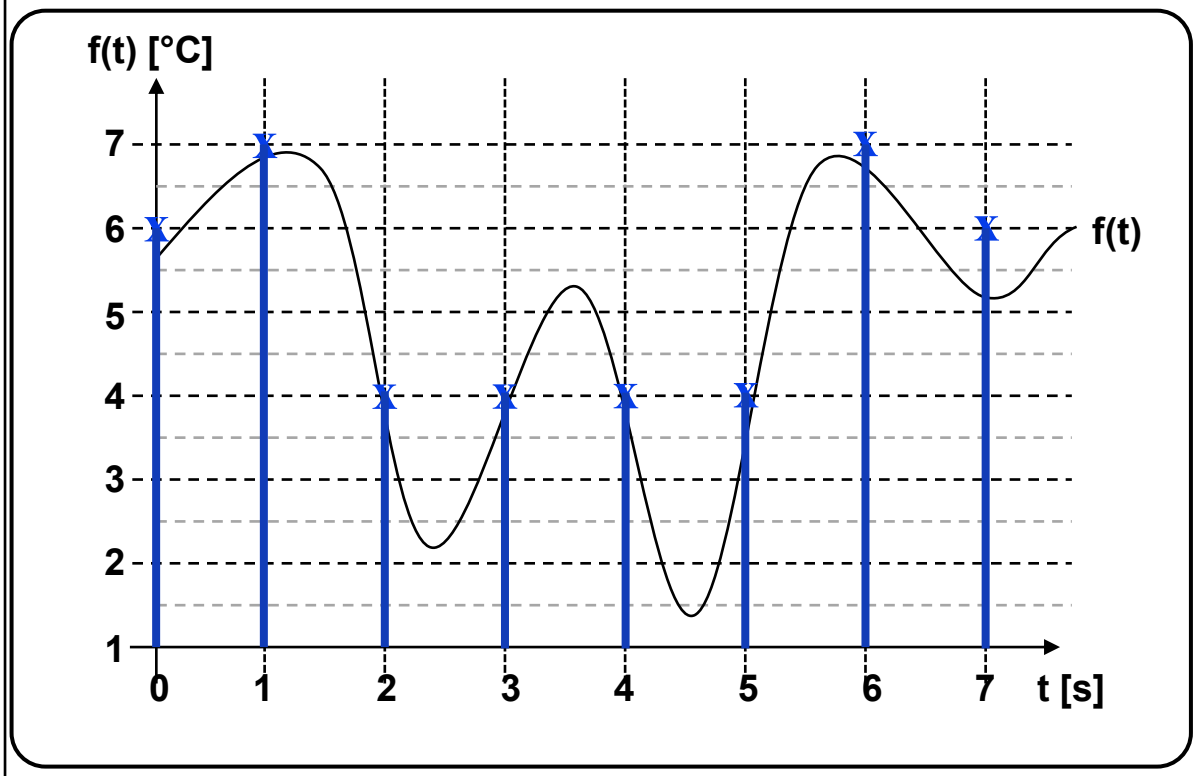
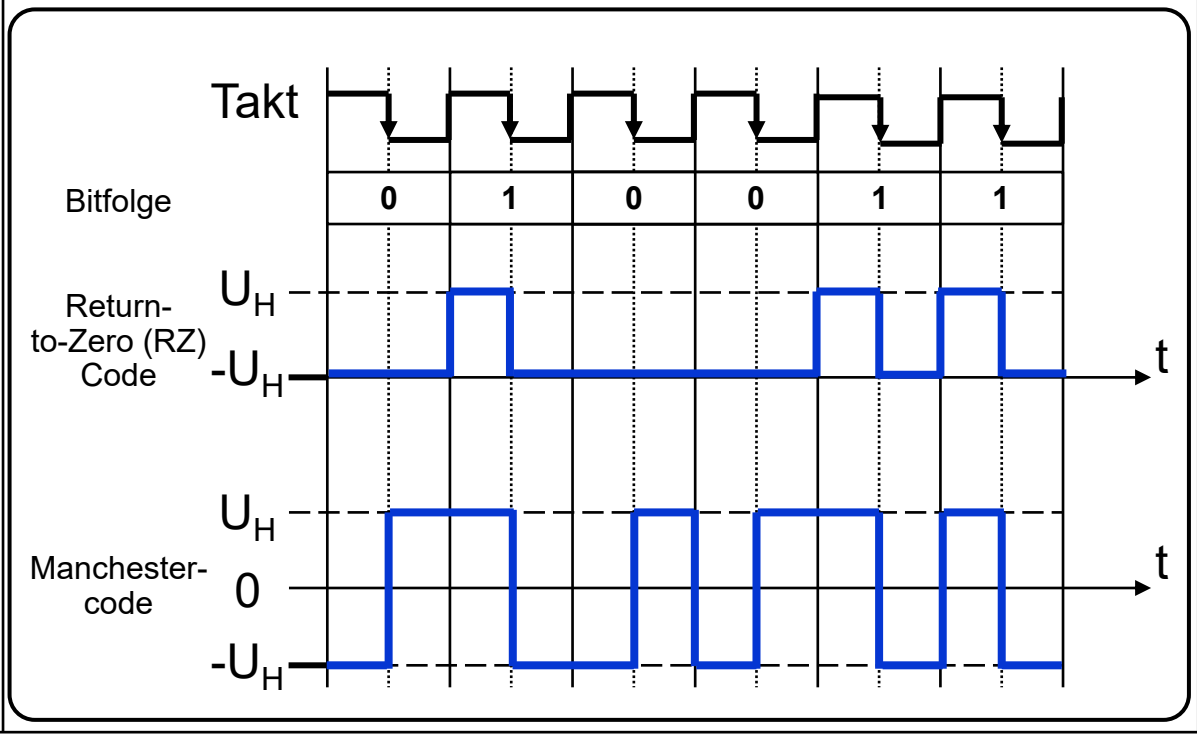
Aufgabe 1: Grundlagen der Informationstechnik und Digitaltechnik

