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

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Sommersemester 2017

04.09.2017

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält 32 nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

Bitte nicht rot oder grün schreibenden Stiften oder Bleistift ausfüllen!

Aufgabe	Erreichte Punkte
1	
2	
3	
4	
5	
6	
7	
∑G	
8	
9	
10	
11	
12	
13	
∑BS	
14	
15	
16	
17	
∑MSE	
18	
19	
20	
21	
22	
∑C	
∑	

Vorname Nachname _____ Matrikelnummer _____

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 (2A0)$_{11}$ = (**352**)$_{10}$ = (**540**)$_{8}$

2 (18,625)$_{10}$ = (**10010,101**)$_{2}$

2. IEEE 754 Gleitkommazahlen

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

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

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

Vorzeichen

V= **„-“ oder „negativ“**

Bias und biased Exponent

B= **$R^{4-1}-1=7$** **e =** **10**

Exponent

E = **e – B = 10 – 7 = 3**

Mantisse (Dualzahl und Denormalisiert)

M_2= **$(1{,}01011*2^3)_2 = (1010{,}11)_2$**

Vollständige Dezimalzahl Z (inkl. Vorzeichen)

Z= **-10,75**

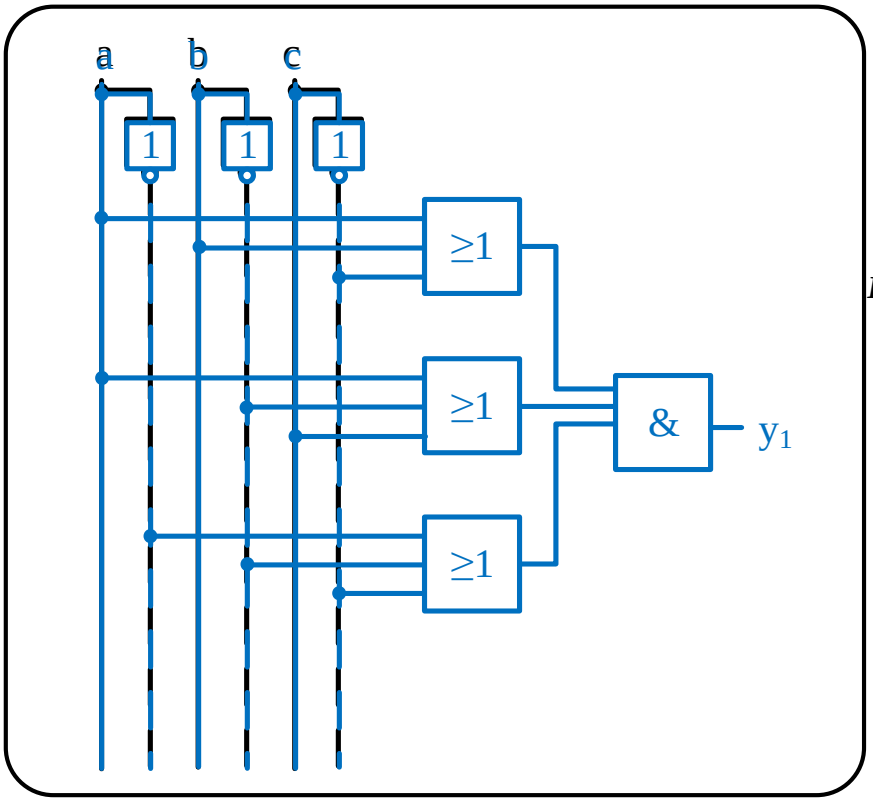
Vorname Nachname Matrikelnummer

3. Logische Schaltungen und Schaltbilder

Sie sind zuständig für den Schaltungsentwurf. Ihnen wurde die angegebene Wahrheitstabelle (Bild G-3.1) übergeben. Erstellen Sie eine *graphische Schaltung in Normalform* (KNF / DNF). Erstellen sie diejenige Normalform, die am *wenigsten Schaltglieder* erfordert.

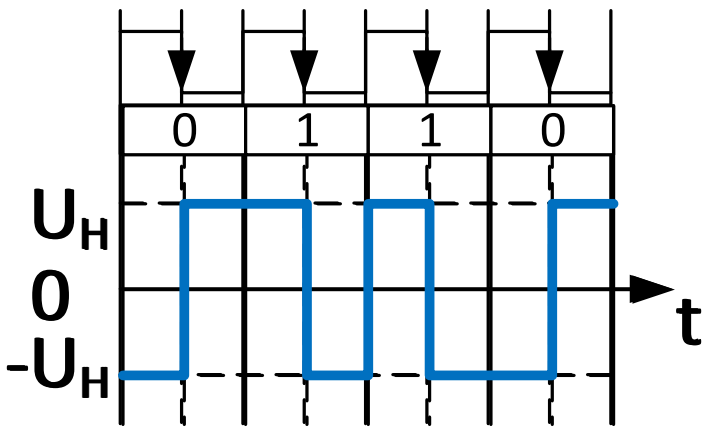
a	b	c	y_1
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Bild G-3.1: *Wahrheitstabelle*



4. Leitungscodes

Die folgende *Binärfolge* soll mit Hilfe eines seriellen Bussystems übertragen werden. Hierfür müssen die Daten *serialisiert* werden. Verwenden Sie dazu den *Manchester-Code*. *Hinweis:* Beachten Sie das angegebene Taktsignal.



5. Normalformen und Minimierung

Gegeben sind folgende KV-Diagramme.

☐ **Dieses KV-Diagramm werten**

	a	a	$\bar{a}$	$\bar{a}$
b	X	1	1	1
$\bar{b}$	1	0	0	X
	$\bar{c}$	c	$\bar{c}$	

Bild G-5.1: *KV-Diagramm 1*

☐ **Dieses KV-Diagramm werten**

	a	a	$\bar{a}$	$\bar{a}$
b	X	1	1	1
$\bar{b}$	1	0	0	X
	$\bar{c}$	c	$\bar{c}$	

Bild G-5.2: *KV-Diagramm 2*

a) Minimieren Sie das KV-Diagramm in Form der DNF *(Disjunktive Normalform)* durch Einrahmen (Schleifen) der entsprechenden Felder im oben dargestellten KV-Diagramm. Schreiben Sie die minimierte Funktion in boolescher Algebra in das Lösungsfeld unten auf. Die Felder mit y=„X“ sind don‘t care bits.

Hinweis: Das zweite abgebildete KV-Diagramm dient als Ersatz, falls Sie sich verzeichnen. Kennzeichnen Sie durch Ankreuzen im Feld „dieses KV-Diagramm werten", welches KV-Diagramm bewertet werden soll.

☐

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

b) Übertragen Sie die gegebene Wahrheitstabelle (Bild G-5,3) in eine Konjunktive Normalform (KNF) der booleschen Algebra und minimieren Sie die Schaltung mithilfe der Rechenregeln der booleschen Algebra.

Hinweis: Schreiben Sie alle Zwischenschritte in das Lösungsfeld!

a	b	y_1
0	0	0
0	1	1
1	0	0
1	1	1

Bild G-5.3: *Wahrheitstabelle*

Term 1 **Term 2**

$y_1 = (a \vee b) \wedge (\bar{a} \vee b)$

Term 1 mit Term 2

$y_1 = (a \wedge \bar{a}) \vee b$

Letzten beiden Terme

$y_1 = b$

☐

6. Flip-Flops

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

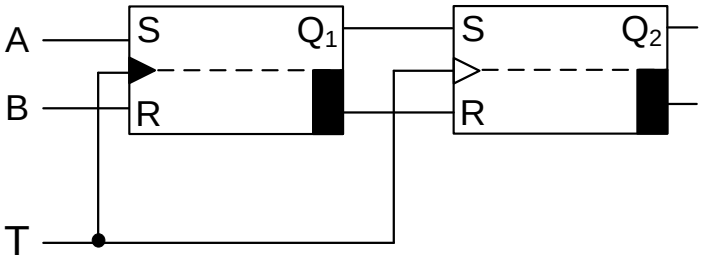
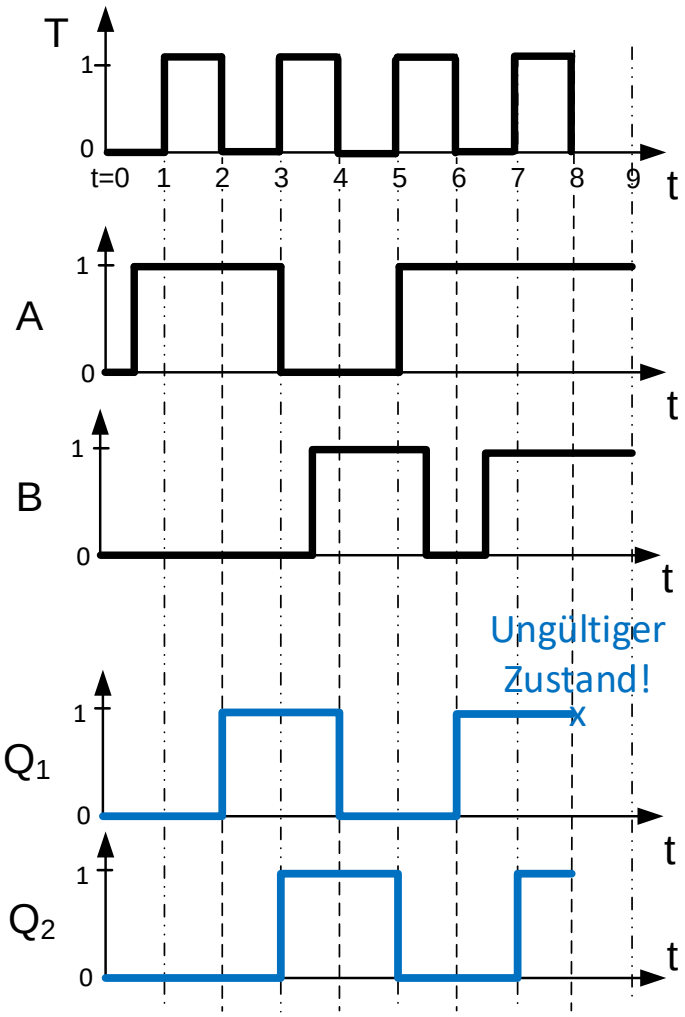


Bild G-6.1: *MS-FF*

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

Analysieren Sie die Schaltung für den Bereich $t = [0; 9[$, 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.



7. MMIX-Rechner

Im Registerspeicher eines MMIX-Rechners befinden sich die in Bild G-7.2 gegebenen Werte. Es sollen nacheinander die zwei Befehle (Bild G-7.4) abgearbeitet und das Ergebnis in dem Registerspeicher (Bild G-7.2) bzw. Datenspeicher (Bild G-7.3) abgelegt werden.

	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	...

Bild G-7.1: *MMIX-Code-Tabelle*

Registerspeicher	
Adresse	Wert vor Befehlsausführung
...	...
$0x87	0x00 00 00 00 00 01 B0 0F
$0x88	0x00 00 00 00 DE AC 10 CF
$0x89	0x00 00 00 00 00 00 00 00
$0x8A	0x00 00 00 00 00 00 61 FE
...	...

Bild G-7.2: *Registerspeicher*

Datenspeicher	
Adresse	Wert
...	...
0x00 00 00 00 00 00 61 FF	0xF0
0x00 00 00 00 00 00 62 00	0x01
0x00 00 00 00 00 00 62 01	0xDA
0x00 00 00 00 00 00 62 02	0x53
0x00 00 00 00 00 00 62 03	0x1B
0x00 00 00 00 00 00 62 04	0x00
0x00 00 00 00 00 00 62 05	0xB0
...	...

