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

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Sommersemester 2019

02.09.2019

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält **31** nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

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

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

Aufgaben GL: Grundlagen

Aufgaben GL: 46 Punkte

1. Umrechnung zwischen Zahlensystemen

Über welche Adresslänge verfügt ein im Dualsystem operierender Mikroprozessor mindestens, wenn er rund 65 kByte Speicherblöcke à 1 Byte adressieren kann?

2. IEEE 754 Gleitkommazahlen

Rechnen Sie die Dezimalzahl $(-5{,}375)_{10}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) mit folgender Formatierung um:

V e (3 Bit) M (3 Bit)

Vorzeichen: **V = 1**

Mantisse (Binärzahl und Normalisiert)

M_2=

Exponent

E =

Bias und biased Exponent

B=

e =

Vollständige Gleitkommazahl nach gegebener Formatierung

Kann die gegebene Dezimalzahl bei gegebener Genauigkeit als Binärzahl exakt dargestellt werden?

Ja Nein

Was ist die betragsmäßig größte Dezimalzahl, die mit der gegebenen Formatierung dargestellt werden kann?

3. Querparität

Folgende Nachricht wurde mittels ungerader Parität gegen Übertragungsfehler geschützt. Ermitteln und markieren Sie die Zeile, die den Übertragungsfehler enthält.

0	0	1	**0**
0	1	1	**1**
1	0	0	**1**

4. Logische Schaltungen und Schaltbilder

a) Gegeben sei nebenstehende Wahrheitstabelle (Tabelle 4.1). Erstellen Sie das zugehörige Schaltbild der DNF.

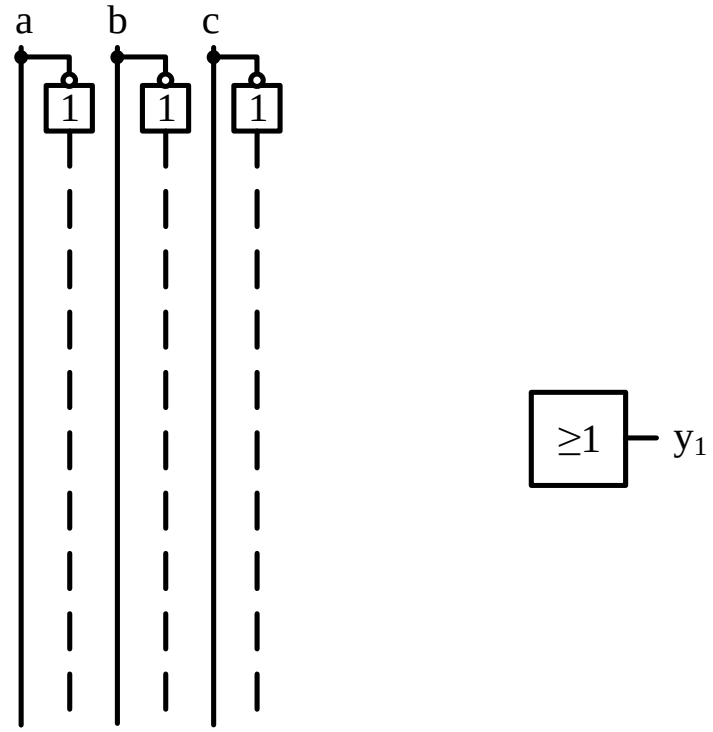


Tabelle 4.1:
Wahrheitstabelle

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

b) Welche Schaltung ist in a) dargestellt? Bitte kreuzen Sie den richtigen Fachbegriff an.

( ) Dreikanal Demultiplexer

( ) 2 Bit Asynchron Zähler

( ) Zweikanal-Multiplexer mit Selektionseingang „c“

( ) XNOR-Gatter für 3 Eingänge

5. Normalformen und Minimierung

Ermitteln Sie die minimierte DNF für die nebenstehende Wahrheitstabelle (Tabelle 5.1) mittels eines KV-Diagramms.

Hinweis 1: Das Einzeichnen der Schleifen in das KV-Diagramm ist als Lösung ausreichend, die minimierte Formel muss nicht abgelesen werden.

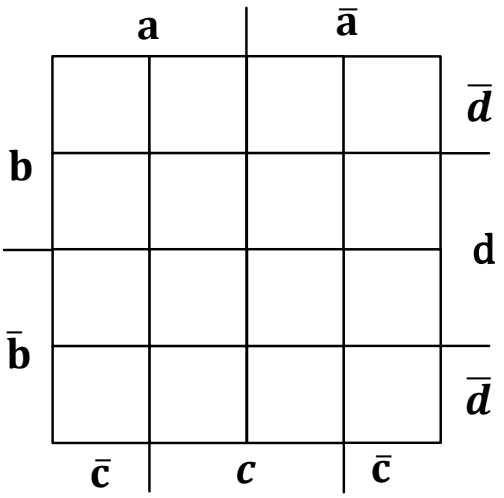
Hinweis 2: Das zweite abgebildete KV-Diagramm dient als Ersatz, falls Sie sich verzeichnen. Kennzeichnen Sie durch Ankreuzen im Feld „dieses KV-Diagramm werten“ eindeutig, welches KV-Diagramm bewertet werden soll.

Hinweis 3: „X“ entspricht „don‘t care“-Einträgen

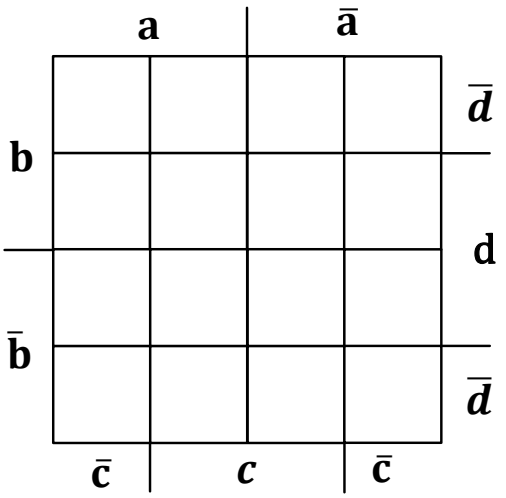
Tabelle 5.1:
Wahrheitstabelle

a	b	c	d	y_1
0	0	0	0	1
0	0	0	1	0
0	0	1	0	X
0	0	1	1	0
0	1	0	0	X
0	1	0	1	0
0	1	1	0	0
0	1	1	1	X
1	0	0	0	X
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	X
1	1	0	1	1
1	1	1	0	0
1	1	1	1	X

☐ Dieses KV-Diagramm werten



☐ Dieses KV-Diagramm werten



6. Flip-Flops

Gegeben ist die folgende Master-Slave-Flip-Flop-Schaltung (Bild 6.1).

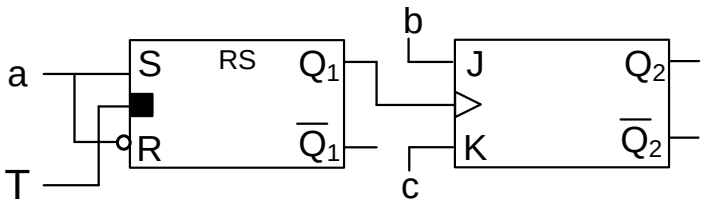
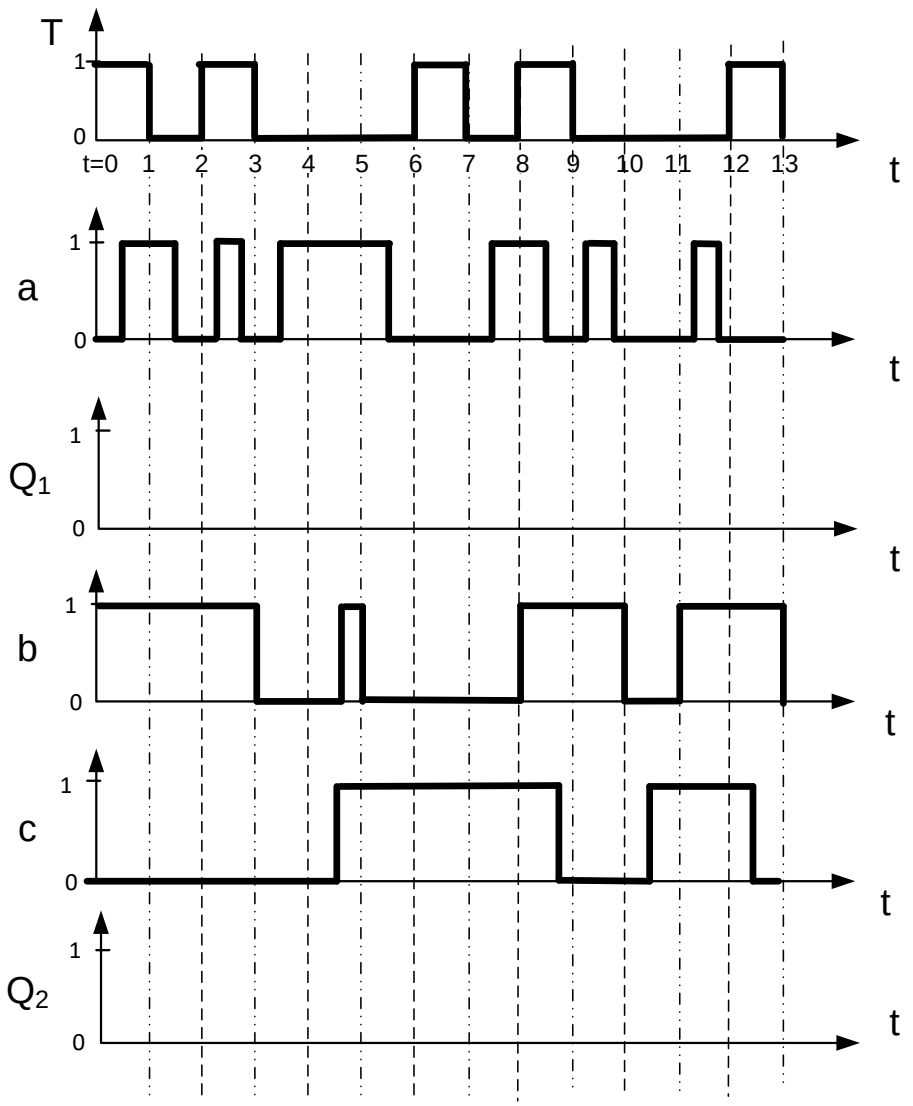