a) Gegeben ist das dargestellte, wert- und zeitkontinuierliche Signal f(t). Zeichnen Sie den wert- und zeitdiskreten Signalverlauf von f(t) für t ∈ [0;7] in das Schaubild ein. Die Diskretisierung erfolgt jede Sekunde [s] auf ganzzahlige °C, wobei alle Werte aufgerundet werden (z.B. 3.1 → 4 und 3.7 → 4).



b) Sie versenden die **Bitfolge `010011`** auf einem seriellen Bussystem. Zeichnen Sie den resultierenden Leitungscode als Return-To-Zero (RZ) Code und im Manchestercode. Für beide Codes liegt **zu Beginn $-U_H$** an.



Aufgabe 2: IEEE 754 Gleitkommadarstellung und Zahlensysteme

a) Rechnen Sie die Dezimalzahl $\mathbf{(-11{,}875)_{10}}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) um, indem Sie die folgenden Textblöcke ausfüllen.

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

Vorzeichen V 1 Bit	Biased Exponent E 5 Bit	Mantisse M 7 Bit

Vorzeichenbit

1

Dezimalzahl $(11{,}875)_{10}$ als Binärzahl

1011,111

Bias als Dezimalzahl

$$B = 2^{(x-1)} - 1 = 2^{(5-1)} - 1 = \mathbf{(15)_{10}}$$

Exponent als Dezimalzahl

$$e = \mathbf{(3)_{10}}$$

Biased Exponent als Dualzahl

$$E = e + B = 3 + 15 = (18)_{10} = \mathbf{(10010)_2}$$

Vollständige Gleitkommazahl (nach obigem Schema)

1	**10010**	**0111110**
Vorzeichen	**Biased Exponent**	**Mantisse**

b) Überführen Sie die unten gegebene Zahl in die jeweils anderen Zahlensysteme. *Hinweis*: Achten Sie genau auf die jeweils angegebene Basis.

$\mathbf{(CD)_{16}}$ = (**11001101**)$_2$

= (**315**)$_8$

Aufgabe 3: Logische Schaltungen und Schaltbilder

a) Vervollständigen Sie das Schaltbild, um die vollständige disjunktive Normalform (DNF) passend zur Wahrheitstabelle (Tabelle 3.1) zu erhalten.