Bild G-7.3: *Datenspeicher*

Nr.	Maschinensprache	Assemblersprache	Befehlsbeschreibung
1	0x21 89 87 FF	?	?
2	?	?	$0x89=$M_4$[$0x8A+0x04]

Bild G-7.4: *Befehle in Maschinensprache, Assemblersprache und Befehlsbeschreibung*

Bearbeiten Sie nun folgende Fragen (nächste Seite) zu den Befehlen und zu den Änderungen, die sich durch die Befehle ergeben.

Vorname Nachname Matrikelnummer

Unterstrichenes ist wichtig der Rest ist Herleitung / Erklärung, z.B. 0x oder die führenden Nullen kann man auch weglassen.

Befehl Zeile 1:

a) Geben Sie die Formulierung in Assemblersprache für den Befehl Nr. 1 in Bild G-7.4 an.

0x21 89 87 FF → ADDI $0x89, $0x87, 0xFF

b) Wie lautet die Befehlsbeschreibung des in Nr. 1 in Bild G-7.4 angegebenen Befehls?
Hinweis: Ein Beispiel für eine Befehlsbeschreibung ist in Bild G-7.4, Zeile Nr. 2 gegeben.

$0x89 = $0x87 + 0xFF, oder „Speichert die Summe des Registerinhalts $0x87 und des Sofortoperanden 0xFF in Register $0x89“

c) Geben Sie die Adresse und den Wert der durch den Befehl Nr. 1 in Bild G-7.4 geänderten Registerspeicherzelle an!

Adresse: 0x89

Schriftliche Addition:

```
  00 01 B0 0F
+ 00 00 00 FF
     1  1
```

Wert: 0x0..00 00 01 B1 0E

Befehl Zeile 2:

d) Geben Sie die Formulierung in Maschinensprache für Befehl Nr. 2 in Bild G-7.4 an.
Hinweis: Ein Beispiel für einen Befehl in Maschinensprache ist in Bild G-7.4, Zeile Nr. 1 gegeben.

$0x89=$M_4$[$0x8A+0x04] → LDTI $0x89, $0x8A, 0x04 → 0x89 89 8A 04

e) Geben Sie den Wert der durch Befehl Nr. 2 in Bild G-7.4 geänderten Register- oder Datenspeicherzelle an!

$0x8A = 0x0..61 FE → 0x 0x0..61 FE + 0x04 = 0x0..62 02 → nächste Datenspeicherzelle, die durch 4 teilbar ist (Tetra!): 0x0..62 00 → Lese das Tetra ab 0x0..62 00 und speichere den Inhalt in Registerspeicher 0x89

→ 0x0..00 01 DA 53 1B

Allgemeines zu MMIX:

f) Beantworten Sie die folgenden, allgemeinen Fragen zur Architektur von MMIX.

In welcher Komponente ist der MMIX-Programmcode gespeichert?

(X) Befehlsspeicher ( ) Registerspeicher ( ) Datenspeicher

Welche Komponente ist für die Ausführung mathematischer Operationen zuständig?

( ) Steuerung (X) ALU ( ) Demultiplexer

Worüber wird der Signalfluss in der MMIX-Architektur umgesetzt?

( ) ADD ( ) BZ (X) Multiplexer

Vorname Nachname Matrikelnummer

Aufgabe BS: Betriebssysteme

Aufgabe BS:
48 Punkte

8. Zeitparameter von Rechenzuständen

Gegeben sei das folgende Prozessmodell (Bild BS-8.1). Ordnen Sie jeweils die entsprechenden Zeitparameter den Lücken im Modell zu. Beantworten Sie außerdem die untenstehende Frage bezüglich Scheduling-Algorithmen.

Hinweis: Nur Antworten innerhalb der Lösungskästen werden gewertet!

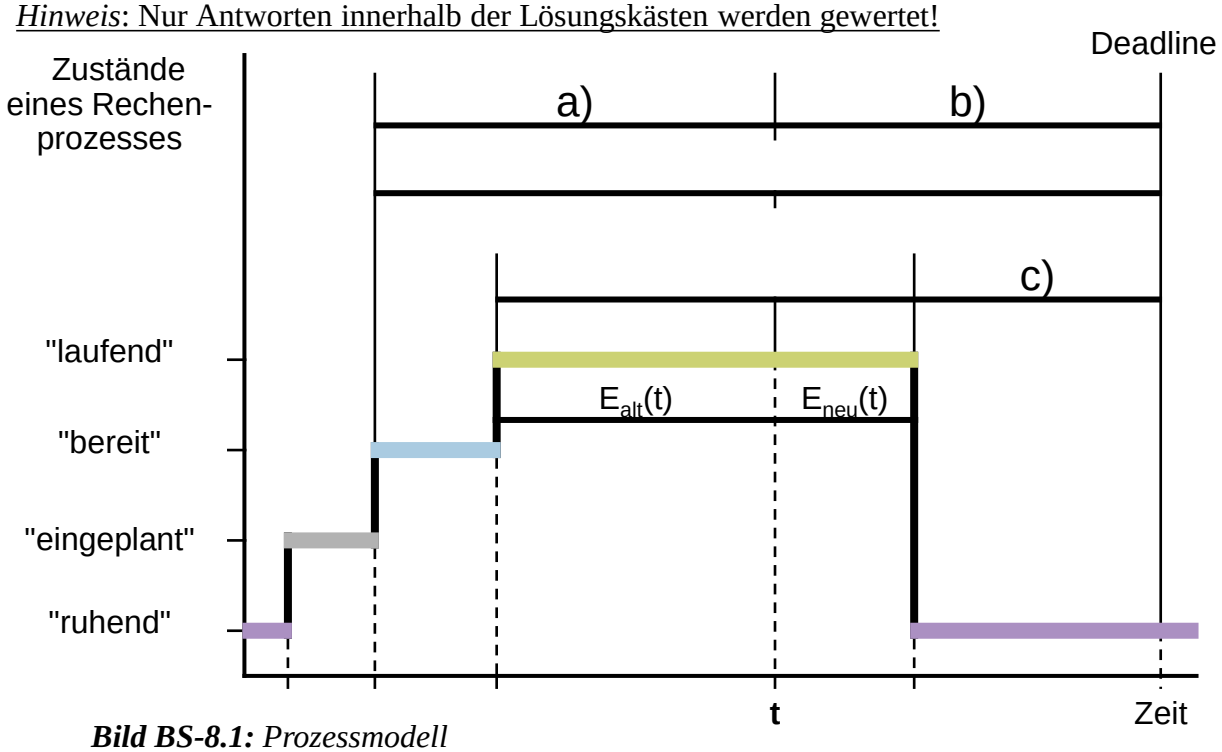


Bild BS-8.1: *Prozessmodell*

a) Ausführungszeit ( ), Antwortzeit (X), maximale Antwortzeitdauer ( )

b) Spielraum ( ), Restantwortzeit (X), Ausführungszeitdauer ( )

c) Spielraum (X), Antwortzeit ( ), verbleibende Ausführungszeit ( )

Für welchen Scheduling-Algorithmus spielt Parameter c) die zentrale Rolle bei der Priorisierung der Tasks?

EDF ( ), LIFO ( ), RR ( ) , LL (X)

9. Asynchrones Scheduling, präemptiv, RR

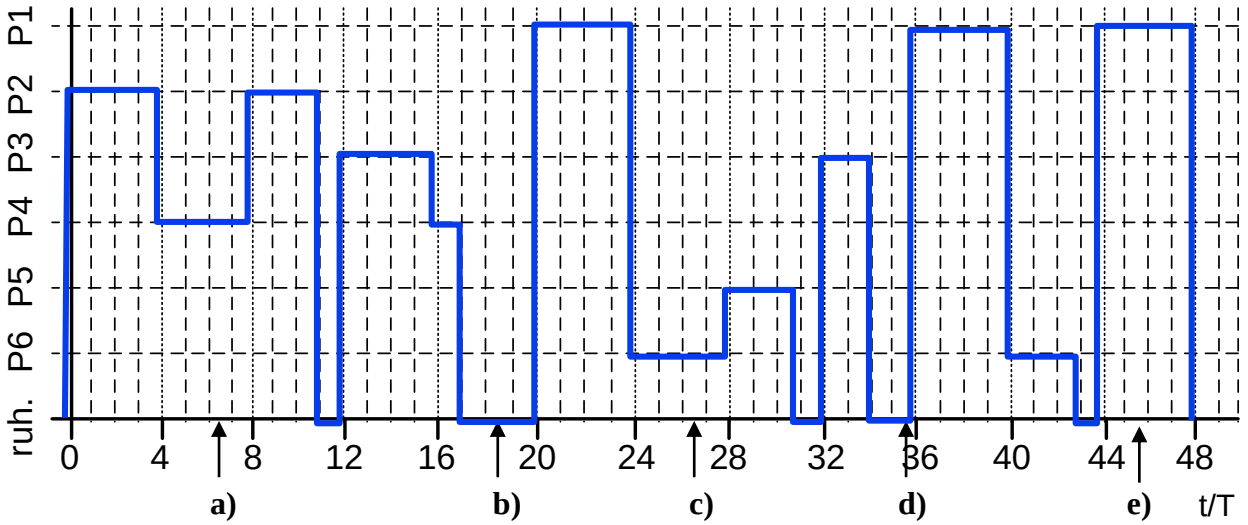
Gegeben seien die folgenden sechs Prozesse (Bild BS-9.1), welche jeweils ab dem Zeitpunkt „Start“ eingeplant werden sollen. Zur Abarbeitung eines Tasks wird die Rechenzeitspanne „Dauer“ benötigt. Periodische Tasks werden mit der Häufigkeit „Frequenz“ erneut aufgerufen. Erstellen Sie im untenstehenden Diagramm das präemptive Scheduling nach dem Schema „Round-Robin“ für den Zeitraum 0 bis 50s für einen Einkernprozessor. Treffen innerhalb eines Zeitschlitzes mehrere Tasks ein, beachten Sie die „Prioritäten“. Ein Zeitschlitz hat eine Größe von vier Sekunden. Tragen Sie anschließend die Zeitpunkte, an denen die Tasks fertig abgearbeitet werden, mit einer Genauigkeit von 1s in den untenstehenden Lösungskasten ein.

Hinweis: Nur Antworten innerhalb des Lösungskastens werden gewertet!

	Priorität	Start	Dauer	Frequenz
P1	1 (hoch)	10 s	4 s	14 s
P2	2	0 s	7 s	einmalig
P3	3	6 s	6 s	einmalig

	Priorität	Start	Dauer	Frequenz
P4	4	3 s	5 s	einmalig
P5	5	13 s	3 s	einmalig
P6	6 (niedrig)	9 s	7 s	einmalig

Bild BS-9.1: *Taskspezifikation I*



a) P1 ( ), P2 ( ), P3 ( ), P4 (x), P5 ( ), P6 ( ), ruhend ( )

b) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend (x)

c) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 (x), ruhend ( )

d) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend (x)

e) P1 (x), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend ( )

10. Asynchrones Scheduling, präemptiv, Interrupts

Die drei periodischen Prozesse P1, P2 und P3 (Bild BS-10.1) sollen mit dem Verfahren der asynchronen Programmierung präemptiv auf einem Einkernprozessor eingeplant werden. Der Prozess P1 besitzt die höchste, der Prozess P3 die niedrigste Priorität. Die Ausführung wird durch einen Interrupt unterbrochen.

Tragen Sie in das unten angegebene leere Diagramm den Verlauf der Abarbeitung von Prozessen und Interrupts ein. Kreuzen Sie danach an den durch einen Pfeil markierten Stellen den aktiven Prozess an.

Hinweis: Nur Kreuze innerhalb des Lösungskastens werden gewertet!