Bild 6.1: *MS-FF*

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

Analysieren Sie die Schaltung für den Bereich $t = [0; 13[$, indem Sie für die Eingangssignale a, b, c 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

Gegeben sei der nachfolgende Algorithmus sowie ein Ausschnitt der MMIX-Code-Tabelle (Bild 7.1), eines Register- (Bild 7.2) sowie eines Datenspeichers (Bild 7.3):

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9		0x_C	0x_D	
…	…	…	…	…	…	…
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	

Bild 7.1: *MMIX-Code-Tabelle*

Algorithmus: $\frac{b(a-258)-c}{255}$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 61 0F	Nicht Veränderbar
$0x87	0x00 00 00 00 00 00 00 06	Variable a
$0x88	0x00 00 00 00 00 00 00 01	Variable b
$0x89	0x00 00 00 00 00 00 00 01	Variable c
$0x8A	0x00 00 00 00 00 00 62 08	Zwischenergebnis
$0x8B	0x00 00 00 00 00 00 01 00	Nicht Veränderbar
…	…	

Bild 7.2: *Registerspeicher*

a) Im Registerspeicher eines MMIX-Rechners befinden sich zu Beginn die in Bild 7.2 gegebenen Werte. In der Spalte *Kommentar* wurde angegeben, welche Daten diese enthalten und wofür die einzelnen Zellen benutzt werden müssen. Führen Sie den gegebenen Algorithmus aus. Übersetzen Sie diese Operationen in Assembler-Code mit insgesamt maximal 5 Anweisungen. Verwenden Sie dazu lediglich die in Bild 7.1 umrahmten Befehlsbereiche. Speichern Sie die Zwischenergebnisse nach jedem Befehl des Algorithmus in der Registerzelle mit dem Kommentar *Zwischenergebnis*.

1

2

3

4

5

b) Nehmen Sie an, der Inhalt der Registerspeicherzelle *Zwischenergebnis* sei nach Ausführung des Algorithmus 0x 01 23 45 67 89 AB CD EF. Speichern Sie diesen als Wyde im Datenspeicher ab Adresse 0x0 … 62 0E. Wie lautet der vollständige Assembler-Befehl zum Speichern? Welche Werte befinden sich nach Ausführung des Speicherbefehls im Datenspeicher (Bild 7.3)?

Befehl:

Datenspeicher	
Adresse	Wert
…	…
0x00 00 00 00 00 00 62 07	
0x00 00 00 00 00 00 62 08	
0x00 00 00 00 00 00 62 09	
0x00 00 00 00 00 00 62 0A	
0x00 00 00 00 00 00 62 0B	
0x00 00 00 00 00 00 62 0C	
0x00 00 00 00 00 00 62 0D	
0x00 00 00 00 00 00 62 0E	
0x00 00 00 00 00 00 62 0F	
0x00 00 00 00 00 00 62 10	
0x00 00 00 00 00 00 62 11	
…	…

Bild 7.3: *Datenspeicher*

c) Wie lautet der Assembler-Befehl ADDI $0x01 $0x02 0x03 in Maschinencode?

Aufgaben BS: Betriebssysteme

Aufgaben BS: 58 Punkte

8. Speicherreservierung

a) Gegeben sei der folgende Speicherzustand eines Datenspeichers. Vervollständigen Sie den Ablauf mittels der Methode Least Recently Used (LRU) in der vorgegebenen Tabelle.

Zeit		0	1	2	3	4
Referenz		-	1	0	2	4
Seite 1		3				
Seite 2		2				
Seite 3		4				
Seite 4		5				
Stapel	0	3				
	1	2				
	2	5				
	3	4				

b) Beurteilen Sie die nachfolgenden Aussagen zum Thema Speicherreservierung und Adressumformung auf ihre Korrektheit.

	wahr	**falsch**
Die Zuordnung von Speicher erfolgt beim Seitenwechsel-verfahren statisch zum Compilerzeitpunkt	( )	( )
Die Größe des nutzbaren logischen Adressraums wird durch die Kapazität des Hintergrundspeichers begrenzt	( )	( )
Durch Segmentierung kann der reservierte Speicherplatz an den Speicherbedarf der Anwendung angepasst werden	( )	( )

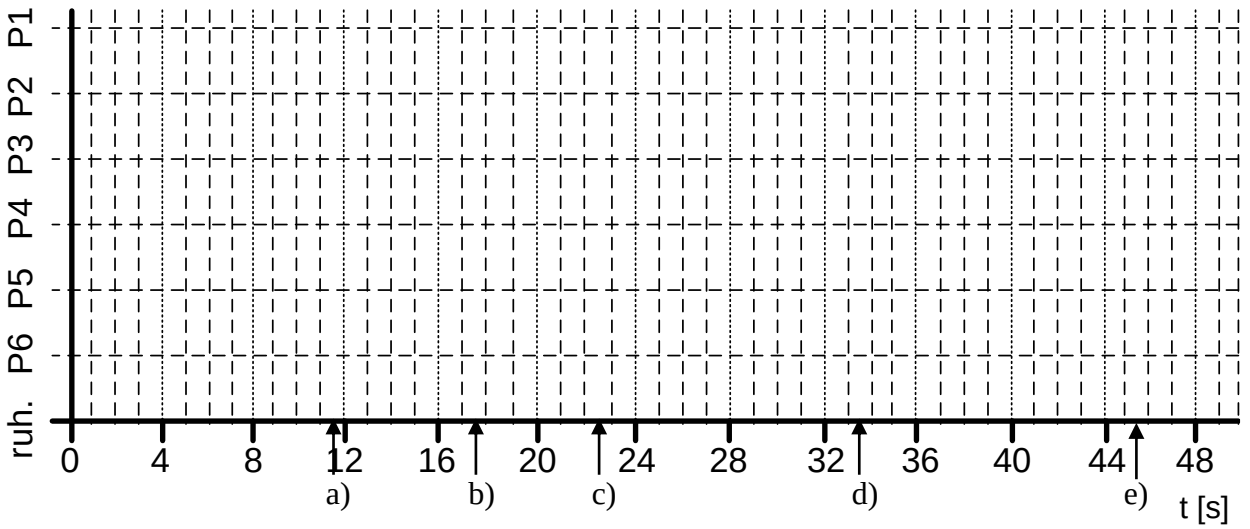
9. Asynchrones Scheduling, präemptiv, Round Robin (RR)

Gegeben seien die folgenden sechs Prozesse (Bild 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 $t = [0; 50[$ s für einen Einkernprozessor. Treffen innerhalb eines Zeitschlitzes mehrere Tasks ein, beachten Sie zuerst die *Prioritäten* und anschließend *FIFO*. Ein Zeitschlitz hat eine Größe von vier Sekunden. Tragen Sie die jeweils zu den gekennzeichneten Zeitpunkten aktiven Tasks ein.

