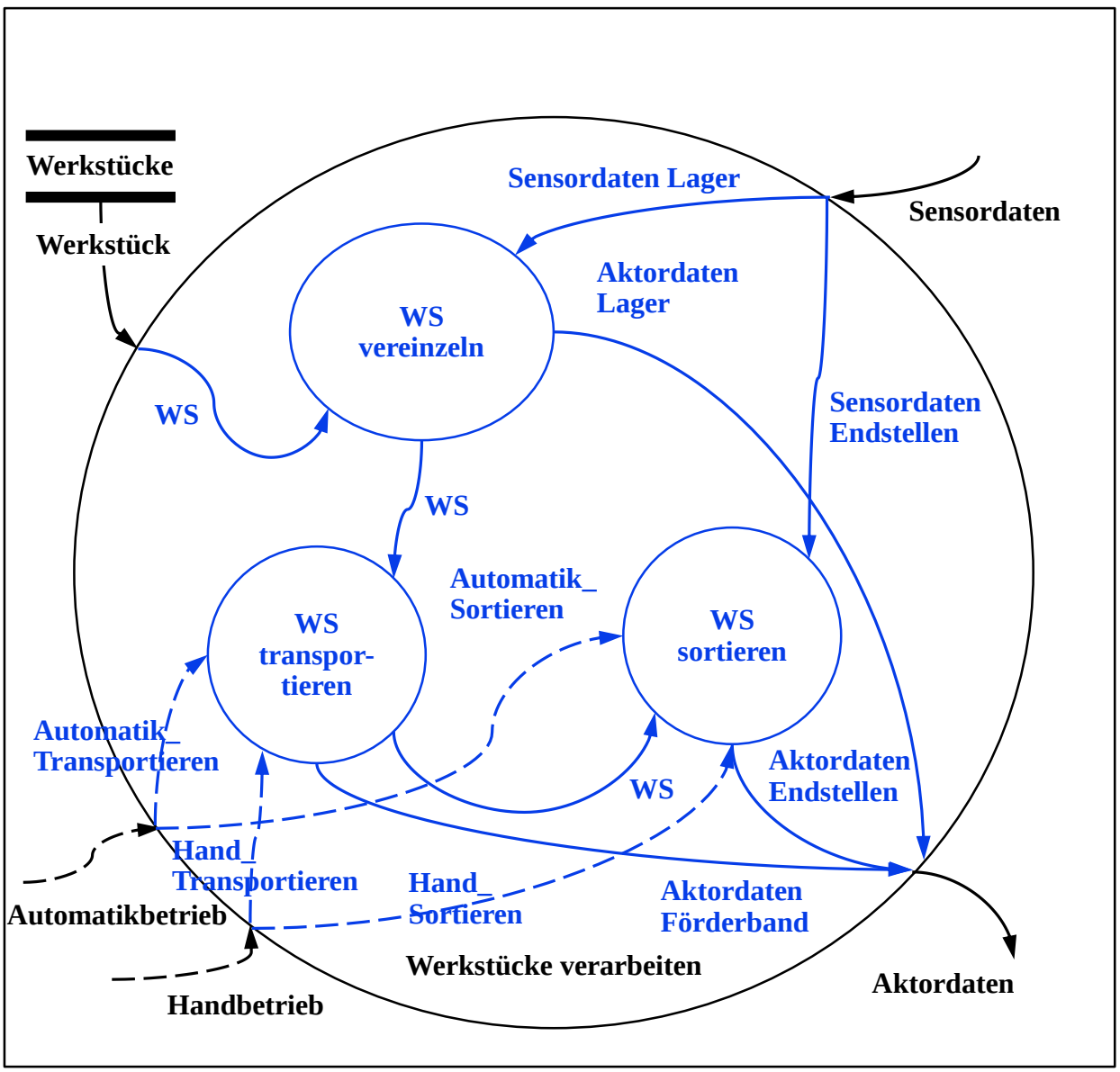
Es soll das F*lussdiagramm* des Prozesses *Werkstücke verarbeiten* modelliert werden. Identifizieren Sie hierzu alle Subprozesse, die zur Realisierung des Prozesses *Werkstücke verarbeiten* notwendig sind. Da die dem Lager nachgelagerten Prozesse kritisch sind, muss hier sowohl ein Hand- als auch ein Automatikbetrieb vorgesehen werden. Verwenden Sie zur Modellierung des Flussdiagramms des Weiteren die folgenden Flüsse:

Datenflüsse Material: Werkstück

Datenflüsse Information:

- **Sensordaten:**
  Sensordaten_Lager, Sensordaten_Endstellen
- **Aktordaten:**
  Aktordaten_Lager, Aktordaten_Förderband, Aktordaten_Endstellen

Steuerflüsse: Handbetrieb, Automatikbetrieb



2. Datenflussdiagramm: Sortiervorgang

Um die spätere Implementierung der Sortieranlage vorzubereiten, möchten Sie den Sortiervorgang in die beiden Endstellen der Sortieranlage detaillieren. Identifizieren Sie hierzu, welche Teilprozesse dem Sortiervorgang zuzuordnen sind und geben Sie eine kurze Beschreibung dieser Teilprozesse an.