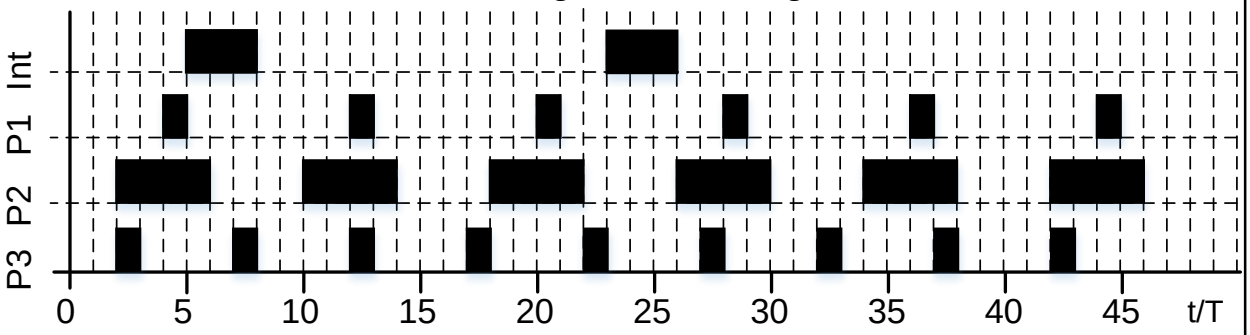
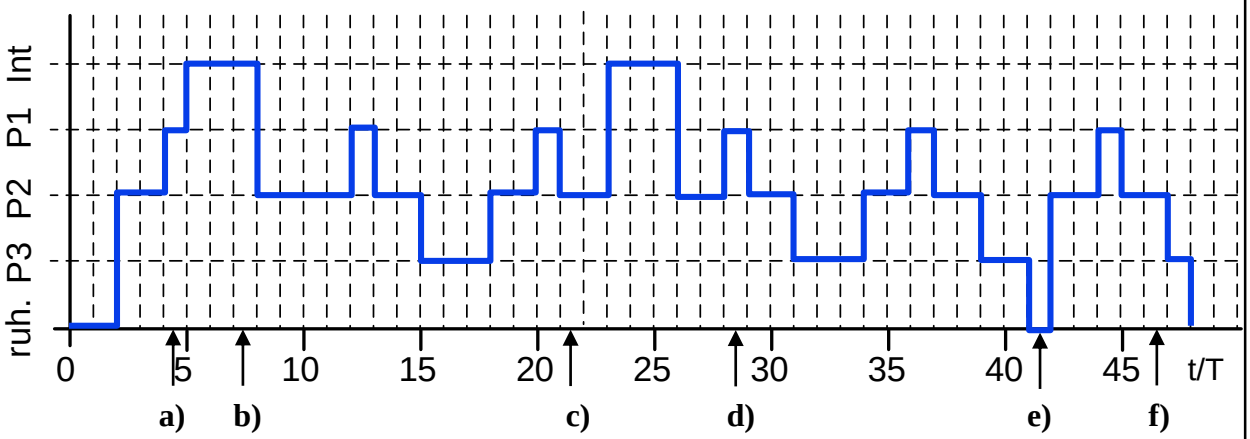


Bild BS-10.1: *Prozessverlauf*



a) Interrupt ( ), P1 (x), P2 ( ), P3 ( ), ruhend ( )

b) Interrupt (x), P1 ( ), P2 ( ), P3 ( ), ruhend ( )

c) Interrupt ( ), P1 ( ), P2 (x), P3 ( ), ruhend ( )

d) Interrupt ( ), P1 (x), P2 ( ), P3 ( ), ruhend ( )

e) Interrupt ( ), P1 ( ), P2 ( ), P3 ( ), ruhend (x)

f) Interrupt ( ), P1 ( ), P2 (x), P3 ( ), ruhend ( )

11. Semaphoren

Gegeben seien die folgenden vier Tasks T1 bis T4 mit absteigender Priorität sowie die dazugehörigen Semaphoren S1 bis S4 (Bild BS-11.1). Die Startwerte der Semaphoren entnehmen Sie der Antworttabelle. Tragen Sie in der ersten Spalte der Antworttabelle den aktuell laufenden Task ein sowie im Rest der Zeile die Werte der Semaphoren nach Ausführung des jeweiligen Tasks.

T1	T2	T3	T4
P(S1)	P(S2)	P(S3)	P(S4)
	P(S2)	P(S3)	P(S4)
…	…	…	…
		V(S2)	V(S1)
V(S3)	V(S4)	V(S2)	V(S3)

Bild BS-11.1: *Semaphorenzuweisung*

Task	S1	S2	S3	S4
-	1	0	1	1
T1	**0**	**0**	**2**	**1**
T3	**0**	**2**	**0**	**1**
T2	**0**	**0**	**0**	**2**
T4	**1**	**0**	**1**	**0**
T1	**0**	**0**	**2**	**0**
T3	**0**	**2**	**0**	**0**
T2	**0**	**0**	**0**	**1**

Beantworten Sie außerdem, ob im betrachteten Schedule folgende Phänomene auftreten:

a) Partielle Verklemmung: ( ) ja (**x**) nein

b) Globale Verklemmung: (**x**) ja ( ) nein

c) Endlosschleife: ( ) ja (**x**) nein

Vorname Nachname Matrikelnummer

12. Echtzeitbetriebssysteme

Im Folgenden (Bild BS-12.1) ist ein unvollständiges „erweitertes Taskzustandsdiagramm von RTOS-UH“ gegeben.

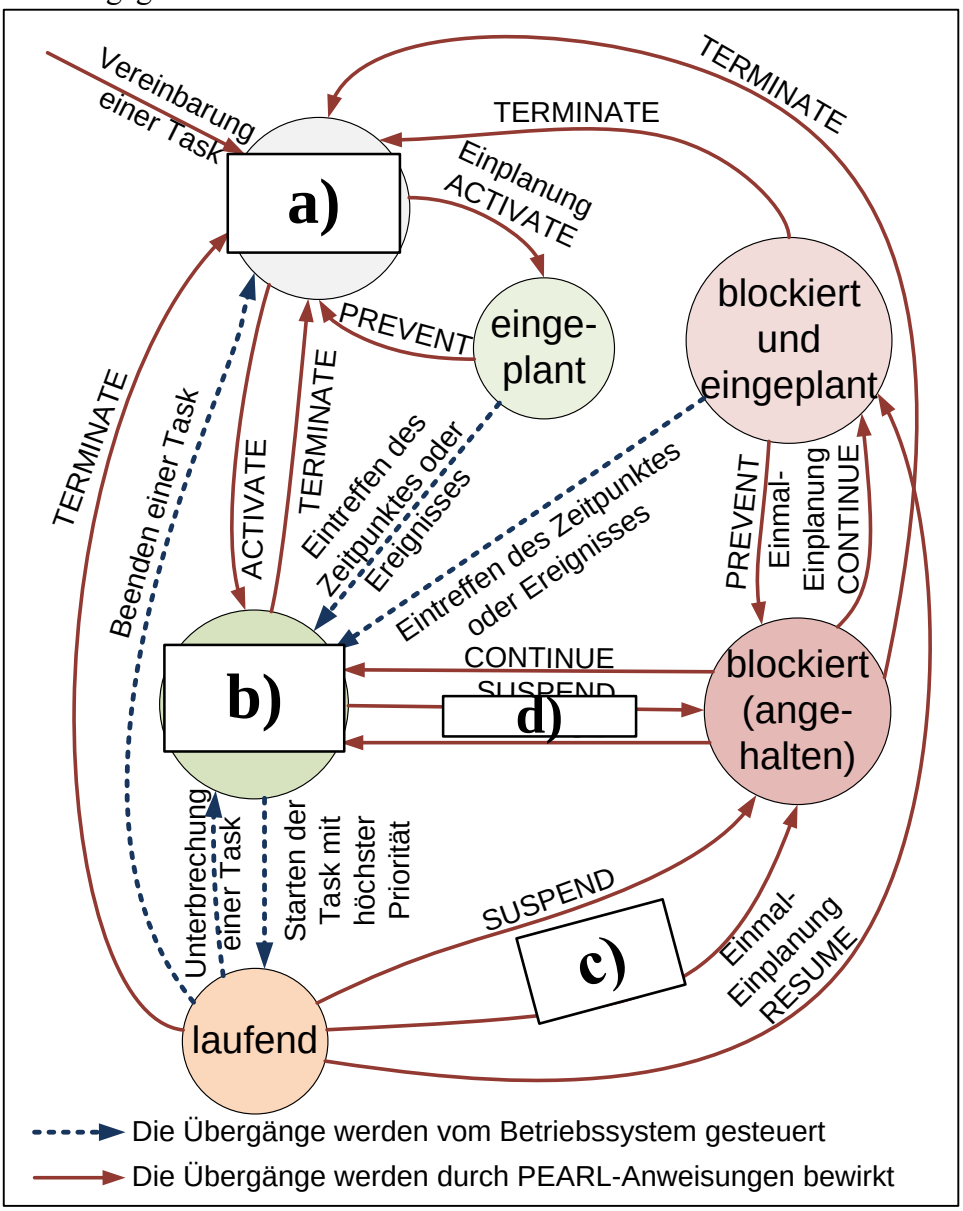


Bild BS-12.1: *Erweitertes Taskzustandsdiagramm von RTOS-UH*

Bezeichnen Sie die im Diagramm mit Buchstaben markierten Lücken:

a) **Ruhend/bekannt**

b) **Bereit/lauffähig**

c) **Erfolgloses REQUEST**

d) **RELEASE**

13. IEC 61131-3: Funktionsbausteinsprache (FBS)

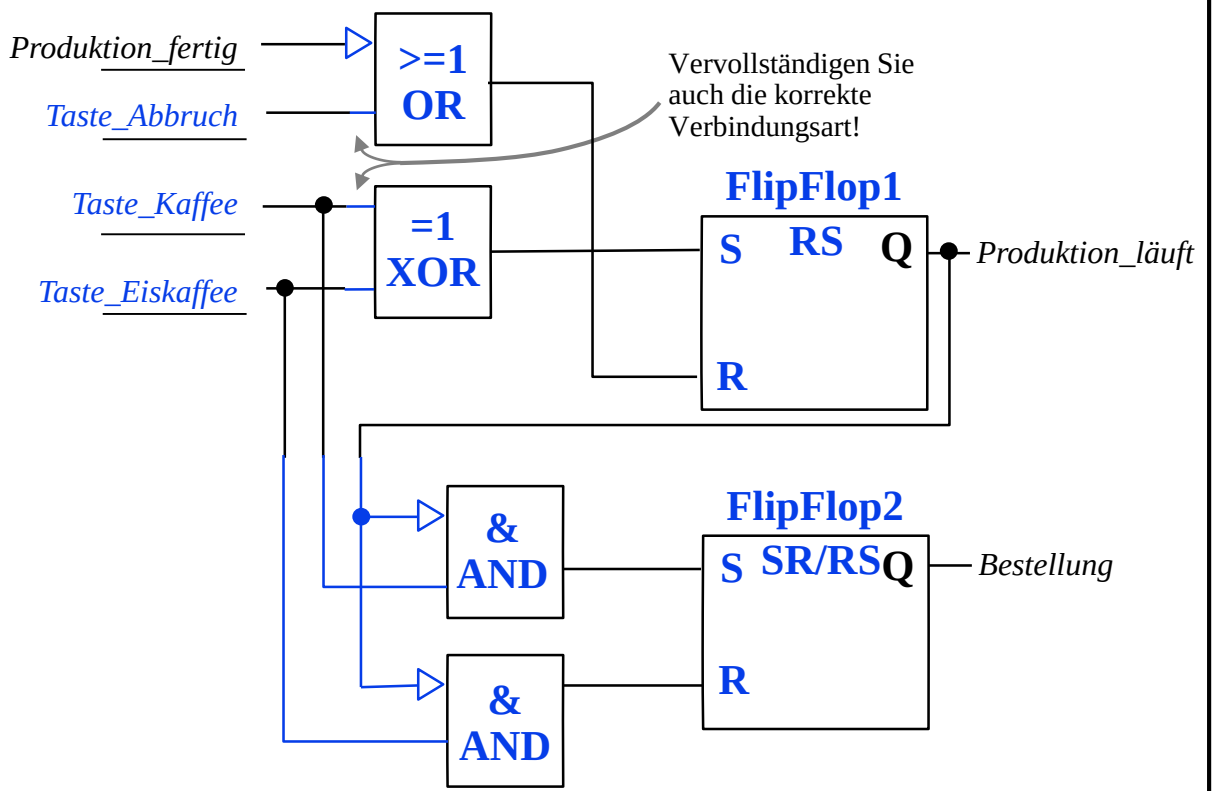
Ein Kaffeeautomat soll über drei Tasten (*Taste_Abbruch*, *Taste_Kaffee*, *Taste_Eiskaffee*) gesteuert werden. Der Automat verwendet drei Schnittstellen zum Rest der Maschinensteuerung (*Produktion_läuft*, *Produktion_fertig* und *Bestellung)*.