	Priorität	Start	Dauer	Frequenz
P1	1 (hoch)	5 s	4 s	13 s
P2	2	1 s	11 s	Einmalig
P3	3	0 s	4 s	einmalig

	Priorität	Start	Dauer	Frequenz
P4	4	5 s	2 s	einmalig
P5	5	1 s	3 s	einmalig
P6	6 (niedrig)	23 s	5 s	einmalig

Bild 9.1: *Taskspezifikation*



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

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

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

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

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

10. Semaphoren

Gegeben seien die folgenden drei Tasks T1 bis T3 sowie die dazugehörigen Semaphoren S1 bis S3. Die Startwerte der Semaphoren entnehmen Sie der Hilfstabelle (Bild 10.1). Entwerfen Sie die Semaphorenoperationen derart, so dass der eindeutige Taskablauf $T3, T3, \overline{T1, T2}$ entsteht. Tragen Sie die minimal notwendigen Operationen analog zur exemplarischen Semaphorenzuweisung (Bild 10.2) in die Antworttabelle ein. Beantworten Sie zudem die angegebenen Fragen.

Task	S1	S2	S3
-	0	0	4
T3			
T3			
T1			
T2			
T1			
…			

Bild 10.1: *Hilfstabelle Taskablauf*

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

Bild 10.2: *Exemplarische Semaphoren-zuweisung*

T1	T2	T3
…	…	…

Kann eine Semaphore einen Wert < 0 annehmen?

☐ Ja ☐ Nein

11. Echtzeitbetriebssysteme

Im Folgenden (Bild 11.1) ist ein unvollständiges „erweitertes Taskzustandsdiagramm von RTOS-UH“ gegeben.

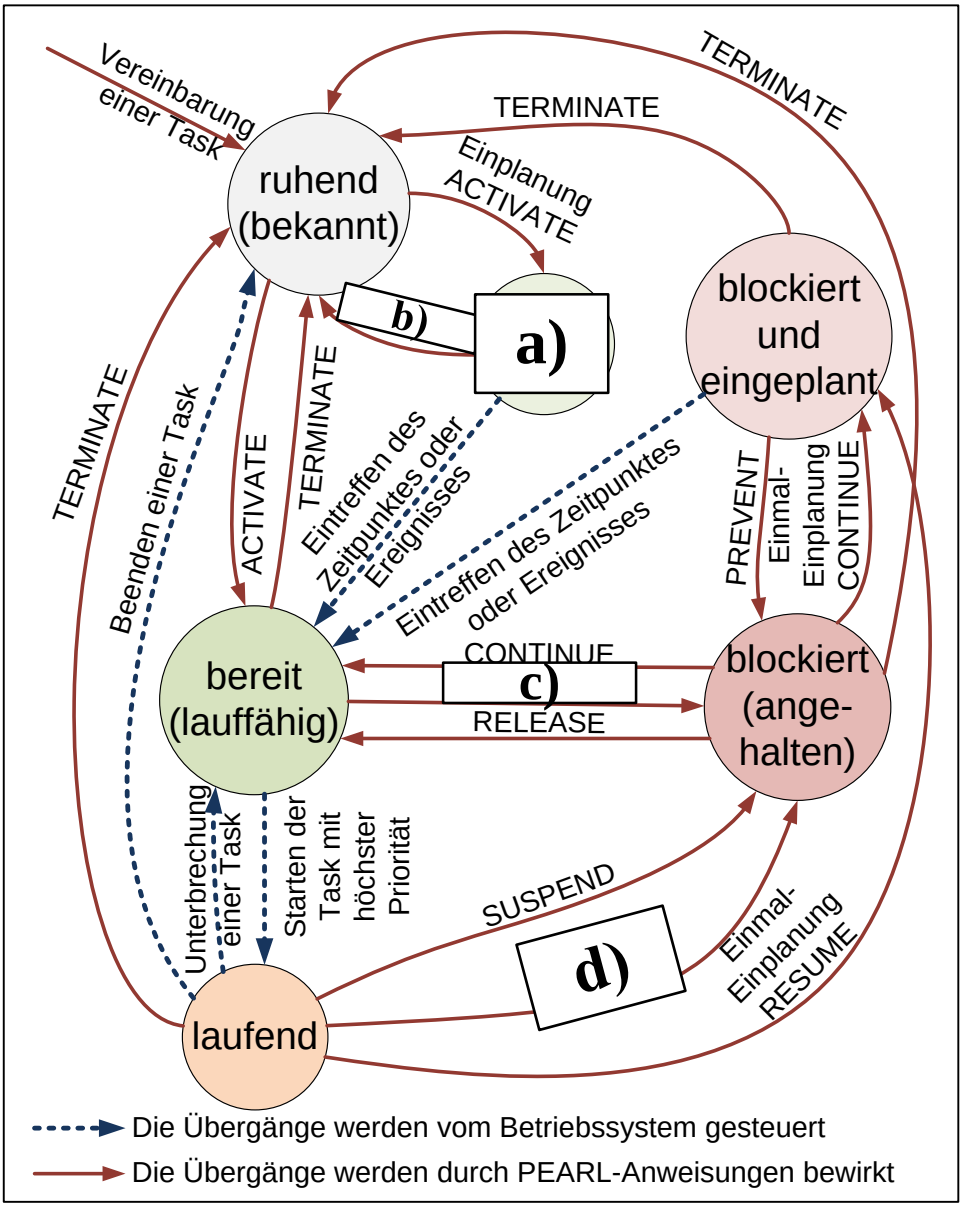


Bild 11.1: *Erweitertes Taskzustandsdiagramm von RTOS-UH*

Bezeichnen Sie die im Diagramm mit Buchstaben markierten Lücken:

a) _____

b) _____

c) _____

d) _____

12. IEC 61131-3: Funktionsbausteinsprache (FBS)

Die Tür einer U-Bahn soll über vier Tasten (*Tür_öffnen_außen, Tür_öffnen_innen, Nothalt, Tür_verriegeln*) sowie einen Sensor (*Zug_in_Bewegung*) gesteuert werden. Der Sollzustand der Tür wird über einen Ausgang (*Tür_offen*) angesteuert.

Wechselt das Signal der Taster *Tür_öffnen_außen* oder *Tür_öffnen_innen* von 0 auf 1, soll sich die Tür öffnen. Wird eine der Tasten länger gedrückt gehalten oder ist verklemmt, könnte die Steuerung beschädigt werden. Dies muss somit bei der Signalauswertung unterbunden werden.

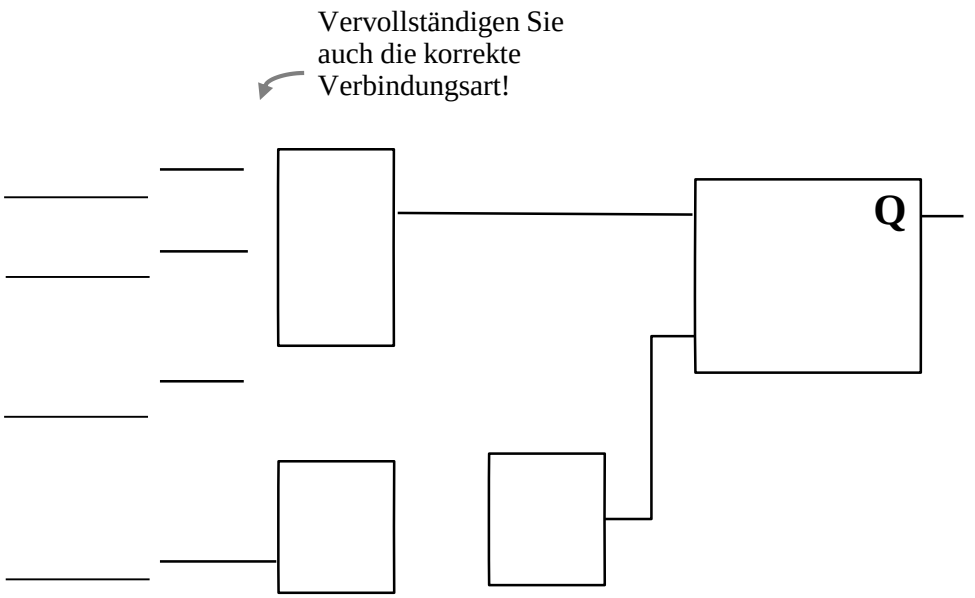
Der Zugführer verfügt über den Schalter *Tür_verriegeln*, mit dessen Hilfe er die Eingaben der Fahrgäste übersteuern kann und die Tür schließt, solange der Zustand dieses Schalters 1 ist.

Für den Notfall existiert der *Nothalt* -Schalter, der im Zustand 1 die Taster der Fahrgäste und den Schalter des Zugführers übersteuert und die Tür öffnet, sobald der Zug zum Stillstand gekommen ist.