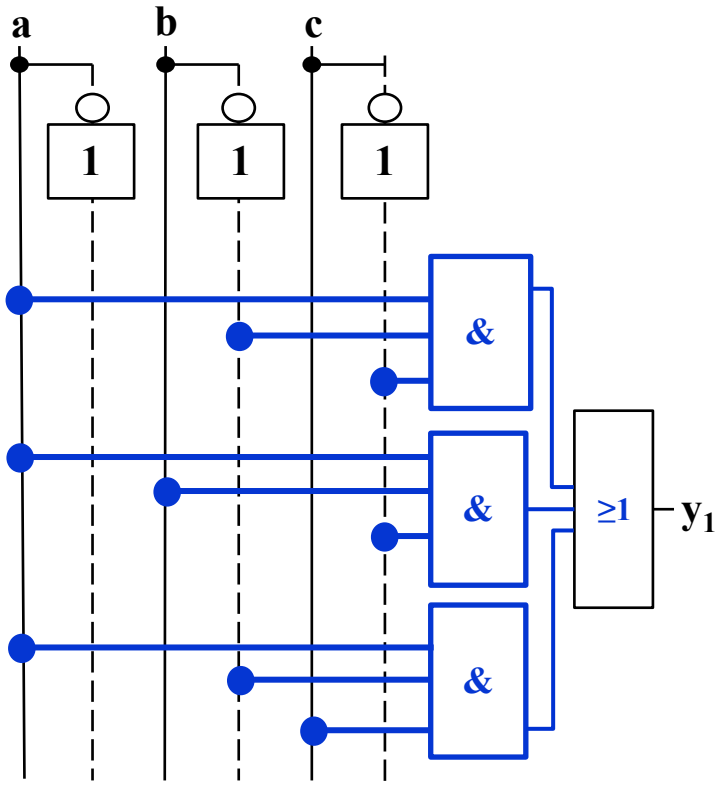


Bild 3.1: *Schaltung*

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

Tabelle 3.1: *Wahrheitstabelle zu Bild 3.1*

b) Gegeben ist folgende Wahrheitstabelle (Tabelle 3.2). Stellen Sie die zugehörige Disjunktive Normalform (DNF)-Gleichung auf.

Hinweis: Die Schreibweise a ∧ b können Sie mit ab abkürzen.

$$y1 = (\bar{a} \wedge \bar{b} \wedge \bar{c}) \vee (\bar{a} \wedge \bar{b} \wedge c) \vee (\bar{a} \wedge b \wedge c) \vee (a \wedge \bar{b} \wedge c)$$

Alt.:

$$y1 = (\overline{a\,b\,c}) \vee (\overline{a\,b}\,c) \vee (\bar{a}\,b\,c) \vee (a\,\bar{b}\,c)$$

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

Tabelle 3.2: *Wahrheitstabelle*

Aufgabe 4: FlipFlops

Gegeben ist die folgende Primary-Secondary-(Master-Slave)-Flip-Flop-Schaltung (Bild 4.1):