Wählt der Kunde über einen Druck der Tasten *Taste_Kaffee* oder *Taste_Eiskaffee* ein Produkt, soll das Signal *Produktion_läuft* von 0 auf 1 wechseln und *Bestellung* für einen Kaffee auf 1 und für einen Eiskaffee auf 0 gesetzt werden. Im selben Moment wechselt der Sensor *Produktion_fertig* auf 0. Der Zustand von *Bestellung* darf bis zum Ende der Herstellungsdauer nicht verändert werden. Ein Drücken von beiden Bestelltasten gleichzeitig muss abgefangen werden. Ebenso der Beginn einer neuen Bestellung, so lange das Signal *Produktion_läuft* sich noch im Zustand 1 befindet.

Ist die Produktion abgeschlossen, wechselt der Sensor *Produktion_fertig* auf 1. Als Reaktion soll das Signal *Produktion_läuft* auf 0 gesetzt werden.

Wird die Taste *Taste_Abbruch* betätigt, soll das Signal *Produktion_läuft* sofort auf 0 gesetzt werden. So lange *Taste_Abbruch* gedrückt wird, darf keine Reaktion auf die Bestelltasten erfolgen.

Hinweis: Signalverzögerungen im System sind zu vernachlässigen



Vorname Nachname _____ Matrikelnummer _____

Aufgabe MSE: Modellierung und Softwareentwicklung

Aufgabe MSE:
48 Punkte

14. Automaten

a) Gegeben sei der nachfolgende Automat. Der Automat befindet sich aktuell im Zustand *s2*.
Leiten Sie eine Übersicht der Übergänge ab, indem sie die untenstehende Tabelle vervollständigen (Zustand und Ausgabe)!

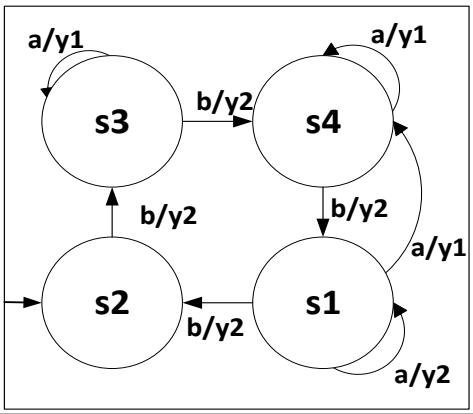


Bild MSE-14.1: *Automat*

T	s1	s2	s3	s4
a	**s1,y2** **s4,y1**	**-**	**s3,y1**	**s4,y1**
b	**s2,y2**	**s3,y2**	**s4,y2**	**s1,y2**

Welche Eingabe müssen Sie tätigen damit Sie die Ausgabesequenz

y2 y1 y2 y1 y1 y2 erhalten?

b a b a a b

Handelt es sich bei dem gegebenen Automaten um einen Moore- oder Mealy-Automaten?

Moore-Automat ( ) Mealy-Automat (**x**)

Handelt es sich um einen deterministischen oder nicht-deterministischen Automaten?

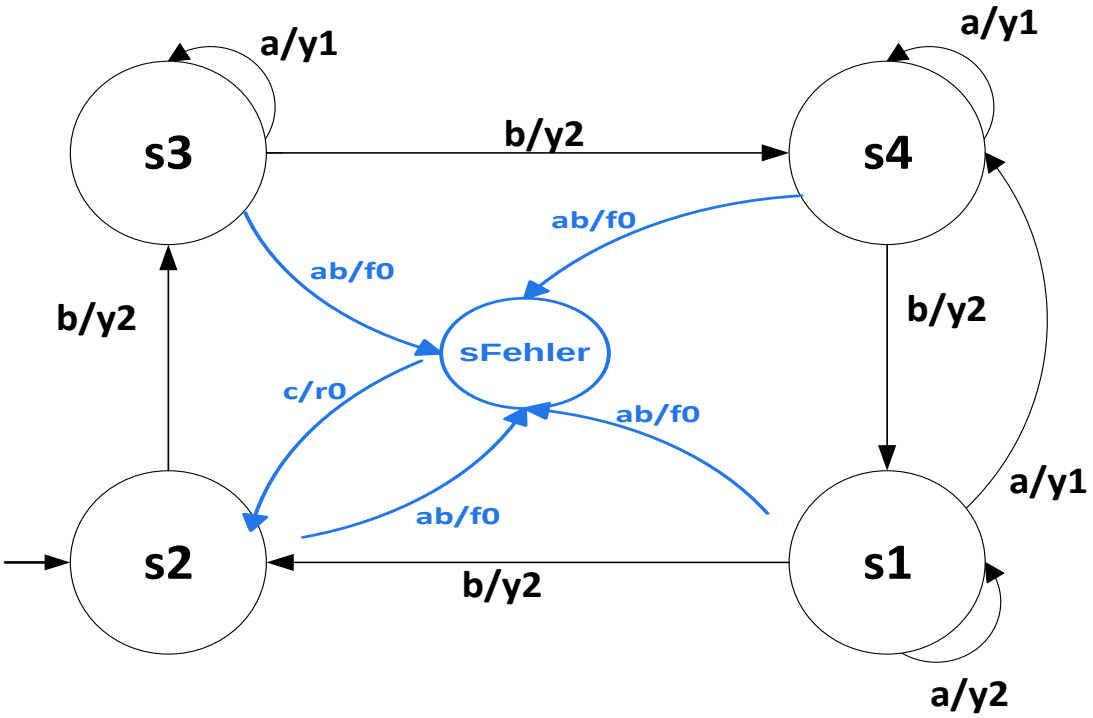
Begründen Sie Ihre Antwort! (Begründung ausschlaggebend)

Nicht Deterministisch, da S1 uneindeutige Übertragungsfunktion

b) Der Automat aus a) repräsentiert die Betriebsarten einer Maschine. Im nachfolgenden soll der Automat erweitert werden:

- Der Anfangszustand *s2* bleibt bestehen.
- Durch die Eingabe von a und b gleichzeitig, also *ab*, soll die Maschine aus allen Zuständen in den Zustand *sFehler* übergehen. In diesen Fällen wird *f0* ausgegeben.
- Das Verlassen des Zustands *sFehler* ist nur durch den Übergang zum Anfangszustand *s2* möglich. Hierfür ist eine Eingabe *c* notwendig. Ausgegeben wird dabei *r0*.

Vervollständigen Sie den Automaten gemäß der obigen Beschreibung!



15. Zustandsdiagramm

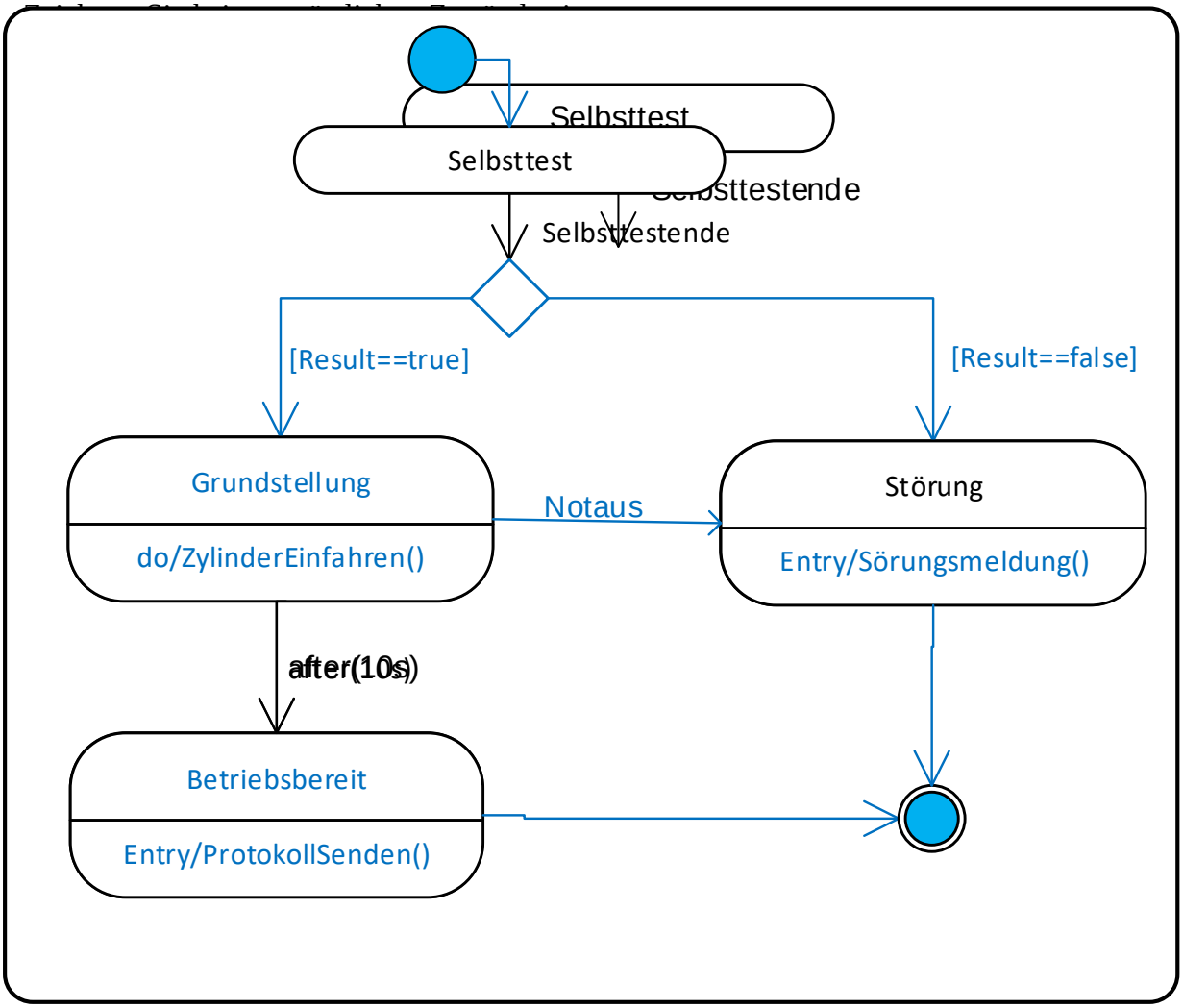
Der Hochlauf einer Maschine soll als Zustandsdiagramm modelliert werden. Dafür wird das System als Erstes in den Zustand *Selbsttest* versetzt. Nachdem das Selbsttestende erreicht wurde, wird überprüft, ob das Ergebnis (Variable *Result*) des Selbsttests wahr oder falsch war.

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 den 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.

Wird im Zustand *Grundstellung* der Trigger *Notaus* ausgelöst, soll das System in den Zustand *Störung* übergehen.

Vervollständigen Sie das Zustandsdiagramm. Fügen Sie Start- und Endzustand ein.



16. SA/RT: Flussdiagramm

Für die folgenden Teilaufgaben ist eine Maschine zur Abfüllung von Kaffee und Eis in Becher gegeben, die aus drei Prozessschritten besteht.