Endstelle 1 einsortieren: ***Zylinder 1 fährt aus und transportiert WS in Endstelle 1.***

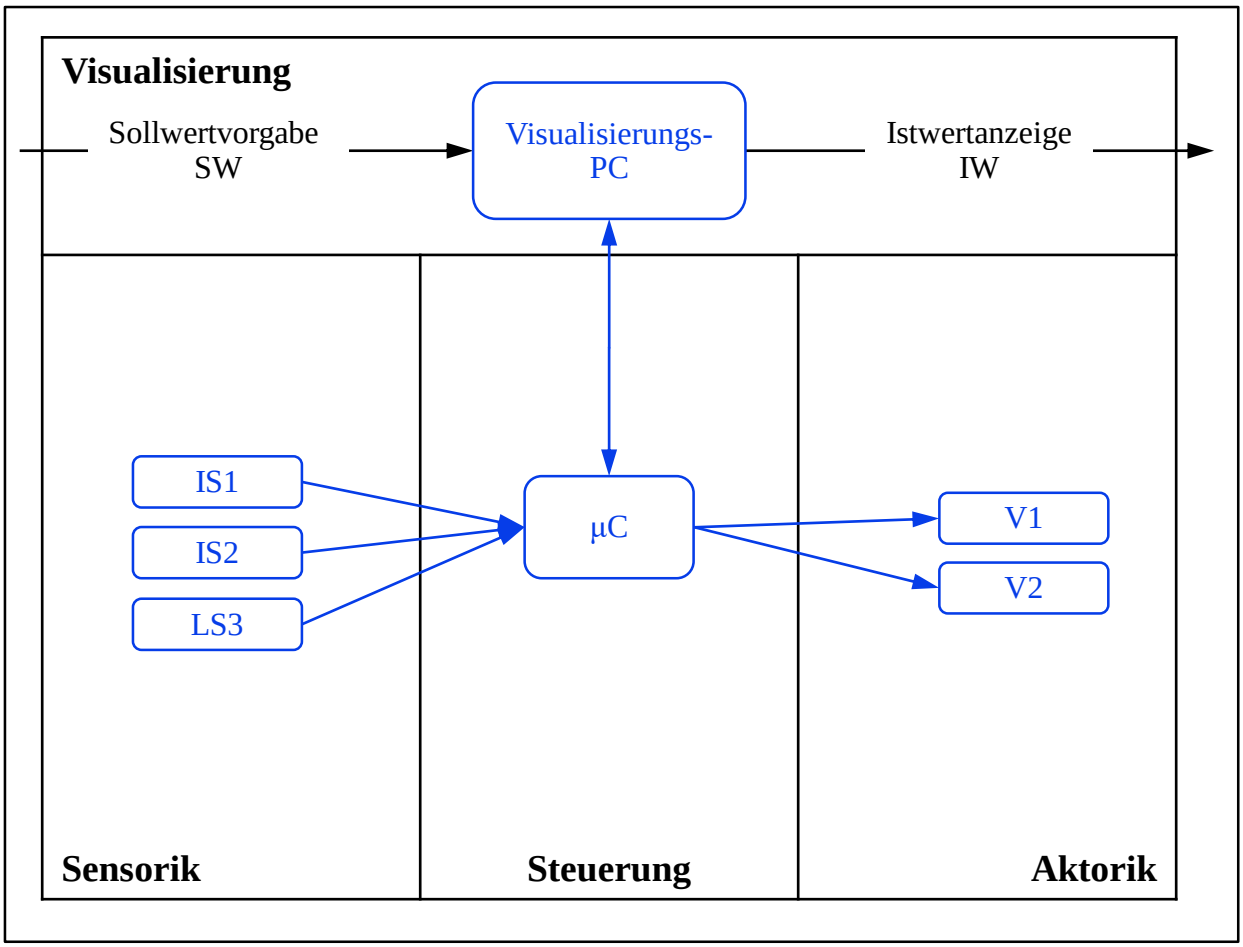
Endstelle 2 einsortieren: ***Zylinder 2 fährt aus und transportiert WS in Endstelle 2.***