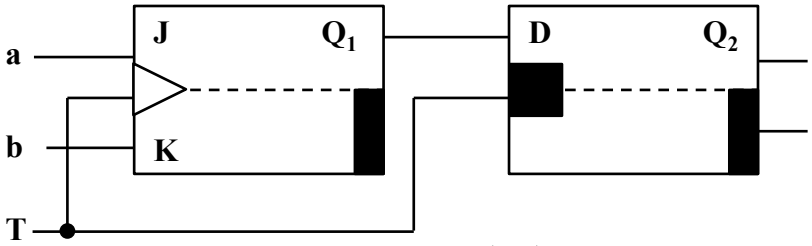
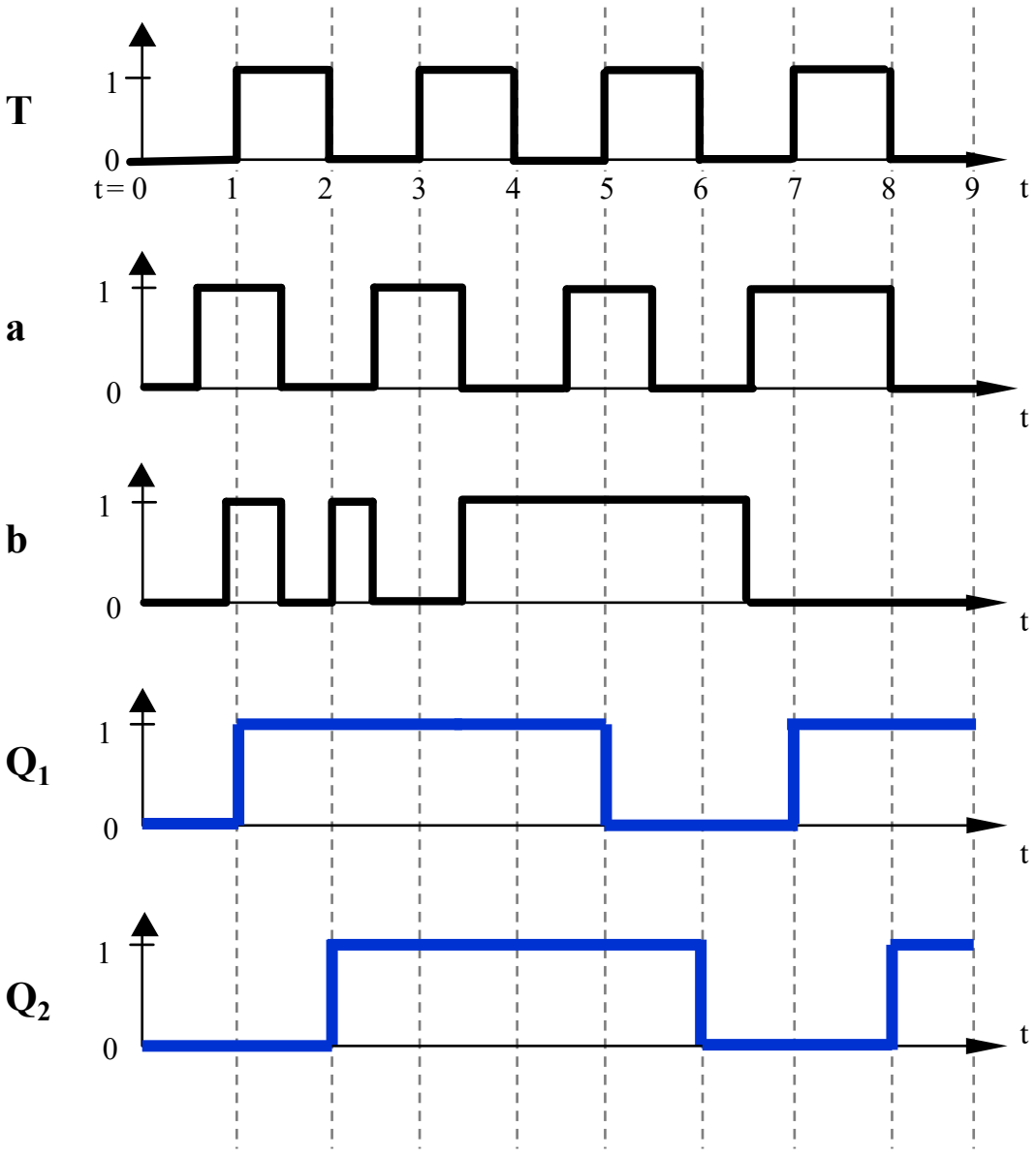


Bild 4.1: *PS(MS)-FF*

Bei t = 0 sind die Flip-Flops in folgendem Zustand: Q1 = Q2 = 0.

Analysieren Sie die Schaltung für den Bereich t=[0;9[, indem Sie für die Eingangssignale a, b und T die zeitlichen Verläufe für Q1 und Q2 in die vorgegebenen Koordinatensysteme eintragen.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.



Aufgabe 5: MMIX – Assembler-Code

Gegeben sei der nachfolgende Algorithmus sowie die Ausschnitte der MMIX-Code-Tabelle (Tabelle 5.1) und eines Registerspeichers (Tabelle 5.2).

	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	

Tabelle 5.1: *MMIX-Code-Tabelle*

Algorithmus:

$$y = \frac{10 \bullet (z+240)^4}{x}$$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x91	0x00 00 00 00 00 00 00 06	Nicht veränderbar
$0x92	0x00 00 00 00 00 00 66 12	Variable x
$0x93	0x00 00 00 00 00 00 DE 34	Variable y
$0x94	0x00 00 00 00 00 00 AB C3	Variable z
$0x95	0x00 00 00 00 00 00 AA 12	Variable a
$0x96	0x00 00 00 00 00 00 EC 11	Zwischen-ergebnis
…	…	…