Abschließend dürfen die Türen nur geöffnet sein, solange sich der Zug nicht bewegt. Sobald der Bewegungssensor *Zug_in_Bewegung* ausgelöst wird und auf 1 wechselt, muss die Tür in jedem Fall geschlossen werden.

Vervollständigen Sie im Folgenden das Programm der Türsteuerung in der Funktionsbausteinsprache. Achten Sie dabei auch auf die korrekten Verbindungen.

Hinweis: Signalverzögerungen im System sind zu vernachlässigen.



Zug_in_Bewegung

13. Bussysteme

a) Beurteilen Sie die Behauptungen auf ihre Richtigkeit hin. Kreuzen Sie entsprechend an.

	wahr	***falsch***
Bei FlexRay handelt es sich um ein dezentral gesteuertes Buszugriffsverfahren	( )	( )
CSMA/CD eignet sich zum priorisierten Versand wichtiger Nachrichten	( )	( )
Der CAN-Bus setzt CSMA/CA als Buszugriffsverfahren ein	( )	( )
Bei EtherCAT handelt es sich um eine Master-Slave-Architektur	( )	( )

b) Ordnen Sie die nachfolgenden Aufgaben den zuständigen Schichten des ISO 7498-Schichtenmodells zu

	Applikationsschicht	Darstellungsschicht	Sitzungsschicht	Transportschicht	Netzschicht	Sicherungsschicht	Bitübertragungsschicht
Konvertierung von Big und Little Endian	( )	( )	( )	( )	( )	( )	( )
Bereitstellung von Anwendungsdiensten	( )	( )	( )	( )	( )	( )	( )
Routing / Relaying von Datenpaketen durch Netze, via Transitknoten	( )	( )	( )	( )	( )	( )	( )

Der Profibus-DP setzt u.a. Token Passing als Buszugriffsverfahren ein wahr ( ) falsch ( )

Aufgaben MC: Modellierung + Programmierung

14. Automaten

Aufgabe 14:
14 Punkte

Gegeben ist der in Bild 14.1 gezeigte Automat.

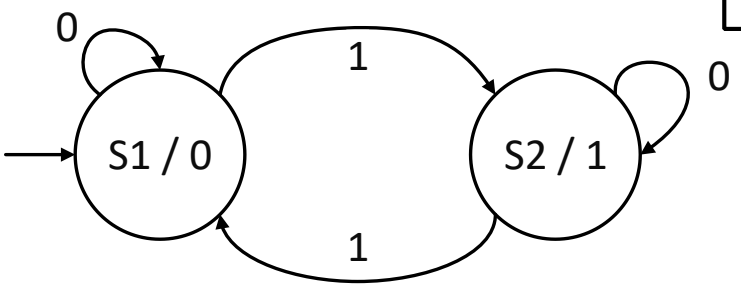


Bild 14.1: *Automat*

a) Zeigt Bild 14.1 einen Mealy- oder einen Moore-Automaten? Kreuzen Sie an.

Mealy ☐ Moore ☐

b) Erstellen Sie zu dem abgebildeten Automaten die zugehörige Übergangstabelle. Welche Sequenzen von Eingaben ausgehend vom Zustand S1 führen zu den Ausgaben 00101 und 10011?

T	s1	s2
0		
1		

Eingabe für Ausgabe 00101:

Eingabe für Ausgabe 10011:

c) Welche Darstellungsform von Automaten benötigt zur Abbildung der selben Logik in der Regel weniger Zustände? Mealy oder Moore?

d) Überführen Sie den Automat in seine andere Form (Mealy in Moore, Moore in Mealy).

Aufgabe 15:
15 Punkte

15. C: Grundlagen

a) Welche Ausgaben erzeugen die `printf`-Anweisungen in den folgenden Codefragmenten? Wählen Sie die korrekte Antwort (nur Einfachantwort möglich) bzw. füllen Sie die Lücken.

```
float x = 4.2;
int y = 2;
float f = x / y;
printf("%.2f", f);
```

( ) 3.100000
( ) 2.10
( ) 2.0
( ) 2

```
float x = 6;
int y[] = {12, 24, 48};
int *z = y;
float f = x / *z;
printf("%.1f", f);
```

→ Ausgabe: _____

```
printf("%i", *(++z));
```

→ Ausgabe: _____

b) Die folgende Aufgabe betrachtet die Umwandlung von Datentypen. Geben Sie durch Ankreuzen an (nur Einfachantwort möglich), welchen Datentyp die Ergebnisse der links angegebenen Berechnungen haben.

```
int * char  / short
int + float / long
```

( )short ( )float ( )int ( )char
( )int ( )float ( )double ( )char

c) Sie erstellen ein textbasiertes Interface für einen Schaufelradbagger. Zum Bewegen des Baggers gibt der Nutzer die Nummer der Raupe und die gewünschte Drehzahl in die Kommandozeile ein (Eingabeformat: X:YYY, X = Raupe, YYY = Drehzahl). Sie möchten diese Eingabe einlesen und in den bereits deklarierten Variablen `int x` und `int y` speichern. Wählen Sie die korrekte Alternative (nur Einfachantwort möglich) bzw. füllen Sie die Lücke.

```
_____(1)_____("%i:%i", _____(2)_____);
```

(1)
( ) fprintf
( ) print
( ) scanf
( ) printf

(2) _____

d) Bestimmen Sie das Ergebnis der Ausdrücke im Dezimalsystem. Gegeben sind die folgenden Variablen:

```
int a = 4;
int b = 6;
int c = 8;
int *d = &c;
```

Nach jedem Ausdruck werden die Variablen auf die oben genannten Werte zurückgesetzt.

d.1)	`(((a & b) | 1) >> 2) + c`	
d.2)	`(b >= *d) + (*d >= a)`	

e) Schreiben Sie jeweils einen boolschen Ausdruck, der die Aussagen der textuellen Beschreibungen wiedergibt. Die Variablen `i`, `j` und `k` sind bereits definiert.

e.1) `i` ist genau doppelt so groß wie j und die Summe aus `j` und `k` ist nicht gerade

```
i == (________) && ((j + k) % 2) ____ 0
```

e.2) `i` ist kein ganzzahliges Vielfaches von j und halb so groß wie k

```
(_________) != 0 && i == ___________
```

f) Füllen Sie die Lücken in der folgenden Abbildung entsprechend der Anweisungen. **Die leeren Zeilen lassen keine Rückschlüsse auf die Länge der korrekten Lösung zu.**

Erstellen Sie eine Schleife, die genau 10-mal durchlaufen wird. Verwenden Sie die Variablen i und inkrementieren Sie diese bei jedem Durchlauf. Geben Sie den Wert von i bei jedem Durchlauf aus.

```
int i = 0;

_______ (_____________________________)
{
    _______ (_________________);

    ____________________________
}
```