WS erkennen: ***WS-Typ wird anhand der Sensoren erkannt, um zu entscheiden, ob in Endstelle 1 oder 2 einsortiert werden soll.***

3. Architekturflussdiagramm

Für das *Lager* der Sortieranlage soll nun das *Architekturflussdiagramm* modelliert werden. Hierzu soll ein *Mikrocontroller (µC)-basiertes System* verwendet werden, in welchem ein dem Lager zugeordneter Mikrocontroller *µC* die Steuerung des Lagers übernimmt und seine Daten mit einem *Visualisierungs-PC* austauscht. Modellieren Sie das *Architekturflussdiagramm* des Lagers. Flüsse von der Umgebung zu den Sensoren bzw. von den Aktoren zu der Umgebung müssen nicht modelliert werden.

Hinweis: Achten Sie darauf alle Sensoren und Aktoren des Lagers zu berücksichtigen!



Aufgabe C: Programmieren in C

Aufgabe C: 96 Punkte

1. Datentypen, Ein-/Ausgabe und Boolesche Algebra

a) Sie sollen folgende Variablen so definieren, dass **so wenig Speicher wie möglich** genutzt wird:

- 8-stellige Matrikelnummer, die nur Ziffern enthält (Variable *matrikelnr*)
- Körpergröße in Metern, auf vier Nachkommastellen genau (Variable *groesse*)
- Alter in Jahren (Variable *alter*)

Kreuzen Sie an, welchen Datentyp Sie für die jeweilige Variable verwenden würden (nur Einfachnennung möglich).

Variable	char	int	double	float	long	short
matrikelnr	☐	☐	☐	☐	☒	☐
groesse	☐	☐	☐	☒	☐	☐
alter	☒	☐	☐	☐	☐	☐

b) Ergänzen Sie die nummerierten Lücken in der formatierten Ausgabe, sodass die **Körpergröße in Zentimetern** ohne Nachkommastellen und das **Alter in Jahren** ausgegeben werden.

```
(1)("Groesse: (2), Alter: (3)", (4));
```

(1)	`printf`	(3)	`%i`
(2)	`%.0f`	(4)	`groesse*100, alter`

c) Für eine Palettierungseinheit möchten Sie Werkstücke, welche in 6er-Paletten geliefert werden, **entweder nur** in 3er- **oder nur** in 4er-Paletten zusammenfassen. Es kommen immer nur volle 6er-Paletten an (d. h. die Variable *anzahl* gibt an, wie viele Werkstücke ankommen und ist immer durch 6 teilbar). Da 4er-Paletten günstiger sind, sind diese stets gegenüber den 3er-Paletten zu bevorzugen. Konstruieren Sie zwei Boolesche Ausdrücke, die Anhand der Variable *anzahl* prüfen, ob 3er- oder 4er-Paletten (Variablen *paletten3er* bzw. *paletten4er*) zu verwenden sind.

```
// Definition der Variablen
int anzahl, paletten3er, paletten4er ;
// Code zum Einlesen der Anzahl
// Überprüfung mittels Boolescher Ausdrücke

paletten3er = (anzahl%3==0) && !(anzahl%4==0) ;
paletten4er = (anzahl%4==0) ;
```

2. Kontrollstrukturen

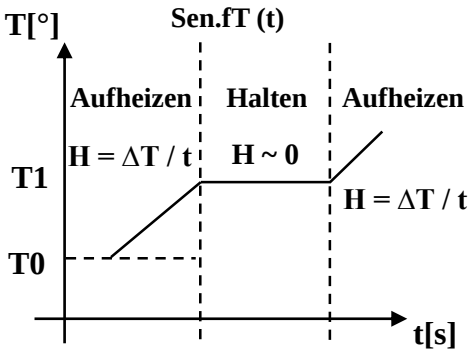
Im Folgenden soll ein C-Programm zur Kontrolle der Durchmesser der zu palettierenden Werkstücke geschrieben werden. Hierzu wird mittels einer einfachen Berechnung der durchschnittlichen Abweichung zwischen Ist- und Solldurchmesser nach möglichen Durchmesserschwankungen der eingehenden Werkstücke gesucht. Dazu wird anhand von *NUM* Werkstücken der Werkstückdurchmesser (Array *diameter*) ermittelt. Der ideale Durchmesser ist zudem gegeben (Variable *ideal*). Schreiben Sie nun ein C-Programm, das die durchschnittliche Abweichung vom idealen Durchmesser berechnet. Es sollen dabei nur jene Werkstücke berücksichtigt werden, die als „gut“ bezeichnet werden (Werkstückdurchmesser soll größer als *9*, aber kleiner als *11* sein). Geben Sie am Ende der Berechnung die Ergebnisse in der Form „Anzahl guter Werkstücke: X, Durchschnittlicher Durchmesser: Y, Durchschnittliche Abweichung: Z“ aus. Verwenden Sie für Ihr C-Programm die folgende Vorlage.