Tabelle 5.2: *Registerspeicher*

Im Registerspeicher eines MMIX-Rechners befinden sich zu Beginn die in Tabelle 5.2 gegebenen Werte. In der zusätzlichen Spalte Kommentar ist angegeben, welche Daten diese enthalten und wofür die einzelnen Zellen benutzt werden müssen.

a) Führen Sie den gegebenen Algorithmus aus. Verwenden Sie dazu lediglich die in Tabelle 5.1 **umrahmten Befehlsbereiche**. **Speichern Sie die Zwischenergebnisse** nach jedem Befehl des Algorithmus in der Registerzelle mit dem Kommentar **Zwischenergebnis**. Das Endergebnis soll in der Registerzelle mit dem Kommentar **Variable y** gespeichert werden. Übersetzen Sie die Operationen in **Assembler-Code mit insgesamt maximal 5 Anweisungen**.

1	ADDI $0x96, $0x94, 0xF0
2	MUL $0x96, $0x96, $0x96
3	MUL $0x96, $0x96, $0x96
4	MULI $0x96, $0x96, 0x0A
5	DIV **$0x93**, $0x96, $0x92

Alternative Lösung:

```
ADDI $0x96, $0x94, 0xF0
MUL $0x96, $0x96, $0x96
MUL $0x96, $0x96, $0x96
DIV $0x96, $0x96, $0x92
MULI $0x93, $0x96, 0x0A
```

b) Angenommen das Zwischenergebnis im Registerspeicher (Tabelle 5.2) ist nun 0x0...00 00 51 03. Sie speichern es mit dem Befehl STO $0x96, $0x96, $0x91 in den Datenspeicher (Tabelle 5.3). Geben Sie für die vier Eingangsbusse am Datenspeicher an, welcher Wert dort jeweils anliegt. Tragen Sie die durch den Speicherbefehl in den Datenspeicher geschriebenen Werte in den Datenspeicher (Tabelle 5.3) ein.
Hinweis: Für diese Teilaufgabe gilt die Big-Endian Adressierung.

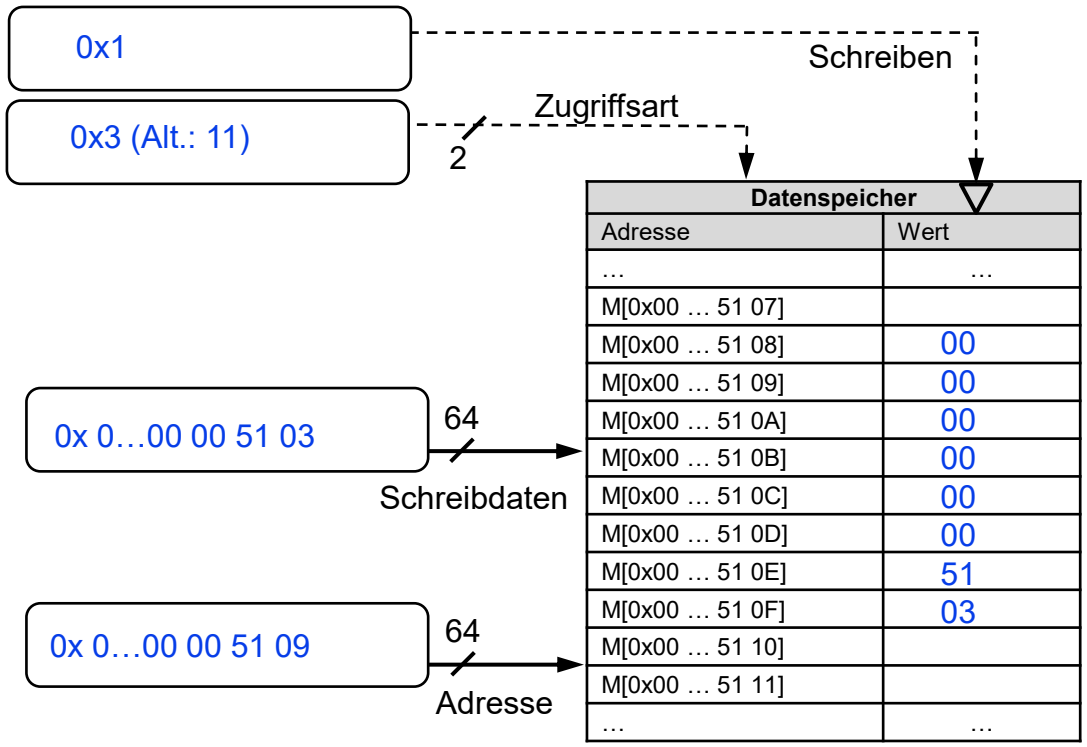


Tabelle 5.3: *Datenspeicher*

c) Übersetzen Sie den folgenden Befehl aus Maschinensprache (binär) in Maschinensprache (Hexadezimal) und in Assembler-Code.
Hinweis: Nutzen Sie die Informationen aus Tabelle 5.1.