Im Prozess *Becher platzieren* (Prozess 1) wird ein *leerer Becher* aus einem *Becherstapel* in der Maschine platziert. Im Prozess *Kaffee einfüllen* (Prozess 2) wird der *leere Becher* mit *Kaffee* befüllt. Mittels einer CSPEC kann kontrolliert werden, ob laut *Bestellung* im Prozess *Kaffee einfüllen* ein *einfacher Kaffee* oder ein *doppelter Kaffee* eingefüllt werden soll. Im Prozess *Eis einfüllen* (Prozess 3) wird dem *Kaffeebecher Eis* hinzugefügt und der *Eiskaffee* abschließend an den nächsten Prozessschritt weitergegeben, der hier nicht weiter betrachtet wird.

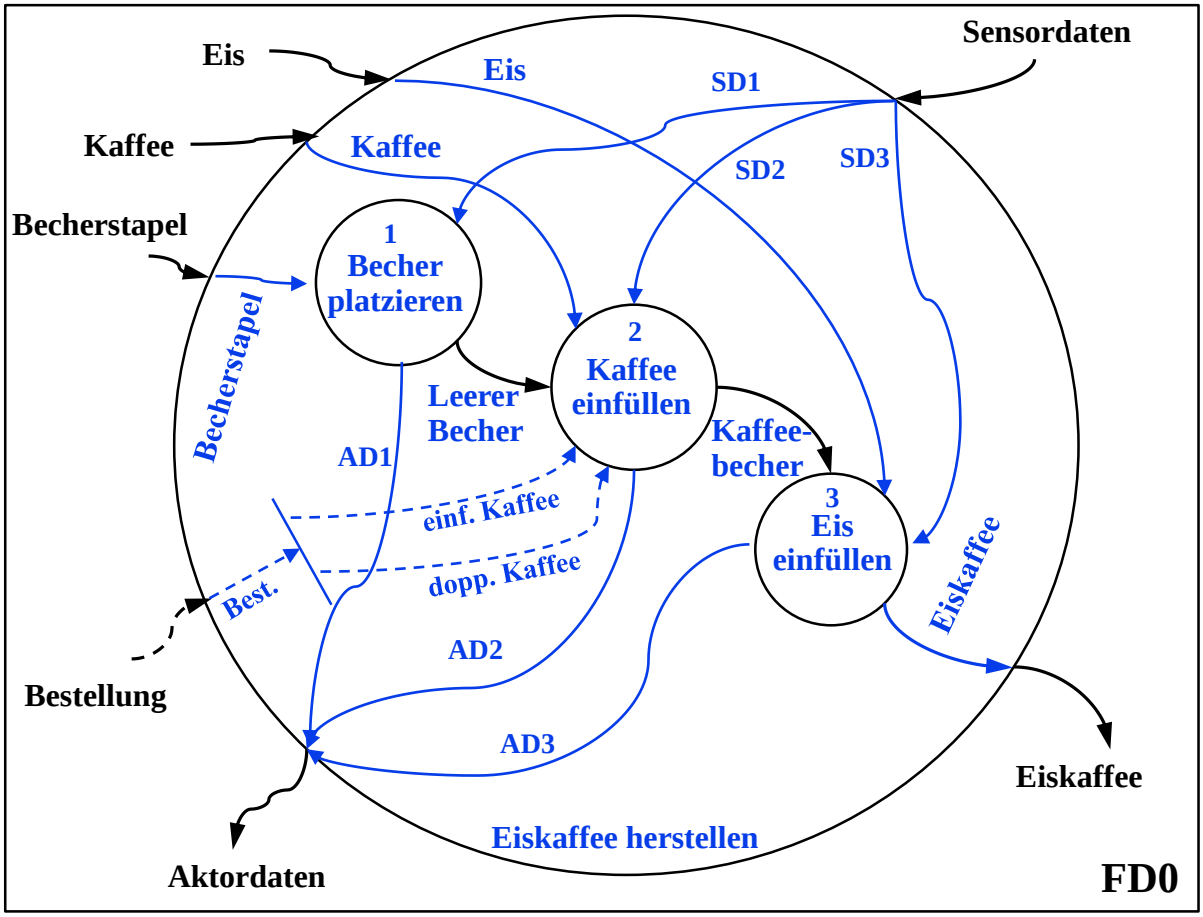
Modellieren Sie den Prozess *Eiskaffee herstellen* (Prozess 0) mittels Strukturierter Analyse / Real-Time (SA/RT) in einem *Flussdiagramm*. Identifizieren Sie hierzu alle Subprozesse, Daten-, und Steuerflüsse und tragen Sie diese mit Bezeichnung ein. Beachten Sie, dass Sensor- und Aktordaten eines Prozesses mit SD*Prozessnummer* und AD*Prozessnummer* (z.B. SD1 oder AD3) zusammengefasst werden. Beachten Sie außerdem folgende Informationen:

Flüsse Material:
Becherstapel, Kaffee, Eis, leerer Becher, Kaffeebecher, Eiskaffee

Flüsse Sensordaten:
je nach Prozess: SD*Prozessnummer*, also z.B. SD2 bei „Kaffee einfüllen“.

Flüsse Aktordaten:
Wie Sensordaten nur AD*Prozessnummer*

Steuerflüsse:
Bestellung, *einfacher Kaffee*, *doppelter Kaffee*

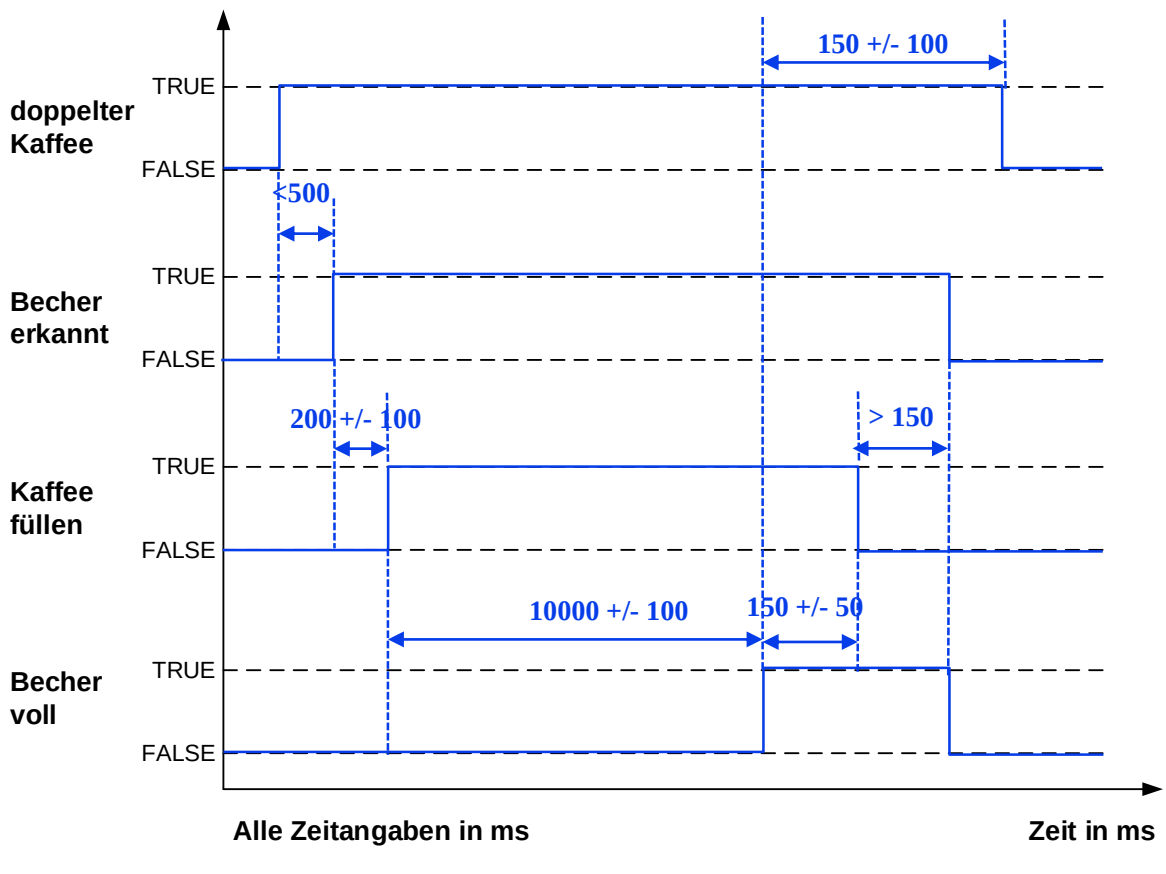


17. Antwortzeitspezifikation: Timing-Diagramm

Zur Verfeinerung der Eiskaffee-Herstellung soll eine Antwortzeitspezifikation in Form eines Timing-Diagramms erstellt werden, um sicherzustellen, dass der Prozess korrekt durchgeführt wird.
Die Werte der Sensoren und Aktoren können jeweils *TRUE* (z. B. Becher erkannt) oder *FALSE* (z. B. Becher nicht erkannt) sein.

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

- Sobald ein großer Kaffee bestellt wird, liefert das Signal *doppelter Kaffee* den Wert TRUE. Um das Getränk zu produzieren, muss nach spätestens 500 ms ein Becher eingestellt werden und somit der Sensor *Becher erkannt* auf TRUE wechseln.
- Nachdem ein Becher erkannt wurde, aktiviert sich nach 200 +/- 100 ms die Kaffeepumpe durch Wechsel des Signals *Kaffee füllen* auf TRUE. Der Füllstandssensor *Becher voll* soll erwartungsgemäß 10 s +/- 0,1 s nach Start des Füllvorgangs einen mittlerweile gefüllten Becher erkennen und auf TRUE wechseln. Die Pumpe wird daraufhin nach 150 +/- 50 ms wieder deaktiviert.
- Nach Abschalten der Pumpe darf der Becher frühestens nach 150 ms entnommen werden. Entsprechend werden ab diesem Moment weder ein Becher noch ein voller Füllstand erkannt.
- Ab dem Zeitpunkt, in dem der volle Becher erkannt wird, wechselt das Signal *doppelter Kaffee* nach 150 +/- 100 ms wieder auf FALSE.



Vorname Nachname Matrikelnummer

Aufgabe C: C-Programmierung

Aufgabe 18: 13 Punkte

18. Datentypen und Boolsche Algebra

a) Datentypen

Definieren Sie die Datentypen der folgenden Variablen so, dass so wenig Speicher wie möglich benötigt wird. Die Variablen sollen zur Beschreibung der Artikels in einem Getränkeautomat verwendet werden.

`Variable preis` Preis eines Artikels auf zwei Nachkommastellen genau

`Variable gesamtUmsatz` Umsatz seit der letzten Leerung gerundet auf ganze Euro, Die Leerung erfolgt monatlich, Im Schnitt werden monatlich 300 Artikel zu einem Durchschnittspreis von 2€ verkauft

`Variable verbArtikel` Verbleibende Artikel in einem Fach (ein Fach enthält höchstens 20 Artikel)

	`char`	`int`	`double`	`float`
`preis`	( )	( )	( )	(x)
`gesamtUmsatz`	( )	(x)	( )	( )
`verbArtikel`	(x)	( )	( )	( )

Es soll der Wert der in einem Fach verbleibenden Artikel berechnet werden, also (`preis * verbArtikel`). Welchen Datentyp hat das Ergebnis dieser Berechnung?