Hinweis: Sie können die Funktion „double fabs(double x)“ zur Berechnung der Betragsfunktion verwenden.

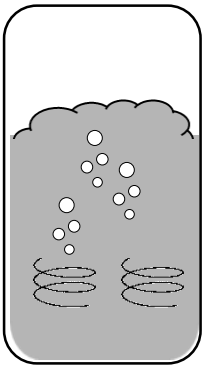
```
#include <stdio.h>
#include <math.h>
#define NUM 10

int main(void) {
  float diameter[NUM] =
    {12 , 9.3 , 9 , 9.2 , 10 , 9.9 , 9 , 10.5 , 10.1 , 8};
  float ideal = 10.0;
  float sumVal = 0.0, sumDev = 0.0, avg = 0.0, dev = 0.0;
  int i=0, j=0;

   for ( i = 0 ; i < NUM ; i++ )
   {
     if (diameter[i] < 11 && diameter[i] > 9 )
     {
       sumVal += diameter[i];
       sumDev += fabs(ideal - diameter[i]);
       j++;
     }
   }
   avg = sumVal / j;
   dev = sumDev / j;

   printf("Anzahl guter Werkstücke: %i,
     Durchschnittlicher Durchmesser: %f,
     Durchschnittliche Abweichung: %f", j, avg, dev);


  return 0;
}
```

3. Zyklische Programmierung einer Brauanlage

Temperaturkurve



Braukessel



Rezept-Array *Cmd*

i	*iMode*	*fT* [°]	*fDauer* [s]
0	1	55.2	80.5
1	2	55.2	62.75
2	1	75.5	35.5
…	…	…	…

iSteps

Die Firma ISA GmbH stellt die Microbrew-Brauanlage „myBrew“ her, welche zwei Modi zum Erhitzen besitzt. Die Anlage kann zum einen (*iMode*=1) den Inhalt des Kessels in einer vorgegebenen Zeitdauer *fDauer* aufheizen und zum anderen diese für eine Zeitdauer *fDauer* halten (*iMode*=2). Die Rezeptschritte sind dazu in einem Struct-Array (vgl. oben rechts) gespeichert welches extern eingelesen wird. Diese werden vom Steuerungsprogramm schrittweise abgearbeitet. Zum Beginn muss hierfür ein Datentyp *REZEPT* definiert werden. Legen Sie in diesem Datentyp *iMode (int)*, *fT (float)* und *fDauer* (*float*) an.

Zustand 0: Zu Anfang ist die Heizung ausgeschaltet. Der Befehlszähler wird um eins erhöht und geprüft welcher Befehl vorliegt. Je nach dem wird nach 0→1 (Aufheizen) oder 0→2 (Halten) gewechselt. Sind alle Befehle abgearbeitet (Anzahl *iSteps*), wird von 0→3 gewechselt und das Programm beendet.

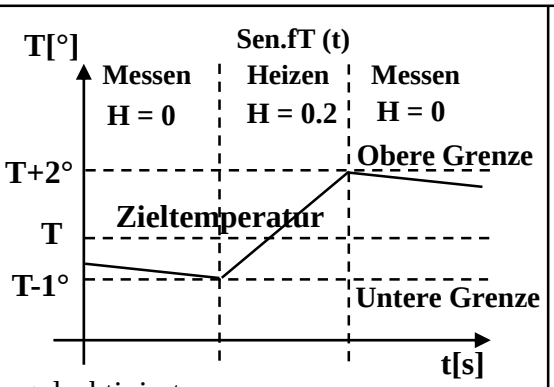
Zustand 1 (Aufheizen): Die Temperatursteigung zum Aufheizen wird aus der Zeitvorgabe des Befehls berechnet (**H = ΔT / t**) und auf den Heizregler *Akt.fH* gesetzt. Nach Überschreitung der Zieltemperatur wird von 1→0 gewechselt um zum nächsten Befehl zu schalten.