Aufgabe 16:
8 Punkte

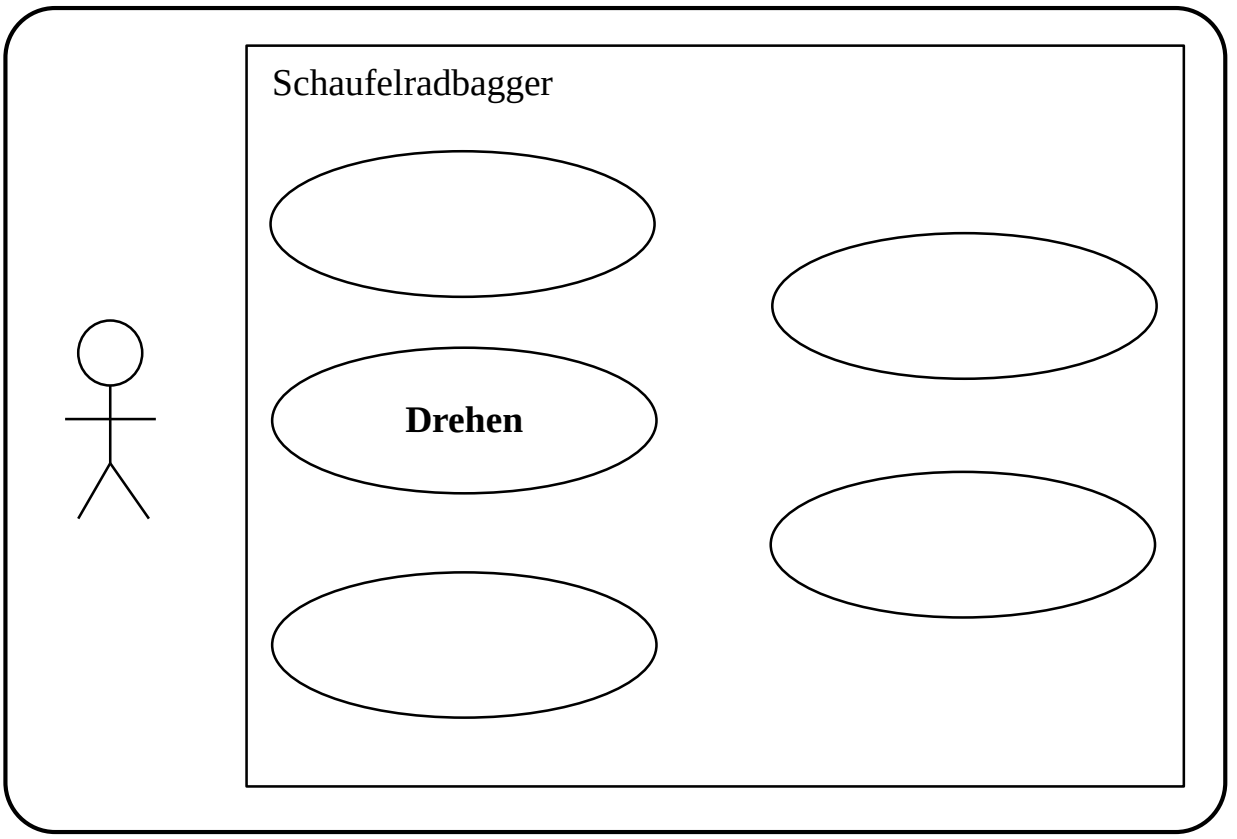
16. Unified Modeling Language

Für die folgenden Aufgaben wird ein Schaufelradbagger betrachtet, der von einem Fahrer gesteuert wird.

Der Bagger erkennt die Kommandos „Fahren“, bei dem er entweder vorwärts oder rückwärts fährt und „Drehen“, bei dem er seinen Arm nach links oder rechts dreht. Erreicht der Bagger dabei seine Endposition, startet er ein Manöver zum „Drehrichtungswechsel“.

Mit dem Kommando „Schaufeln“ wird das Schaufelrad des Baggers eingeschaltet. Für das Schaufeln muss „Schaufeldrehzahl einstellen“ gesetzt werden.

Füllen Sie mithilfe der obigen Angaben das Use-Case-Diagramm der UML für den Schaufelradbagger aus. Benennen Sie die Akteure sowie die Anwendungsfälle. Bitte achten Sie beim Verbinden der Use-Cases auf die richtigen Beziehungen.

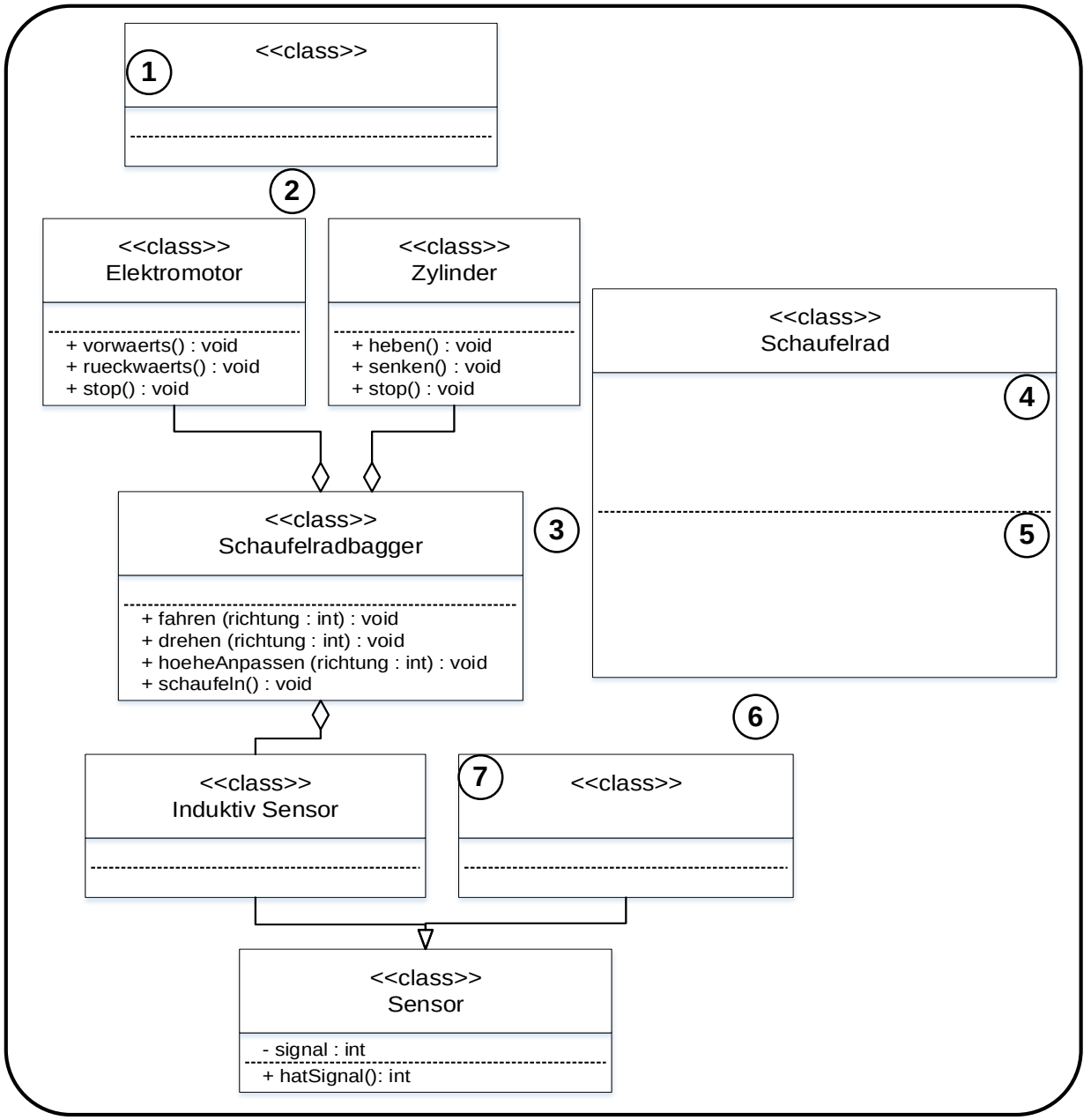


17. Klassendiagramm

Aufgabe 17:
10 Punkte

Zur Hinderniserkennung nutzt der Schaufelradbagger induktive Sensoren und das Schaufelrad optische Sensoren.
Zum Fahren und Drehen besitzt der Schaufelradbagger zwei Aktoren, einen Elektromotor zum Verfahren sowie einen Zylinder zum Heben und Senken des Schaufelrads.
Zum Ein- und Ausschalten besitzt das Schaufelrad eine Methode („umschalten“) sowie eine Methode zum Einstellen der Drehzahl („setzeDrehzahl“). Die Drehzahl („drehzahl“), sowie der Zustand des Schaufelrads („eingeschaltet“) werden als Ganzzahl erfasst und intern gespeichert. Die Methoden des Schaufelrads liefern jeweils den aktuellen Wert zurück.

Füllen Sie die mit Zahlen markierten Lücken im folgenden Klassendiagramm. Lücken **zwischen** Klassen erfordern das Einzeichnen von Beziehungen, Lücken **innerhalb** von Klassen erfordern das Einfüllen von Attributen, Methoden oder Klassennamen. Genaue Attribut- und Methodennamen sind in dieser Teilaufgabe nicht relevant.



Aufgabe 18:
16 Punkte

18. Objektorientierte Programmierung

a) Folgend werden grundlegende Konzepte der objektorientierten Programmierung abgefragt. Bitte wählen Sie aus den Antwortalternativen die korrekte Alternative aus (nur Einfachnennung möglich), oder füllen Sie die markierten Lücken mit dem korrekten Begriff.

1. Wie wird eine Zusammenfassung einer Menge an Objekten mit gleicher Struktur und gleichem Verhalten bezeichnet?

_____

2. Definieren Sie das Prinzip der Kapselung (Information Hiding).

3. Wer kann auf protected-Methoden oder Attribute zugreifen?
( ) Alle anderen Klassen
( ) Klassen im selben Package
( ) Subklassen
( ) Assoziierte Klassen

4. Was beschreiben die Attribute eines Objekts?
( ) Schnittstelle
( ) Verhalten
( ) Zustand
( ) Abstraktion

5. Wann wird ein Konstruktor aufgerufen?
( ) Bei der Instanziierung der Instanz
( ) Bei der Zerstörung der Instanz
( ) Bei jeder Verwendung der Instanz
( ) Bei erstmaliger Verwendung der Instanz

6. Die Klasse *Schaufelradbagger* erbt von der Klasse *Bagger*. Welche der folgenden Aussagen ist falsch?
( ) Ein *Bagger* besitzt die selben Methoden wie ein *Schaufeldradbagger*
( ) Ein *Schaufeldradbagger* kann wie ein *Bagger* verwendet werden
( ) *Schaufeldradbagger* können der Klasse *Bagger* neue Methoden hinzufügen
( ) Bei einem *Schaufeldradbagger* handelt es sich um einen *Bagger*

b) Vom Klassendiagramm zum Code

Sie möchten eine Software zur Steuerung unterschiedlicher Bagger schreiben. Bild 18.1 zeigt das vereinfachte UML-Klassendiagramm für die zu implementierende Software.