( ) `char` ( ) `int` ( ) `double` (x) `float`

b) Ein- und Ausgabe

Füllen Sie die folgenden Ein- und Ausgabebefehle sowie die Formatstrings aus. Die verwendeten Variablen sind deklariert und initialisiert.

i) Ausgabe einer Gleitkommazahl mit 3 Nachkommastellen

```
printf (" %.3f ", fZahl);
```

ii) Einlesen eines Strings von maximal 50 Zeichen von der Standardeingabe.

```
fgets(puffer, 50 , stdin );
```

c) Boolsche Algebra
Bestimmen Sie das Ergebnis der nachfolgend angegebenen Ausdrücke im Dezimalsystem. Gegeben sind folgende Variablen:

```
int a = 3;
int b = 17;
int c = 1;
int* d = &a;
float f = 1.16;
```

Nach jedem der nachfolgenden Ausdrücke werden die Variablen auf die oben genannten Werte zurückgesetzt.

x.1	`((c + a++) << !(b < a))`	**08**
x.2	`((int)f | ++*d) && ('c' > 0))`	**01**

d) Boolsche Ausdrücke
Schreiben Sie jeweils einen boolschen Ausdruck, der die Aussagen der gegebenen textuellen Beschreibungen wiedergibt.
Hierfür sind die Variablen `int iZahl1`, `int iZahl2` und `int b` bereits definiert.

i) Die Zahl iZahl1 ist größer oder gleich 12 und die Zahl b ist 0.
```
iZahl1 >= 12 && b == 0
```

ii) Die Zahl `iZahl1` ist ungerade oder die `iZahl2` ist gerade und größer 20.
```
(iZahl1 % 2 != 0) || ((iZahl2 % 2 == 0) && iZahl2 > 20)
```

Vorname Nachname Matrikelnummer

19. Kontrollstrukturen

Aufgabe 19: 10 Punkte

Sie sollen ein Programm entwickeln, dass die Verkaufszahlen eines Getränkeautomaten analysiert. Hierzu liegen die Verkaufszahlen der letzten drei Tage der vier Artikel als zwei-dimensionales Array *verkaeufe*, sowie die Preise der Artikel als Array *preise* vor.

Das Programm soll den

- Gesamtumsatz mit 2 Nachkommastellen,
- den umsatzstärksten Tag und den
- durchschnittlichen Umsatz pro Artikel mit 2 Nachkommastellen

ausgeben.

Kreuzen Sie auf der folgenden Seite im Bild C-19.2 die korrekten Codefragmente für die in Bild C-19.1 angegebenen Lücken an.

```
        (1)
_____________________
   (2)
_______FAECHER 4

int main()
{
  int verkaeufe [][FAECHER] = {{3, 2, 1, 2},         //Tag1
                               {1, 2, 6, 2},         //Tag2
                               {7, 2, 7, 2}};        //Tag3
  float preise[] = {1.5, 0.5, 3.7, 1}; // Preise Artikel 1-4
  float umsaetzeArtikel[] = {0, 0, 0, 0};
  int umsatzTageweise[] = {0, 0, 0};
  int i = 0, j = 0, iStaerksterTag = 0;
  float fGesamtumsatz = 0;
  for (_____(3)_____)
  {
  ____(4)____ (j = 0; j < FAECHER; j++)
    {
      umsatzTageweise[i] += verkaeufe[i][j] * preise[j];
      umsaetzeArtikel[j] += verkaeufe[i][j] * preise[j];
    }
    if (umsatzTageweise[iStaerksterTag] _____(5)_____  )
    {
      iStaerksterTag = i;
    }
    fGesamtumsatz _____(6)_____
  }
  printf("Der Gesamtumsatz war %.2f. Tag %i war der umsatzstaerkste Tag.\n",
         fGesamtumsatz,_____(7)_____ );

  printf("Durchschnittlicher Umsatz war:___(8)___.", _____(9)_____ );
  ____(10)_____
}
```

Bild C-19.1: *Sourcecode zur Auswertung der Verkaufszahlen eines Getränkeautomaten.*

Kreuzen Sie in dem folgenden Bild C-19.2 die korrekten Codefragmente für die in Bild C-19.1 angegebenen Lücken an (nur Einfachantwort möglich).

i) Lücke ① in Bild C-19.1

```
( ) #include stdio.h        ( ) #include <stdio>     (x) #include <stdio.h>
( ) #define <stdlib.h>      ( ) #include stdlib      ( ) #define <io.h>
```

ii) Lücke ② in Bild C-19.1

```
(x) #define                 ( ) #typedef            ( ) #include
( ) #define <stdlib.h>      ( ) #include stdlib     ( ) #const
```

iii) Lücke ③ in Bild C-19.1

```
( ) i = 0; i <= 3; i++      ( ) i = 1; i <= 3; i++  ( ) i = 0; i < 4; i + 1
(x) i = 0; i < 3; i++       ( ) i = 0; i < 3; ++i   ( ) i = 0; i < 3; i + 1
```

iv) Lücke ④ in Bild C-19.1

```
( ) while                   ( ) do                  ( ) if
(x) for                     ( ) switch              ( ) case
```

v) Lücke ⑤ in Bild C-19.1

```
( ) > fGesamtumsatz                 ( ) <= umsatzTageweise[iStaerksterTag - 1]
( ) <= umsatzTageweise[i - 1]       (x) < umsatzTageweise[i]
( ) > iStaerksterTag                ( ) > umsatzTageweise[i++]
```

vi) Lücke ⑥ in Bild C-19.1

```
( ) += umsaetzeArtikel[i];      ( ) *= umsatzTageweise[i]
( ) -= umsatzTageweise[i]       ( ) = umsatzTageweise[i];
(x) += umsatzTageweise[i];      ( ) += umsatzTageweise[i] + umsaetzeArtikel[j];
```

vii) Lücke ⑦ in Bild C-19.1

```
( ) iStaerksterTag          ( ) ++iStaerksterTag    ( ) i
( ) i + 1                   (x) iStaerksterTag + 1  ( ) ++i
```

viii) Lücke ⑧ in Bild C-19.1

```
(x) %.3f                    ( ) .3%f                ( ) .3%i
( ) %d                      ( ) %f                  ( ) .%d2
```

ix) Lücke ⑨ in Bild C-19.1

```
( ) fGesamtumsatz                   ( ) FAECHER / fGesamtumsatz
( ) fGesamtumsatz *iStaerksterTag   (x) fGesamtumsatz / FAECHER
( ) fGesamtumsatz * Preise          ( ) fGesamtumsatz * FAECHER
```

x) Lücke ⑩ in Bild C-19.1

```
( ) return                  ( ) break 0;            ( ) continue 1;
( ) continue;               ( ) break;              (x) return 0;
```

Bild C-19.2: *Codefragmente für Bild C-19.1.*

20. Objektorientierte Programmierung

Aufgabe 20: 25 Punkte

a) Grundlagen & Konzepte

Im Folgenden werden grundlegende Konzepte der objektorientierten Programmierung besprochen. Bitte wählen Sie aus den Antwortalternativen die eine korrekte Alternative aus (nur Einfachnennung möglich).

i) Wie bezeichnet man eine konkrete Ausprägung einer Klasse?

( ) Objekt ( ) Exemplar (x) Instanz
( ) Klasse ( ) Assoziation ( ) Typ

ii) In welcher Beziehung steht die Klasse `Bier` zur Klasse `Getränk`?

( ) Aggregation (x) Vererbung ( ) Instanz
( ) In keiner ( ) Assoziation ( ) Schnittstelle

iii) Was versteht man unter Polymorphie?

( ) Objekte der Unterklasse besitzen die selben Attribute wie Objekte der Oberklasse.
( ) Der Datentyp einer Variablen wird automatisch umgewandelt.
( ) Zwischen Oberklasse und Unterklasse dürfen keine Beziehungen bestehen.
( ) Der Datentyp einer Variablen wechselt.
(x) Objekt einer Unterklasse kann auch als Objekt seiner Oberklasse auftreten.
( ) Objekte der Unterklasse können auf `private` Methoden der Oberklasse zugreifen.

iv) Wer darf auf Attribute oder Methoden mit dem Zugriff `protected` zugreifen?

( ) Jede andere Klasse
( ) Nur assoziierte Klassen
( ) Alle anderen Methoden
(x) Abgeleitete Klassen
( ) Niemand
( ) Oberklassen

v) Wann wird ein Konstruktor aufgerufen?

( ) Bei der Zerstörung der Instanz
( ) Abhängig vom System auf dem das Programm ausgeführt wird
(x) Bei der Instanziierung eines Objekts
( ) Je nach Implementierung
( ) Bei erstmaliger Verwendung der Instanz
( ) Bei jeder Verwendung der Instanz

vi) Wie greift man auf ein Attribut einer Klasse zu?

( ) .. ( ) & ( ) :
( ) * (x) -> ( ) -->

b) Modellierung mit UML

Sie sollen eine Software für einen Getränkeautomaten entwickeln. Hierfür soll zunächst ein UML-Klassendiagramm erstellt werden. Beim Treffen mit Ihrem Auftraggeber erfragen Sie die notwendigen Informationen in einem Interview. Ein Ausschnitt der Interviewergebnisse, den Sie in ein Klassendiagramm überführen sollen, ist in Bild C-20.1 gegeben.

- Ein Getränkeautomat wird von einem Betreiber betrieben.
- Der Getränkeautomat enthält eine Reihe von Fächern.
- Jedes Fach besitzt eine Nummer und eine Variable, die den aktuellen Füllstand beschreibt. Zudem existiert eine Methode, die den Umsatz seit dem letzten Auffüllen berechnet und als Gleitkommazahl zurückgibt.
- Ein Fach beinhaltet Artikel, wie zum Beispiel ein Getränk oder einen Snack, die einen ganzzahligen Preis besitzen.

Bild C-20.1: *Klassendiagramm des Getränkeautomaten.*

Beachten Sie die folgenden Konventionen: Attribute sollen mit privater Sichtbarkeit und Methoden mit öffentlicher Sichtbarkeit versehen werden.

Füllen Sie die Lücken im folgenden Klassendiagramm (siehe Bild C-20.2) aus. Die auszufüllenden Lücken sind mit Zahlen markiert.

Lücken **zwischen** Klassen erfordern das Einzeichnen von Beziehungen, Lücken **innerhalb** von Klassen erfordern das Einfüllen von Attributen, Methoden oder Klassennamen.