Zustand 2 (Halten): Da der Kessel mit der Zeit abkühlt, wird die Zieltemperatur nun für die Dauer *fDauer* gehalten und nach Ablauf von 2→0 gewechselt. Um das Heizen in einem Totbereich um die Zieltemperatur zu vermeiden, wird abwechselnd zwischen zwei Subzuständen hin- und hergeschaltet. (Fortsetzung Folgeseite)

Aktoren:	
Akt.fH	Heizt den Kessel mit der **Temperatursteigung H** (Grad/Sekunde) auf
Sensoren/Merkvariablen:	
Sen.fT	Sensorwert der **aktuellen Temperatur** im Braukessel
Cmd.iMode	Struktur-Variable welche den **Heizmodus** enthält (1/2)
Cmd.fT	Struktur-Variable welche die **Zieltemperatur** enthält
Cmd.fDauer	Struktur-Variable mit **Dauer** des Aufheizen (1) oder Halten (2) in Sek.
iSteps	Gesamtanzahl der Rezeptschritte
i	Aktueller Rezeptschritt
t	Speichert eine Zeit (in s) für die Verwendung als Timer
plcZeit	Systemlaufzeit der Steuerung (in s)
state / substate	Speichert den aktuellen Zustand / Subzustand

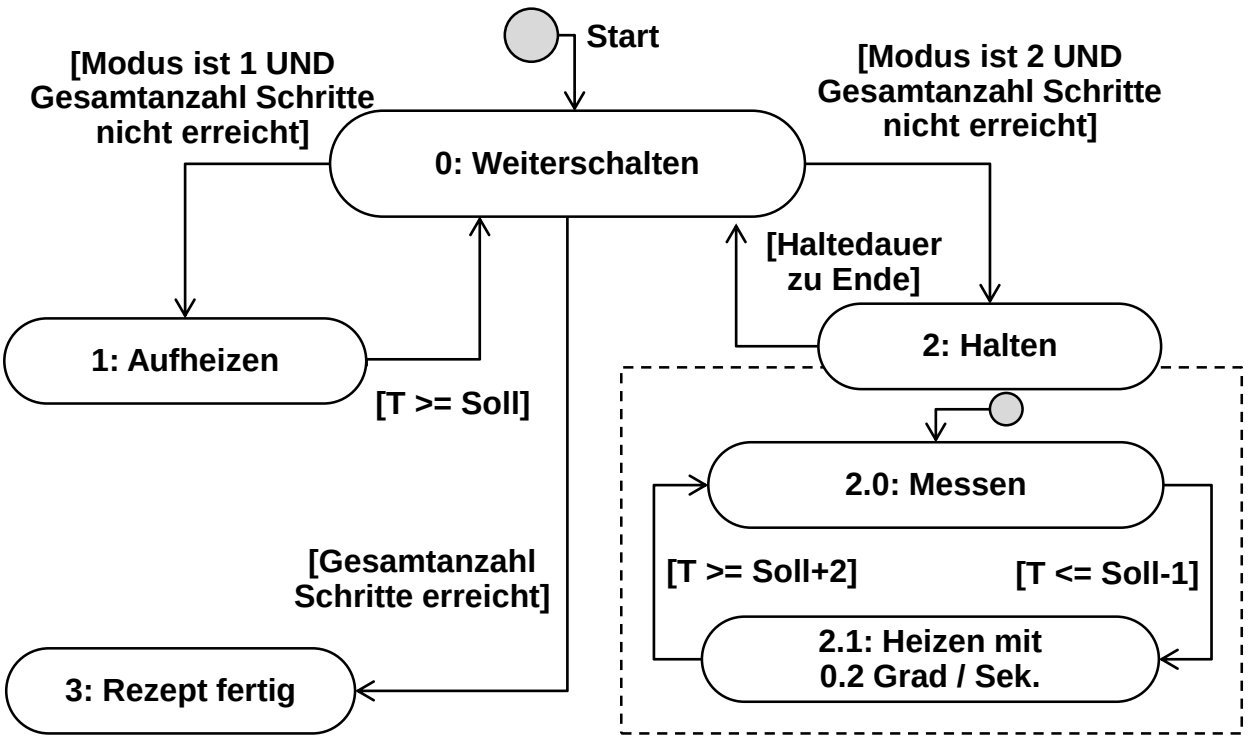
Subzustand 2.0 (Messen): Die Heizung ist ausgeschaltet und es wird gewartet, bis die Zieltemperatur um 1 Grad unterschritten ist, um dann von 2.0→2.1 zu wechseln.