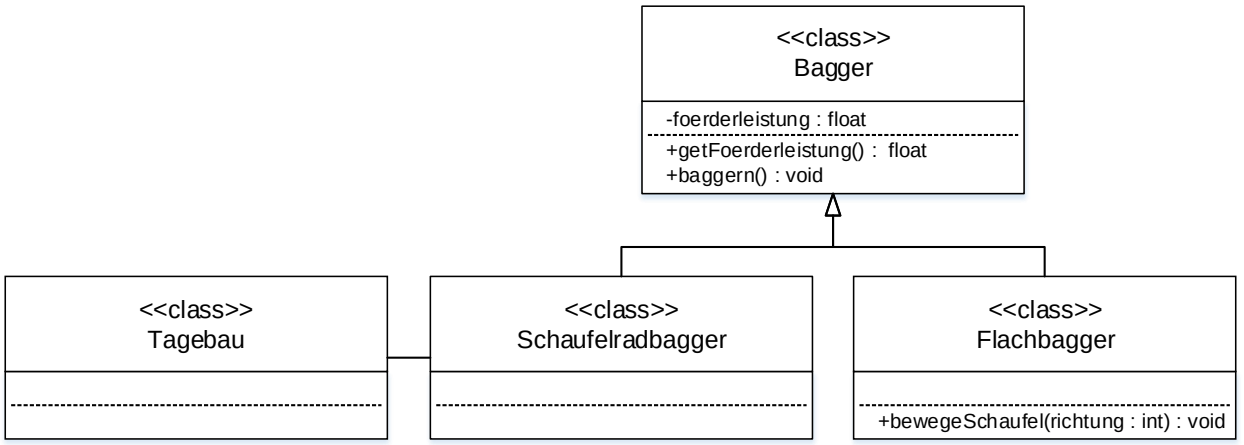


Bild 18.1: Klassendiagramm für Aufgabe 18 b)

Implementieren Sie dieses Klassendiagramm in **C++**, indem Sie den im Lösungskästchen gegebenen Code ergänzen. Deklarieren Sie Attribute und Methoden. Achten Sie auf die korrekten Sichtbarkeiten und Parameter.

```
class Bagger {
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
};

class Schaufelradbagger ____________________________ {
_____________________________________________
_____________________________________________
};

class Flachbagger ___________________________ {
_____________________________________________
_____________________________________________
};

class Tagebau {
};
```

Aufgabe 19:
8 Punkte

19. Zustandsdiagramm

Im Folgenden betrachten wir das selbstständige Schaufelverhalten des Schaufelradbaggers. Bevor der Bagger mit dem Abtragen von Material starten kann, durchläuft er einen Initialzustand in dem alle Ausgänge deaktiviert werden.
Anschließend beginnt der Bagger seinen Schaufelarm zwischen seiner linken und rechten Endposition abwechselnd zu drehen (vgl. Bild 19.1). Die Endpositionen werden detektiert durch die Endschalter `ES_Rechts` und `ES_Links.` Befindet sich der Schaufelarm anfangs nicht auf einer der Endpositionen, soll er mit dem Abtragen nach Rechts beginnen.

Hat der Bagger seinen Arm in die rechte/linke Endposition gedreht, senkt er seinen Schaufelarm für 2 Sekunden, ehe er erneut damit beginnt, Material in die andere Richtung abzutragen.

Wird während eines Senkvorgangs die untere Endposition erreicht, wird die Schaufel bis zur oberen Endposition angehoben und anschließend der Bagger 10 Sekunden vorwärts gefahren, ehe der Materialabtrag fortgesetzt wird. Die vertikalen Endpositionen werden über die Endschalter `ES_Oben` und `ES_Unten` erkannt.

Seitenansicht ***Draufansicht***

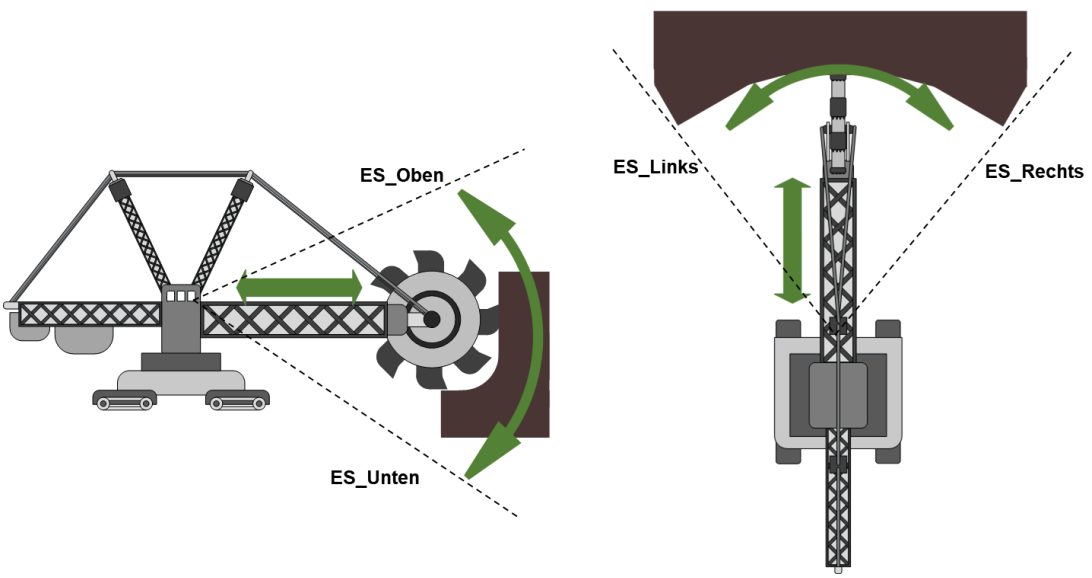


Bild 19.1: Bewegungsabläufe des Schaufelradbaggers

Vorgegeben ist das in Bild 19.2 gezeigte Zustandsdiagramm mit den korrespondieren Zustandsnummern. Füllen Sie die durch römische Ziffern und graue Hinterlegung gekennzeichneten Lücken durch Ankreuzen (nur Einfachnennung möglich) bzw. Angabe der Lösung aus.

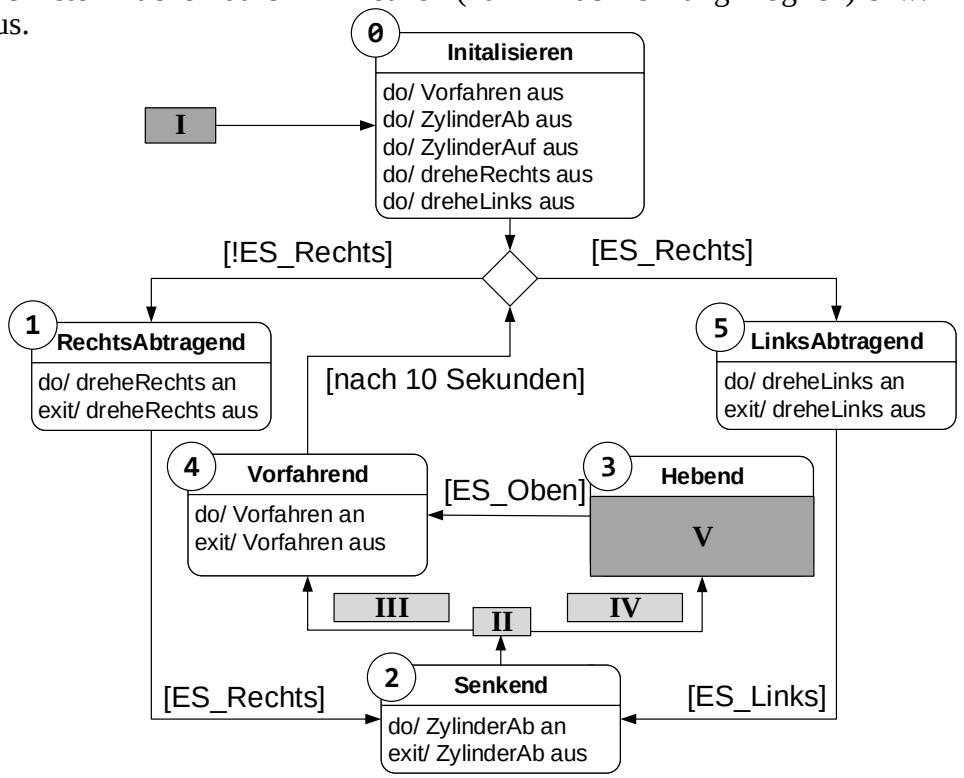


Bild 19.2: Zustandsdiagramm des Schaufelradbaggers

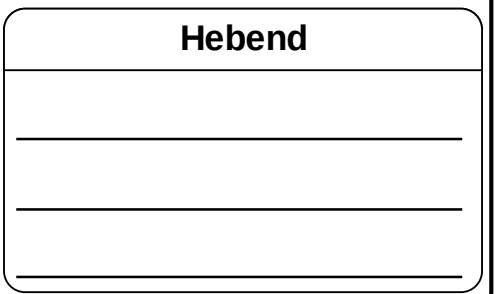
I. Wählen Sie die korrekte Form (nur Einfachnennung möglich).