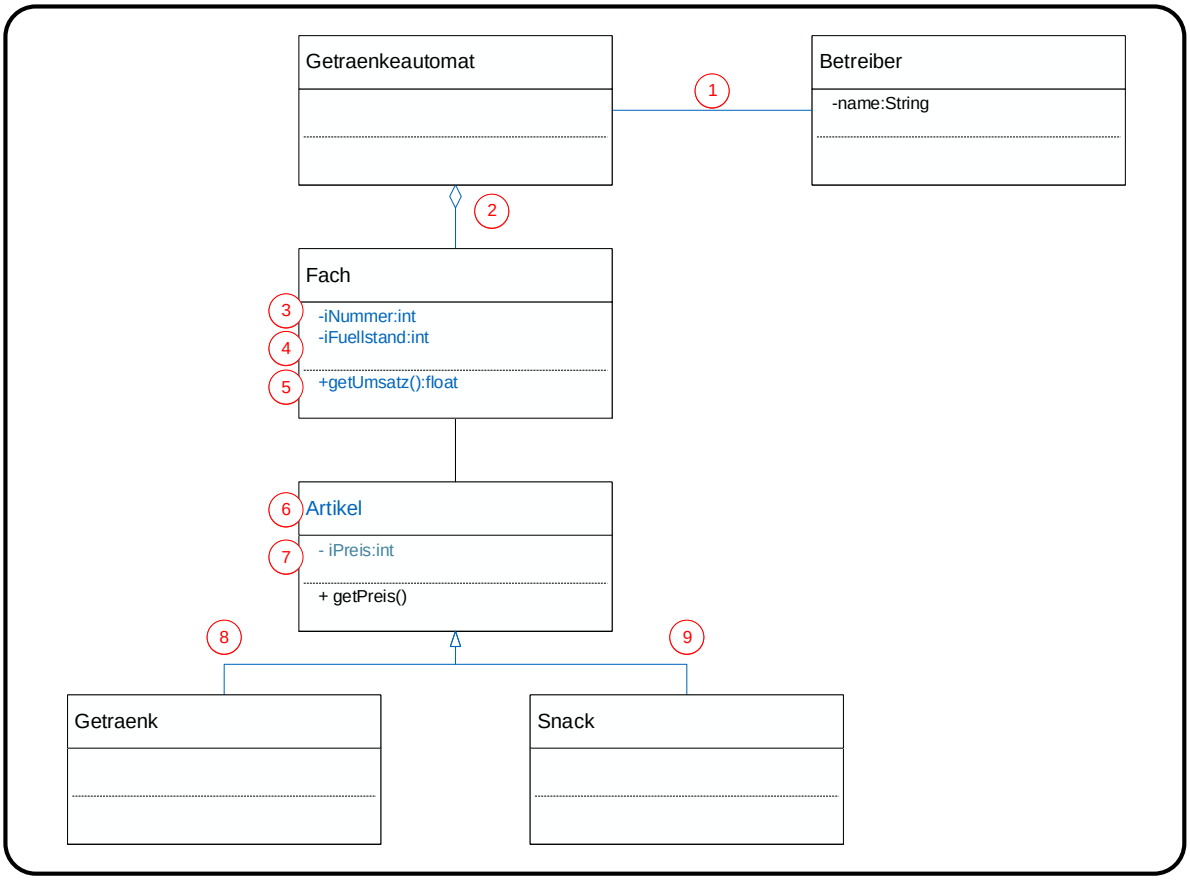


Bild C-20.2: *Klassendiagramm des Getränkeautomaten.*

c) Vom Code zum Klassendiagramm

Sie haben den Programmcode der Software des Getränkeautomaten vom Entwickler erhalten. Um die Struktur der Software zu verstehen, möchten Sie nun ein Klassendiagramm des Codeausschnitts in Bild C-20.3 erstellen.

```
class AlkoholischesGetraenk:public Getraenk
{
  public:
    AlkoholischesGetraenk(float alkoholGehalt);

  private:
    float alkoholGehalt;
};

class Getraenk
{
  public:
    Getraenk();

  private:
    Hersteller* hersteller
    int kalorien;
};

class Getraenkeautomat
{
  private:
    Getraenk getraenke[];
};
```

Bild C-20.3: *Programmcode für den Getränkeautomaten.*

Ein Grundgerüst des Klassendiagramms ist in Bild C-20.4 vorgegeben. Die auszufüllenden Lücken sind mit Zahlen markiert.

Lücken **zwischen** Klassen erfordern das Einzeichnen von Beziehungen, Lücken **innerhalb** von Klassen erfordern das Einfüllen von Attributen, Methoden oder Klassennamen.

Achten Sie beim Ausfüllen auf die Datentypen der Variablen, der Parameter und der Rückgabewerte der Funktionen.

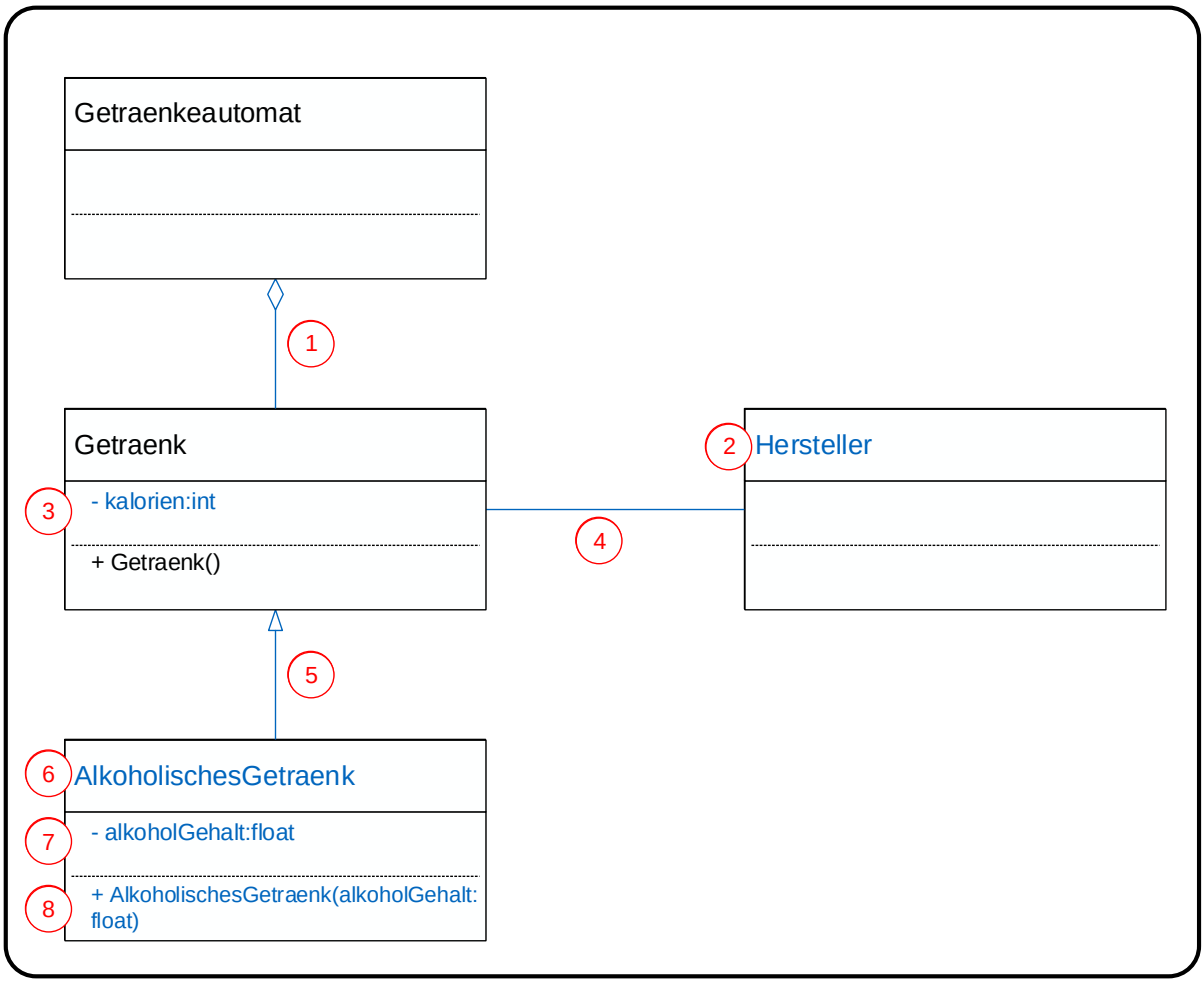


Bild C-20.4: *Klassendiagramm für den Getränkeautomaten.*

21. Anlagen/Zustandsautomat

Aufgabe 21: 25 Punkte

Die Befüllung des zuvor betrachteten Getränkeautomaten soll automatisiert werden. Hierzu wird an der Rückseite ein Aufzugsystem angebracht, welches die benötigten Dosen in die jeweiligen Getränkefächer sortiert. Eine schematische Darstellung des Prozesses ist in Bild C-21.1 gegeben.

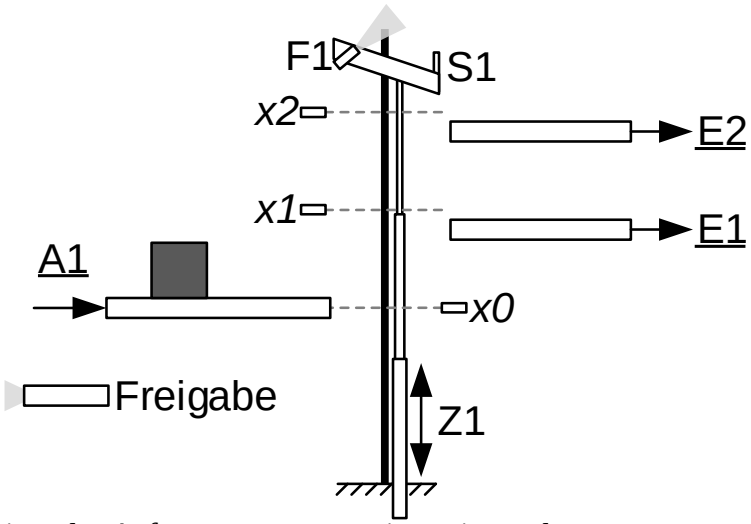


Bild C-21.1: *Skizze des Aufzugsystems zum Einsortieren der Dosen*

Hierzu sind folgende Ein- und Ausgänge vorhanden:

Typ	Name	Beschreibung
AKTOREN	`a.Z1hoch`	Fährt(1) Aufzug nach oben oder stoppt (0)
	`a.Z1runter`	Fährt(1) Aufzug nach unten oder stoppt (0)
	`a.S1`	Öffnet(1) oder schließt (0) Sperre S1
	`a.Freigabe`	Setzt Freigabe (1) an Vorgängerstation
SENSOREN	`s.x0`	Liefert (1) falls sich Aufzug bei Eingangsschacht befindet, sonst (0)
	`s.x1`	Liefert (1) falls sich Aufzug bei Ausgabeschacht E1 befindet, sonst (0)
	`s.x2`	Liefert (1) falls sich Aufzug bei Ausgabeschacht E2 befindet, sonst (0)
	`s.Teil`	Dose im Aufzug vorhanden (1), sonst (0)
	`s.F1`	(0) bei Wasser, (1) bei Zitronenlimonade, Wert nur sinnvoll falls Dose im Aufzug
Variablen	`time`	Aktuelle Laufzeit des Programms in ms

Bild C-21.2: *Sensor- und Aktorwerte des Aufzugs.*

Zunächst soll die Anlage in einen Initialzustand gebracht werden. Hierzu wird der Kran nach unten gefahren, das Freigabesignal zurückgesetzt und die Sperre ausgefahren. Ist dieser Zustand erreicht wird eine Freigabe erteilt und auf eine Dose, welche aus dem Lager über das Band A1 zugeführt wird, gewartet.

Sobald die Dose im Aufzug ist, wird die Freigabe zurückgesetzt und der Aufzug gestartet. Während der Aufzugsfahrt wird mit Sensor F1 der Getränkeinhalt bestimmt. Wasser soll auf Ausgabeband E2 aussortiert werden, Zitronenlimonade auf E1. Die Position des Aufzugs wird über die entsprechenden Anschläge x0, x1 und x2 detektiert und der Aufzug muss am entsprechenden Band gestoppt werden.

Hat der Aufzug seine Position erreicht, soll die Sperre geöffnet werden, damit die Dose auf das Ausgabeband rutscht. Da hier ein Sensor fehlt, muss ein Timer verwendet werden, der die Sperre für zwei Sekunden offen hält. Anschließend wird erneut der Initialzustand hergestellt.

a) Vervollständigen Sie das folgende, in Bild C-21.3 gegebene Zustandsdiagramm entsprechend der oben gegebenen Ablaufbeschreibung unter Berücksichtigung der in Tabelle C-21.2 gegebenen Signale.