Subzustand 2.1 (Heizen): Die Heizung heizt mit konstanten 0.2 Grad / Sek., bis die Zieltemperatur um 2 Grad überschritten ist. Anschließend wird zum Abkühlen wieder von 2.1→2.0 gewechselt.



Zustand 3: Da das Rezept zu Ende ist, wird die Heizung deaktiviert

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_mybrew.h"

struct SData Sen;          // Sensorvariablen
struct AData Akt;          // Aktorvariablen
unsigned int state=0;      // Zustandsvariable1 (Start in 0)
unsigned int substate=0;   // Zustandsvariable2 (Start in 0)
unsigned int plcZeit=0;    // in Sek.

int t=0;   // Speicher für Timer-Messung (in ms)
int i=-1; // Initialisierung Rezeptschrittzähler
int iSteps=8; // Anzahl Rezeptschritte
```

```
typedef struct          // Typdefinition
{
        float fT;

        float fDauer;

        int iMode;

} REZEPT;

int main()
{while (1){     //Zyklische Ausführung

REZEPT Cmd[iSteps];          // Structarray "Cmd" anlegen

Rezept_einlesen(REZEPT *Cmd, int iSteps);

switch( state )
{

  case 0          : //Weiterschalten

    Akt.fH=0;          // Aktor setzen
    i++;

    if(Cmd[i].iMode==0 && i<iSteps) state=1;     //0->1

    if(Cmd[i].iMode==1 && i<iSteps) state=2;     //0->2

    if(i==iSteps) state=3;                       //0->3

   break;

   case 1          :  //Aufheizen
                      //Einmal berechnen u. Aktor setzen

    if(Akt.fH==0)

    Akt.fH=(Cmd[i].fT-Sen.fT)/Cmd[i].fDauer);

    if(Sen.fT>=Cmd[i].fT) state=0;              //1->0

   break;          //(Fortsetzung naechste Seite)
```

```
  case 2        : // Zustand 2 Halten
       if(!t) t=plcZeit; // Timer setzen
       if(plcZeit-t<Cmd.fDauer[i])    // Vergleich
       {
       switch(  substate  ) // Automat Halten
       case 0        : // Messen
       // Warten bis untere Grenze erreicht 2.0->2.1
       Akt.fH=0;
       if(Sen.fT<=(Cmd[i].fT-1)) substate=1;
       break;
       case 1        : // Heizen
         // Warten bis obere Grenze erreicht 2.1->2.0
       Akt.fH=2;
       if(Sen.fT>=(Cmd[i].fT+2)) substate=0;
       break;
       } else {t=0; state=0;}                 // 2->0
  break;

  case 3        : // Fertig
         Akt.fH=0;          // Aktor setzen
        break;
} return 0;
}
```

4. Speichern von Rezeptdaten in Dateien mit benutzerdefiniertem Namen

Zum Abspeichern von Rezepten sollen diese in eine Datei geschrieben werden, deren Name vom Benutzer eingelesen wird. Dazu soll eine Schleife 5 Dateien anlegen. Der Dateiname wird als String *cDatei* mit bis zu 80 Zeichen gespeichert und durch eine Benutzereingabe eingelesen. Damit die Datei von Texteditoren erkannt wird, muss der String mit der Zeichenkette ".txt" abgeschlossen werden. Legen Sie den String hierfür zunächst an und verwenden Sie nach Einlesen vom Benutzer den Befehl „strcat“, welcher zwei Strings als Parameter übernimmt und den zweiten an das Ende des ersten Strings anheftet. Öffnen Sie danach einen FILE-Stream mit der Bezeichnung *pSpeicher* und erstellen Sie eine Datei mit dem eingelesenen Namen. Schreiben Sie anschließend den extern eingelesenen int-Wert *iRezept* in die Datei.

```
#include <stdio.h>
#include <string.h>
int main()
{
    int iWahl=0, iRezept=0, i=0;
    char szDatei[81];

    char szEnd[]={".txt"};          //Dateiendung

    for(i=0;i<5;i++)
                                    //Schleife
    {
     printf("\nDateiname:");
                                    //Benutzer fragen
     fflush(stdin);            // Tastaturpuffer leeren

     gets(szDatei); oder: scanf("%80s",szDatei);
     oder: fgets(szDatei,80);       // String lesen

     strcat(szDatei,szEnd);
                                    // strcat

     FILE *pSpeicher=fopen(szDatei,"w");// FILE
     iRezept=readreceipe(); // Rezeptwerte setzen