II. Wählen Sie die korrekte Form (nur Einfachnennung möglich).

III. Geben Sie die korrekte Wächterbedingung an: _____

IV. Geben Sie die korrekte Wächterbedingung an: _____

V. Modellieren Sie den Zustand `Hebend` korrekt aus:



Aufgabe 20:
22 Punkte

20. Zustandsdiagramm zu C-Code

Sie haben nun die Aufgabe, Teil des in Aufgabe 19 modellierten Zustandsdiagramms in Form von C-Code zu implementieren. Dieser soll zur Automatisierung des Baggers dienen. Hierfür stehen Ihnen die in Tabelle 20.1 dargestellten Ein- und Ausgänge sowie erweiterte Variablen zur Verfügung. Weiterhin sind die in Tabelle 20.2 zusammengefassten Funktionen bereits implementiert und erlauben die Interaktion mit der Hardware des Baggers.

Typ	Name	Beschreibung
AKTOREN	`a.Vorfahren`	Schaltet den Motor zum Bewegen des Baggers nach vorne (1) oder aus (0)
	`a.ZylinderAb`	Gibt Druck auf die Seite des Zylinders, welche den Auslieger nach unten fährt (1), oder stoppt Druckzufuhr (0)
	`a.ZylinderAuf`	Gibt Druck auf die Seite des Zylinders, welche den Auslieger nach oben fährt (1), oder stoppt Druckzufuhr (0)
SENSOREN	`s.ES_Rechts`	Endlagesensor rechts. Liefert (1) wenn Auslieger an rechter Endlage angekommen, sonst (0)
	`s.ES_Links`	Endlagesensor links. Liefert (1) wenn Auslieger an linker Endlage angekommen, sonst (0)
VARIABLEN	`vplcZeit`	Aktuelle Laufzeit des Programms in ms (positive Ganzzahl)
	`t`	Variable für timer-Programmierung (positive Ganzzahl)
	`schritt`	Aktueller Zustand des Zustandsautomaten (Ganzzahl), welcher ausgeführt wird

Tabelle 20.1: Sensor- und Aktorvariablen des Baggers, sowie erweiterte Variablen

Funktion	Beschreibung
leseEingaenge (SENSOREN *s, unsigned int* zeit)	Funktion zum Einlesen der aktuellen Sensorwerte und Zuweisung dieser auf die Variable s. Aktualisiert weiterhin die Laufzeit des Programms über einen Zeiger auf die Zeitvariable.
schreibeAusgaenge (AKTOREN *a)	Überträgt die Werte der Aktoren in Variable `a` an die gesteuerten Ausgänge.

Tabelle 20.2: Bereitgestellte Funktionen zur Interaktion mit der Hardware

a) Programmgrundgerüst

Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst, um eine zyklische Ausführung des Zustandsautomaten zu ermöglichen. Verwenden Sie hierfür die in Tabelle 20.1 angegebenen Variablennamen, da diese in anderen Programmteilen so verwendet werden sollen. Zur Interaktion mit der Hardware verwenden Sie die in Tabelle 20.2 gegebenen Funktionen, welche im Header *Bagger.h* bereits definiert und implementiert sind. Der Platzhalter `/* ZUSTAENDE */` soll später durch den spezifischen Code der Zustände in Form eines Zustandsautomaten ersetzt werden.

```
#include "bagger.h"

SENSOREN s;
AKTOREN a;
unsigned int t = 0;

int main ()
{  // Deklarationen
   _____________________ // Zeitvariable
   _____________________ // Schrittvariable

   _____________________ //Zyklische Ausführung
   {
         ___________________________________;
         // Einlesen von Hardware
         ___________________________________
         // Zustandsautomat
         {
                  /* ZUSTAENDE */
         }
         ___________________________________
         // Schreiben auf Hardware
   }
   return 1;     // Wird nicht erreicht
}
```

b) Implementierung Treiber für die Auf- und Abbewegung des Ausliegers

Zur Vereinfachung der Ansteuerung des doppelten wirkenden Zylinders (vgl. Aktorsignale a.ZylinderAb und a.ZylinderAuf in Tabelle 20.1) soll ein Treiber geschrieben werden. Dieser soll in Form einer Funktion realisiert werden. Die Funktionssignatur lautet (entnommen aus Bagger.h):

```
int treiberKranneigung (int iRichtung);
```

iRichtung gibt die auszuführende Richtung der Bewegung wieder. Bitte beachten Sie für die Definition der gültigen Variablenwerte und der zugehörigen Aktorwerte die folgende Tabelle 20.3:

Tabelle 20.3: Eingangs-, Aktor- und Rückgabewerte des Treiberbausteins

iRichtung	Richtung der Bewegung	a.ZylinderAb	a.ZylinderAuf	Rückgabewert
0	Stop	0	0	0
1	Auf	0	1	0
-1	Ab	1	0	0
sonst	Abbruch	0	0	-1

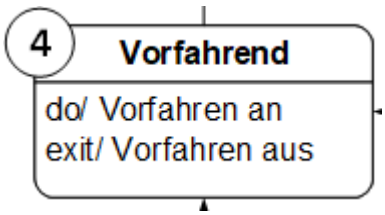
Implementieren Sie im folgenden Lösungskästchen in das vorgegebene Gerüst den Treiber für den doppelt wirkenden Zylinder.

```
_____________________________________
// Funktionskopf
{
   ____________________________ // Auf
   {
      a.ZylinderAb = 0; a.ZylinderAuf = 1;
   }
   ____________________________ // Ab
   {
      a.ZylinderAb = 1; a.ZylinderAuf = 0;
   }
   ______________________ // Stop / Fehler
   {
      a.ZylinderAb = 0; a.ZylinderAuf = 0;
      _________________________ // Fehler abfangen
      {
         return -1;
      }
   }
   return 0;
}
```

c) Implementierung des Zustands „Vorfahren“

Der Zustand Vorfahren soll nun implementiert werden. Einige Codefragmente wurden bereits von einem Kollegen geschrieben, dieser war sich aber mit dem Timer nicht sicher. Deshalb sollen Sie die korrekte Verwendung des Timers ergänzen. Vervollständigen Sie hierfür den vorliegenden Code im Lösungskästchen. Beachten Sie hierbei, dass mehr Leerzeilen zur Verfügung stehen, als für die korrekte Antwort erforderlich sind. Verwenden Sie die in Tabelle 20.1 angegebenen und in Aufgabe 20 a) implementierten Variablen.

```
case 4:   // Vorfahren
   a.Vorfahren = 1;
   _____________________________
   _____________________________
   _____________________________
   _____________________________
   {
      _____________________________
      _____________________________
   }
   _____________________________
   _____________________________
   {
      a.Vorfahren = 0;
      _____________________________
      _____________________________
      _____________________________
      if (s.ES_Rechts) schritt = 5;
      else schritt = 1;
   }
   break;
```



Aufgabe 21:
43 Punkte

21. Algorithmen und Datenstrukturen

Es soll die notwendige Leistung für das Aufrechterhalten einer bestimmten Drehzahl des Schaufelrads auf Basis des Gewichts des Abraums in den Schaufeln berechnet werden. Hierfür können Sie sich auf die letzten fünf Messpunkte (Gleitkommazahlen) beschränken. Zur Verwaltung dieser Messwerte benötigen Sie eine Datenstruktur, die Sie in der folgenden Aufgabe entwickeln.

a) Sie entscheiden sich für einen Ringpuffer als Datenstruktur. Bitte begründen Sie, warum diese Datenstruktur für die beschriebene Anwendung geeignet ist.