Maschinensprache (Binär):	1000 1000 1111 1011 0000 1100 1111 0001
Maschinensprache (Hexadezimal):	0x 8 8 F B 0 C F 1
Assemblersprache:	LDT $0xFB, $0x0C, $0xF1

d) Kreuzen Sie für die zwei folgenden Aussagen an, ob diese jeweils wahr oder falsch sind.

	Wahr	Falsch
1. Das Rechenwerk (ALU) der MMIX-Architektur speichert stets die aktuelle Befehlsadresse.	( )	(X)
2. Ein „Wyde“ entspricht einer Wortlänge von 32 Bit.	( )	(X)

Aufgabe 6: Automaten

a) Gegeben ist der in Bild 6.1 gezeigte Automat. Kreuzen Sie für I. bis III. die jeweils zutreffende Aussage an.

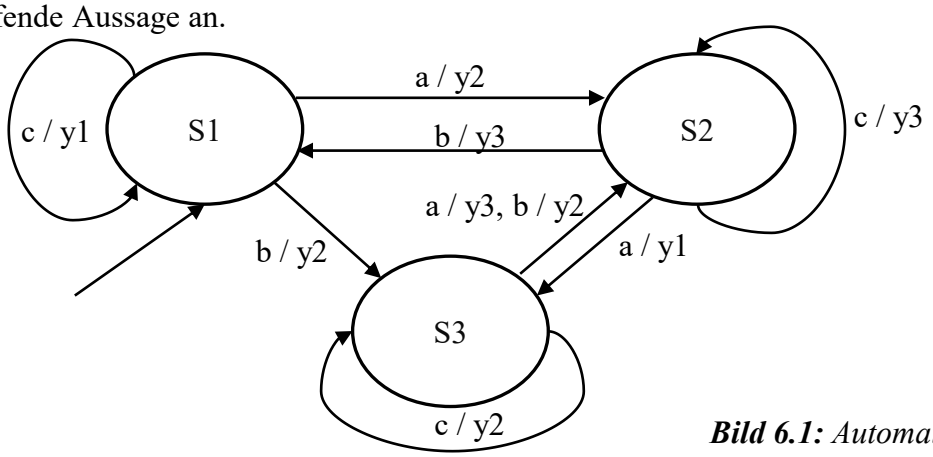


Bild 6.1: *Automat*

Der in Bild 5.1 gezeigte Automat …

I.

(**X**) … ist ein Mealy-Automat

( ) … ist ein Moore-Automat

II.

(**X**) … ist deterministisch

( ) … ist nicht-deterministisch

III.

(**X**) … hat den Startzustand S1

( ) … hat den Startzustand S2

( ) … hat den Startzustand S3

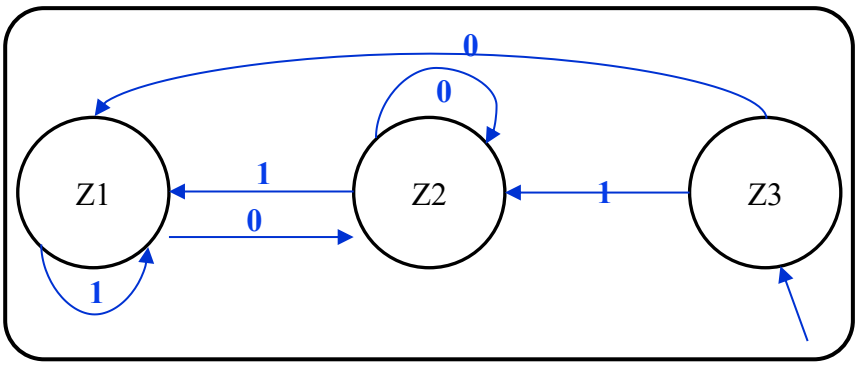
b) Vervollständigen Sie zu dem abgebildeten Automaten (Bild 6.1) die gegebene Übergangstabelle.