     fprintf(pSpeicher,"%i",iRezept);
                                    // Schreiben
     fclose(pSpeicher);
    }    return 0;}
```

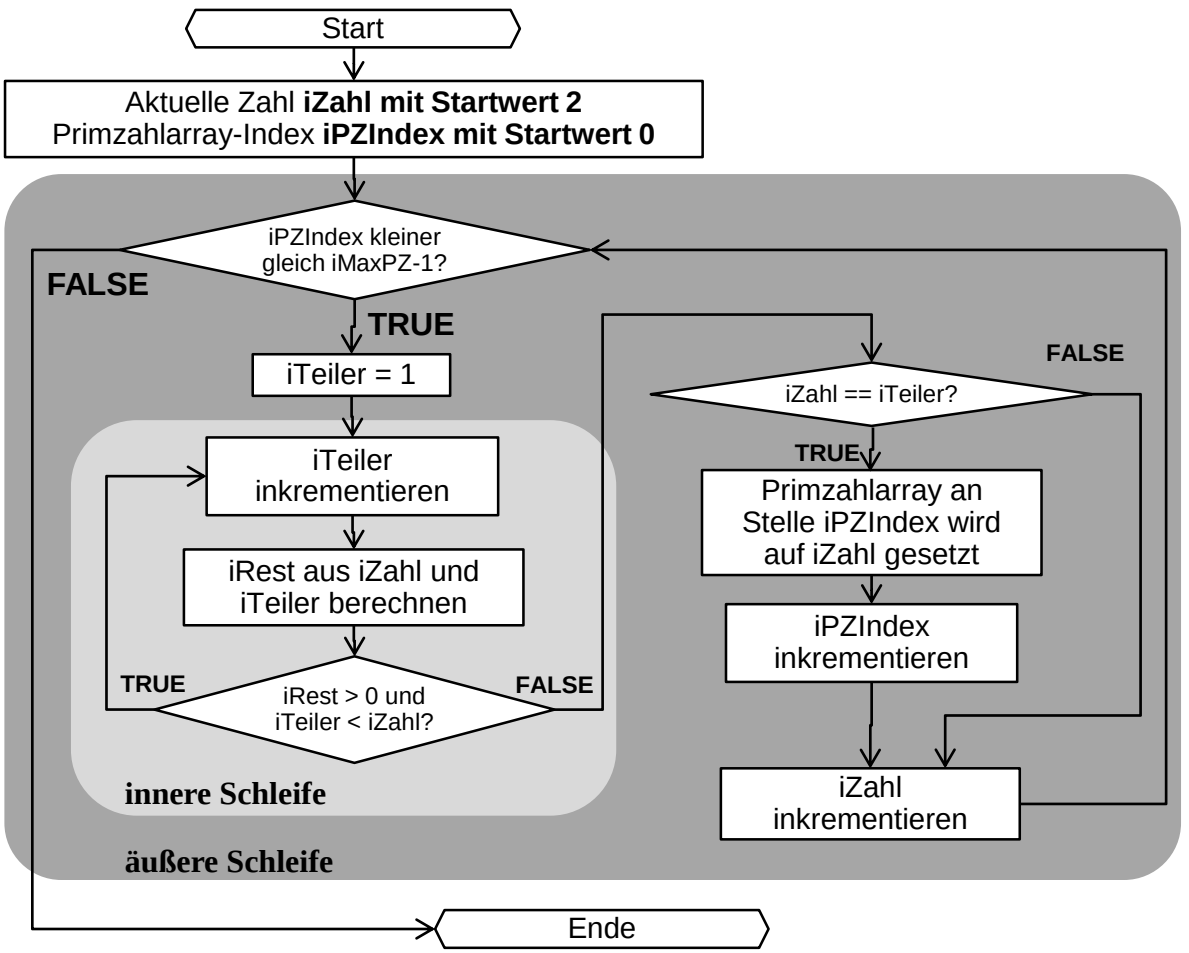
5. Finden von Primzahlen

Zur systematischen Ermittlung von Primzahlen soll ein Algorithmus nach untenstehendem Ablaufdiagramm programmiert werden. Der Algorithmus soll beginnend ab der Zahl 2 eine bestimmte Anzahl Primzahlen („iMaxPZ“) finden und über einen Pointer in ein Array speichern.

Eine Primzahl ist eine natürliche Zahl größer als 1, die nur durch sich selbst und durch 1 ganzzahlig (ohne Rest) teilbar ist. Deshalb werden die natürlichen Zahlen in einer Schleife (äußere Schleife) über die Variable „iZahl“ durchlaufen und für die jeweilige Zahl in einer eingebetteten Schleife (innere Schleife) geprüft, ob diese durch eine kleinere Zahl ohne Rest teilbar ist. Dazu wird dort für Teiler („iTeiler“) beginnend ab dem Wert 2 der Rest der Division „iZahl / iTeiler“ berechnet und in „iRest“ gespeichert. Ist „iZahl“ nicht ohne Rest durch „iTeiler“ teilbar und ist „iTeiler“ noch kleiner als „iZahl“ wird dieser Vorgang mit dem nächst höheren Teiler wiederholt.

Nach Durchlaufen der inneren Schleife wird anhand der vorliegenden Informationen bewertet, ob es sich bei „iZahl“ um eine Primzahl handelt. Ist dies der Fall, wird die Zahl in das per Pointer übergebene Array an die durch „iPZIndex“ festgelegte Stelle geschrieben und dieser Index anschließend erhöht.

Zuletzt wird die Zahl „iZahl“ um eins erhöht und der beschriebene Vorgang wiederholt, solange das Array noch nicht gefüllt ist.



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

Die verwendete Schleife benötigt die Größe des Arrays, welche durch die int-Variable „iMaxPZ“ an die Funktion per Kopie (*call-by-value*) übergeben wird.

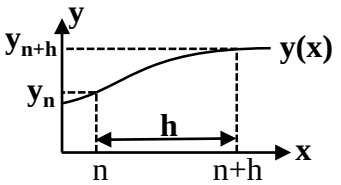
Hinweis: Beachten Sie die zugehörigen Kommentare im Code zur Vervollständigung!