Der *Schreibzeiger* und der *Überlauf* sind wichtige Bestandteile eines Ringpuffers. Erläutern Sie kurz beide Begriffe und deren Funktion.

Schreibzeiger:

Überlauf:

Eine weitere Möglichkeit zur Umsetzung der obigen Anwendung wäre es, alle Messwerte in einer Text-Datei zu speichern. Nennen Sie *einen* Vorteil und *einen* Nachteil dieser Lösung im Vergleich zur Verwendung eines Ringpuffers.

Vorteil:

Nachteil:

b) Sie möchten einen Ringpuffer auf Grundlage einer einfach verketteten Liste implementieren. In den folgenden Teilaufgaben implementieren Sie Teile der verketteten Liste und verwenden diese später für die Implementierung des Ringpuffers.

Zunächst implementieren Sie die Datei *List.h*, die die Datenstrukturen und Schnittstellen der Liste enthält. Zunächst definieren Sie die `structs` `NUTZDATEN`, `LISTENELEMENT` und `LISTE`. Das Gewicht des Schaufelinhalts `fDaten` wird als Gleitkommazahl im Strukturdatentyp `NUTZDATEN` gespeichert. Schützen Sie die Header-Datei gegen mehrfaches Einbinden. Füllen Sie die Lücken des Programmcodes im Lösungskästchen.

```
_____
_____

typedef struct  {
  _____; //Gewicht des Schaufelinhalts
} NUTZDATEN;

typedef struct  {
  _____; //Zeiger auf Nutzdaten
  _____; //Zeiger auf nächstes Element
} LISTENELEMENT;

typedef struct {
  _____; //Erstes Listenelement
  _____; //Anzahl der Elemente
} LISTE;

void init (LISTE* liste);
void insertFront (LISTE* liste, NUTZDATEN* nDaten);


_____
```

c) Bild 21.1 zeigt die Komponenten einer einfach verketteten Liste. Füllen Sie die grau hinterlegten Bereiche mit den Namen der Datenstrukturen (umrandete Kästen) und der Zeiger (nicht umrandete Kästen).

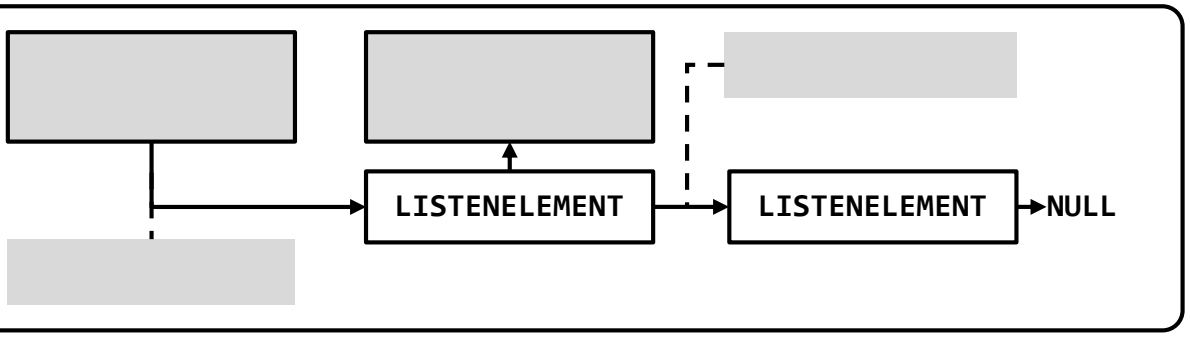


Bild 21.1: *Komponenten der einfach verketteten Liste.*

d) Die Header-Datei *List.h* enthält die Funktion `insertFront(LISTE* liste, NUTZDATEN* nDaten)` (siehe Lösungskästchen). Diese Funktion soll den Speicher für ein neues Listenelement reservieren, es mit den übergebenen Nutzdaten verknüpfen und es am Anfang der einfach verketteten Liste einfügen. Zum Schluss soll die Methode die Anzahl der Elemente der Liste und die Nutzdaten des neuen Elements ausgeben.

Implementieren Sie die Funktion entsprechend der Kommentare und der Methodensignatur. Füllen Sie hierzu die Lücken des Programmcodes im Lösungskästchen.

```
void insertFront (LISTE* liste, NUTZDATEN* nDaten)
{
 //Speicher für neues Listenelement reservieren
 LISTENELEMENT* neuesElement =

   ____________________________________________________;

 //Übergebene Nutzdaten in neuesElement einhängen

 ___________________________________________;

 if (liste->iLaenge == 0) //Liste ist noch leer
 {
   liste -> pFirst = neuesElement; //Zeiger first setzen

   ___________________________________________; //Zeiger next setzen
 }
 else //Liste ist nicht leer
 {
   ___________________________________________;
   liste -> pFirst = neuesElement; //Zeiger first setzen

   ___________________________________________; //Zeiger next setzen
 }


   ___________________________________________;

 printf("Anzahl an Listenelementen: %i.\n",

   ___________________________________________);
 printf("Nutzdaten des Elements: %i.\n",

   ___________________________________________);
}
```

e) In der folgenden Teilaufgabe verwenden Sie die zuvor definierte einfach verkettete Liste, um einen Ringpuffer zu implementieren. **Diese Aufgabe lässt sich unabhängig von den vorhergehenden Teilaufgaben lösen.**

Das folgende `struct` beschreibt die Attribute von `RINGPUFFER`. Die Funktion `initBuffer` füllt den `puffer` mit neu erstellten Elementen entsprechend `iLaenge` und initialisiert die Nutzdaten der neuen Elemente mit 0.

```
typedef struct {
    LISTE* liste; //Verkettete Liste (siehe vorherige Aufgabe).
    LISTENELEMENT* pRead; //Lesezeiger
    LISTENELEMENT* pWrite; //Schreibzeiger
} RINGPUFFER;
void initBuffer(RINGPUFFER* puffer, int iLaenge);
```

Ihnen stehen folgenden Funktionen zur Verfügung. Gehen Sie davon aus, dass diese Funktionen bereits implementiert sind.

```
//Liste initialisieren und in gültigen Ausgangszustand versetzen
void init (LISTE* liste);
//Neues Element mit nDaten versehen und in Liste einfügen
void insertFront (LISTE* liste, NUTZDATEN* nDaten);
```

Füllen Sie die Lücken des Programmcodes im Lösungskästchen, um die Funktion `initBuffer` zu implementieren. Die Funktion reserviert den Speicher für die Liste und initialisiert diese. Anschließend werden `iLaenge` neue Elemente hinzugefügt.

```
void initBuffer(RINGPUFFER* puffer, int iLaenge) {
  int i = 0;
  // Speicher für Liste reservieren
  LISTE* lPufferliste =

    ____________________________________________________;
  puffer->liste = lPufferliste;

  ___________________________________; //Liste initialisieren

  for (i = 0; i < iLaenge; i++) { // Elemente initalisieren
    NUTZDATEN* nDaten = (NUTZDATEN*) malloc(sizeof(NUTZDATEN));

    ___________________________________;//Nutzdaten initialisieren

    ___________________________________________;
  }

  puffer->pWrite = puffer->liste->pFirst;
  puffer->pRead = puffer->liste->pFirst;
}
```

f) Schreiben Sie die Funktion `writeElement`, mit der ein neuer Messwert in den Ringpuffer geschrieben wird. Die Funktion übernimmt als Parameter den Gewichtsmesswert und einen Zeiger auf den Ringpuffer. Gehen Sie davon aus, dass der Ringpuffer bereits vollständig initialisiert ist. Die Elemente der Datenstruktur `RINGPUFFER` sind zur Erinnerung in Bild 21.2 gegeben.

```
typedef struct {
    LISTE* liste;
    LISTENELEMENT* pRead;
    LISTENELEMENT* pWrite;
} RINGPUFFER;
```

Bild 21.2: *Datenstruktur RINGPUFFER.*

Implementieren Sie die Funktion `writeElement(RINGPUFFER* puffer, float fGewicht)`, indem Sie die Lücken des Programmcodes des Lösungskästchens ausfüllen. Achten Sie auf die Behandlung des Überlaufs und setzen Sie den Schreibzeiger nach dem Schreibvorgang.

```
void writeElement(RINGPUFFER* puffer, float fGewicht)
{
  //Nutzdaten in Ringpuffer schreiben
  ____________________________________________ = fGewicht;

  //Behandlung des Überlaufs
  if (___________________________________________)
  {
    ____________________________________________;
  }
  else //Kein Überlauf
  {
    ____________________________________________;
 }
}
```