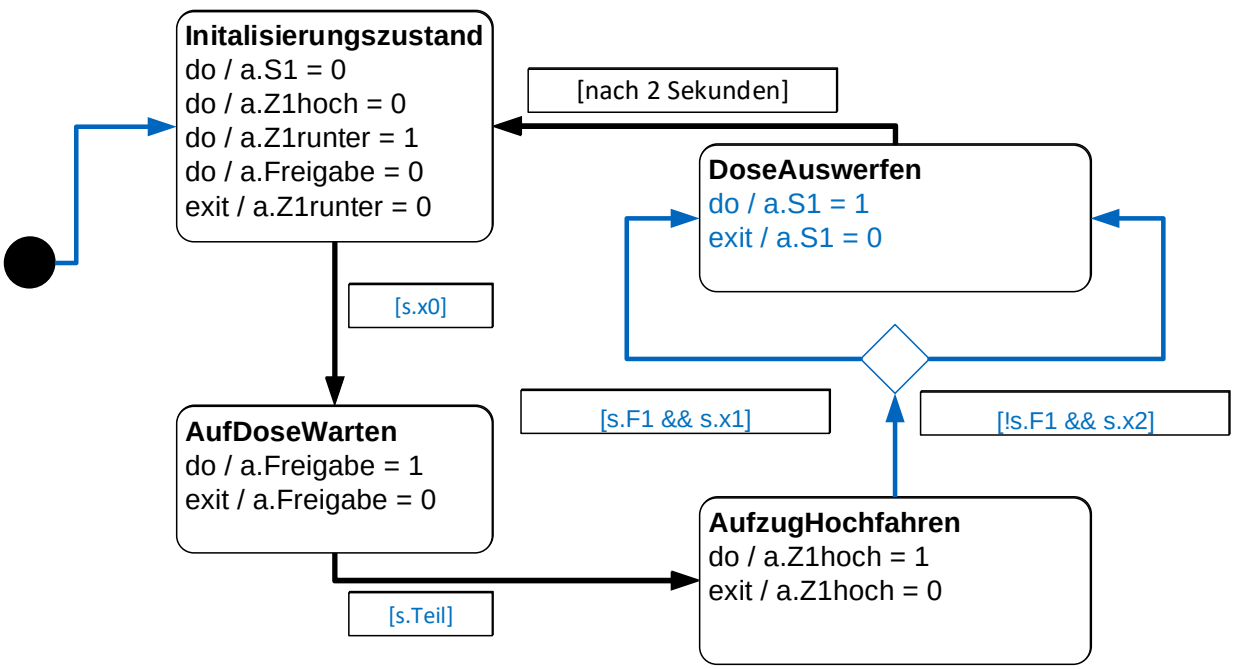


Bild C-21.3: *Zustandsdiagramm des Aufzugsystems.*

b) Vervollständigen Sie nun den vorgegebenen Programmcode (siehe Bild C-21.4 und C-21.5).

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

int schritt = 0;
unsigned int t = 0;        // Timervariable

int main()
{
    switch (schritt){
    case 0:      // Initalisierungszustand
        a.S1 = 0;
        a.Z1hoch = 0;
        a.Z1runter = 1;
        a.Freigabe = 0;
        if (s.x0)              // Waechterbedingung
        {
            a.Z1runter = 0;
            schritt = 1;
        }
        break;
    case 1:      // AufDoseWarten
        a.Freigabe = 1;
        if (s.Teil)
        {
            a.Freigabe = 0;
            schritt = 2;
        }
        break;
     case 2:      // AufzugHochfahren
            a.Z1hoch = 1;
            if (s.F1 && s.x1)
            {
                a.Z1hoch = 0;
                schritt = 3;
            }
            else if (!s.F1 && s.x2)
            {
                a.Z1hoch = 0;
                schritt = 3;
            }
            break;
```

Bild C-21.4: *Programmcode des Aufzugsystems (Teil 1 von 2).*

```
        case 3:      // DoseAuswerfen
                a.S1 = 1;
                if (t == 0)                    // Timer starten
                {
                    t = time;
                }
                if (time - t > 2000)           // Timer pruefen
                {
                    a.S1 = 0;
                    t = 0;
                     schritt = 0;
                }
                break;
            }
            return 0;
        }
```

Bild C-21.5: *Programmcode des Aufzugsystems (Teil 2 von 2).*

Aufgabe 22: 23 Punkte

22. Erweiterte Datenstrukturen und FileIO

Als weitere Verbesserung wurde im Getränkeautomaten eine Protokollierung der getätigten Bestellungen implementiert. Dieser schreibt die Bestellungen in eine Datei, welche nun ausgewertet werden soll. Hierzu benötigen Sie aber zunächst geeignete Datenstrukturen zur Speicherung der Daten. Zwei Strukturdatentypen sollen definiert werden:

Der Typ `BESTELLUNGEN` nimmt hierbei den Zeitpunkt der Bestellung, genannt `Timestamp` auf. Der Automat erzeugt für jeden Tag eine eigene Datei (`bestellungen.csv`, siehe Bild C-21.1), in welcher der jeweilige Bestellzeitpunkt in der Form HHMM (HH = Stunde, MM = Minute), z.B. 1205 für 12:05 Uhr, angeben ist. Weiterhin wird zu jeder Bestellung eine eindeutige, ganzzahlige Identifikationsnummer (ID) für das bestellte Getränk gespeichert. Diese soll in einer Variablen mit dem Namen `Getraenk` gespeichert werden.

In einer zweiten Datei (`getraenke.csv`, siehe Bild C-21.2) werden die Identifikationsnummern der Getränke den jeweiligen Getränken zugeordnet. Hierzu soll erneut die Nummer des Getränks, Feld `Getraenk`, gespeichert werden. Weiterhin besitzt jedes Getränk einen Namen (maximale Länge 20 Zeichen), genannt `Name`, und einen Verkaufspreis (z.B. 2.40 für einen Preis von 2,40€) mit der Bezeichnung `Preis`. Zusätzlich soll für eine spätere Auswertung ein Feld mit dem Namen `Umsatz` erzeugt werden. Dieses besitzt den gleichen Datentyp wie `Preis`. Die Einträge in `getraenke.csv` sind immer nach aufsteigender Identifikationsnummer sortiert, wobei mit einer ID von 1 gestartet wird und keine Zahlen ausgelassen werden können.

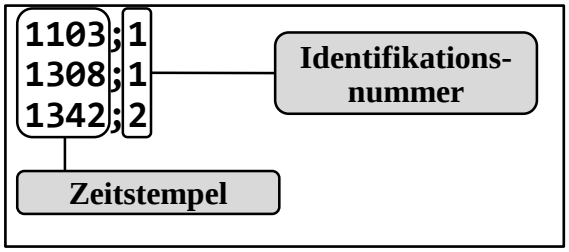


Bild C-22.1: *Auszug aus der Datei* `bestellungen.csv`.

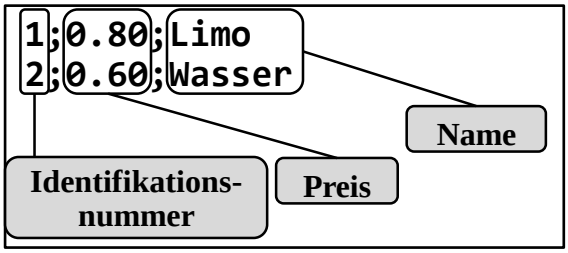


Bild C-21.2: *Auszug aus der Datei* `gestraenke.csv`.

a) Ergänzen Sie den untenstehenden Quelltext (Bild C-22.3) in der Headerdatei `datentypen.h` um die notwendigen Typendefinitionen.

```
// Bestellungen
typedef struct{
    unsigned int Timestamp;  // Verkaufszeitpunkt
    unsigned int Getraenk;       // Identifikationsnummer
 }BESTELLUNGEN;
// Getraenke
typedef struct{
    unsigned int Getraenk;       // Identifikationsnummer
     char Name[20];              // Name des Getraenks
    float  Preis;                // Preis des Getraenks
    float  Umsatz;               // Für spaetere Auswertung
} GETRAENKE;
```

Bild C-22.3: *Typendefinitionen in der Headerdatei* `datentypen.h`.

b) Die in der Headerdatei `datentypen.h` definierten Typen sollen nun in einem Programm zum Einlesen und Auswerten der einzelnen Dateien verwendet werden. Nachdem die Dateien geparst wurden, sollen die Umsätze pro Getränk berechnet und in der Struktur beim jeweiligen Getränk im Feld `Umsatz` gespeichert werden. Vervollständigen Sie den in den Bildern C-22.5 und C-22.6 gegebenen Code für den Spezialfall von nur zwei Getränkesorten, wie in Bild C-21.2 gegeben. Der Programmentwurf ist in Bild C-22.4 als Nassi-Shneidermann-Diagramm gegeben.

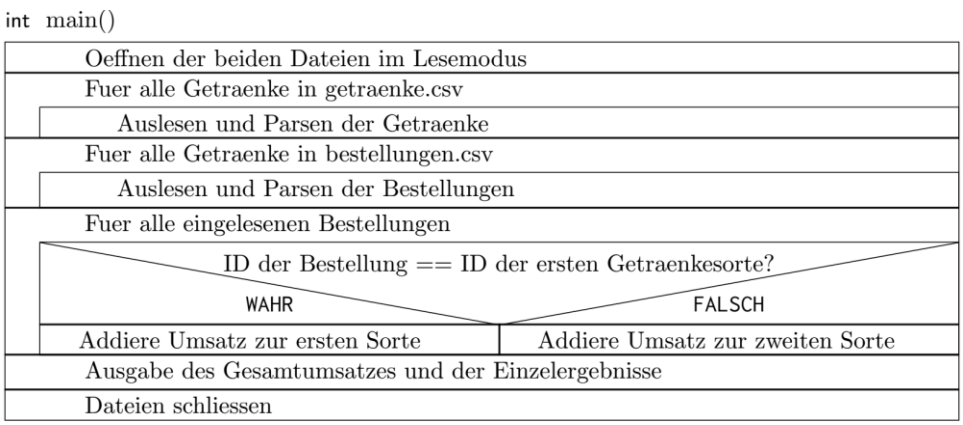


Bild C-22.4: *Nassi-Shneiderman-Diagramm des Auswerteprogramms.*

```
#include <stdio.h>
#include <string.h>
#include "datentypen.h"

//Konstanten mit vordefinierten Laengen
#define NARTIKEL 2
#define NBESTELLUNGEN 3
#define LENGTH 200


int main()
{
    // Variablen und Pointer
    GETRAENKE artikel[NARTIKEL];
    BESTELLUNGEN bst[NBESTELLUNGEN];
    GETRAENKE* pArtikel = &artikel;
    BESTELLUNGEN* pBst = &bst;

    int i = 0;
    float fUmsatz = 0;

    // Dateien oeffnen
    FILE* a = fopen("getraenke.csv","rt");
    FILE* b = fopen("bestellungen.csv","rt");

    // Verfuegbare Getraenke einlesen
    for (i = 0; i < NARTIKEL; i++)
    {
        fscanf(a,"%i;%f;%s\n“, &artikel[i].Getraenk,
               &artikel[i].Preis, artikel[i].Name    );
    }

    // Aufgezeichnete Bestellungen einlesen
    for (i = 0; i < NBESTELLUNGEN; i++)
    {
        fscanf(b,"%i;%i\n", &bst[i].Timestamp      ,
               &bst[i].Getraenk                    );
    }
```

Bild C-22.5: *Programmcode des Auswerteprogramms (Teil 1 von 2).*

```
    for (i = 0; i < NBESTELLUNGEN; i++)
    {
        // Pruefen auf bestellten Artikel
        // und Umsatzberechnung
        if (pBst->Getraenk == pArtikel->Getraenk)
        {
            pArtikel->Umsatz += (*pArtikel).Preis;
        }
        else
        {
            (pArtikel+1)->Umsatz +=
                    (*(pArtikel+1)).Preis;
        }
        pBst++;
    }

    // Ausgabe der Umsaetze
    fprintf(stdout, "Gesamtumsatz heute:\t%.2f\n",
            pArtikel->Umsatz + (pArtikel+1)->Umsatz);
    fprintf(stdout, "Einzelumsatz %s:\t%.2f\n",
            pArtikel->Name, pArtikel->Umsatz);
    pArtikel++;
    fprintf(stdout, "Einzelumsatz %s:\t%.2f\n",
            pArtikel->Name, pArtikel->Umsatz);

    // Dateien schliessen
    fclose(a);
    fclose(b);

    return 0;
}
```

Bild C-22.6: *Programmcode des Auswerteprogramms (Teil 2 von 2).*