```
void primzahlFueller(int* piPZArray, int iMaxPZ      )
{
    int iTeiler, iRest;
    int iPZIndex = 0;
    int iZahl = 2;
    Alt.: for(;iPZIndex<=iMaxPZ-1; iZahl++)
    while(iPZIndex <= iMaxPZ-1)     //äußere Schleife
    {
        iTeiler = 1;

        do                          //innere Schleife
        {
            iTeiler++;
            //Rest berechnen

            iRest = iZahl % iTeiler;

        } while(iRest > 0 && iTeiler < iZahl);

        //Wenn Primzahl, dann in Array speichern

        if( iZahl == iTeiler                        )
        {   Alt.: *(piPZArray+iPZIndex) = iZahl;
            piPZArray[iPZIndex] = iZahl;

            iPZIndex++;
        }

        iZahl++;
    }   Alt. Mit for braucht hier kein Inkrement.
}
```

6. Funktions-Zeiger

Im folgenden Programm soll zunächst eine numerische Ableitungsfunktion „num_deriv“ vervollständigt werden, die über einen Funktionszeiger auf mathematische Funktionen zugreifen kann und deren Ableitung an der Stelle „n“ mit Hilfe des Abstandes „h“ nach folgender Formel berechnet:

$$y'(n) \approx \frac{y(n+h) - y(n)}{h}$$



Definieren Sie für „num_deriv“ einen Parameter, der die Übergabe von Funktionspointern für Funktionen mit Rückgabewert „double“ und einem „call-by-value“-Parameter mit Datentyp „double“ erlaubt. **Ergänzen Sie dann die Berechnung der Ableitung**.

Im Hauptprogramm soll die Funktion „num_deriv“ eingesetzt werden, um nacheinander in einer For-Schleife die Ableitung der Funktionen „sin“, „cos“ und „tan“ aus der „math.h“ an der Stelle „n“ mit dem Abstand „h“ zu berechnen. **Legen Sie dafür ein passendes Funktionszeiger-Array an** und **rufen Sie innerhalb der For-Schleife die Funktion „num_deriv“ mit passenden Parametern auf**. Beachten Sie dabei, dass „n“ und „h“ vorher über Benutzereingabe (scanf) eingelesen werden.

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

double num_deriv(_____double (*f) (double)_____, double n, double h)
{
     return _____((*f)(n+h) - (*f)(n))_____ / h;
}

int main()
{
    int i = 0;
    double n, h;
    //Funktionszeiger-Array für sin, cos und tan

    ___double (*fn[3])(double) = {sin, cos, tan};_____ ;
    //Einlesen der Werte für n und h
    printf("n (x-Wert): ");
    scanf("%lf", &n);
    printf("h (Genauigkeit): ");
    scanf("%lf", &h);

    //Berechnung und Ausgabe der Ableitungen
    for(i=0; i<3; i++)
    {
        printf("\n%i: %f", i, ____num_deriv(fn[i], n, h)_____);
    }

    return 0;
}
```