T	S1	S2	S3
a	S2, y2	S3, y1	S2, y3
b	S3, y2	S1, y3	S2, y2
c	S1, y1	S2, y3	S3, y2

c) Vervollständigen Sie den Automaten gemäß der gegebenen Tabelle 6.1. Der Automat hat keine Ausgaben. Z3 ist der Startzustand.

T	**Z1**	**Z2**	**Z3**
0	Z2	Z2	Z1
1	Z1	Z1	Z2

Tabelle 6.1: *Übergangstabelle*



Aufgabe 7: Round-Robin-Scheduling

Gegeben sei der folgende Soll-Verlauf der vier Tasks A, B, C und D (Bild 7.1). Die Einplanung der Prozesse soll präemptiv nach festen Prioritäten erfolgen. Für jedes Prioritätslevel (Tabelle 7.2) 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 2T. Restzeiten können nicht übersprungen werden. Tragen Sie für jeden Zeitraum von 1s in t ∈ [0;10] ein, welcher Task auf der CPU läuft. Läuft kein Task auf der CPU, tragen Sie „-“ ein. Kreuzen Sie an, ob die jeweiligen Tasks ihre Deadline DL (siehe Bild 7.1) überschreiten.

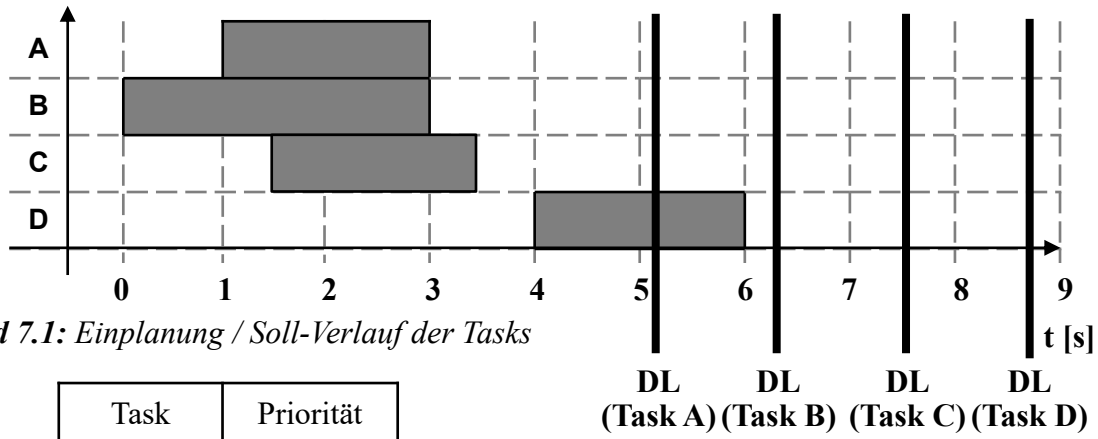
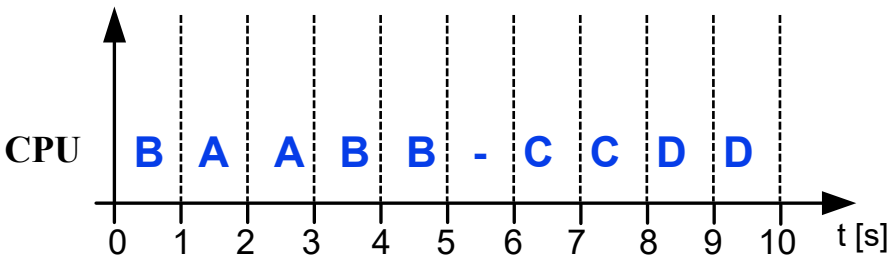


Bild 7.1: *Einplanung / Soll-Verlauf der Tasks*

Task	Priorität
A	1 (hoch)
B	2
C	3
D	4 (niedrig)

Tabelle 7.2: *Prioritätenverteilung*

Ist-Verlauf der Tasks auf der CPU:



Task	Deadline (DL) überschritten?	
A	ja( )	nein(X)
B	ja( )	nein(X)
C	ja(X)	nein( )
D	ja(X)	nein( )

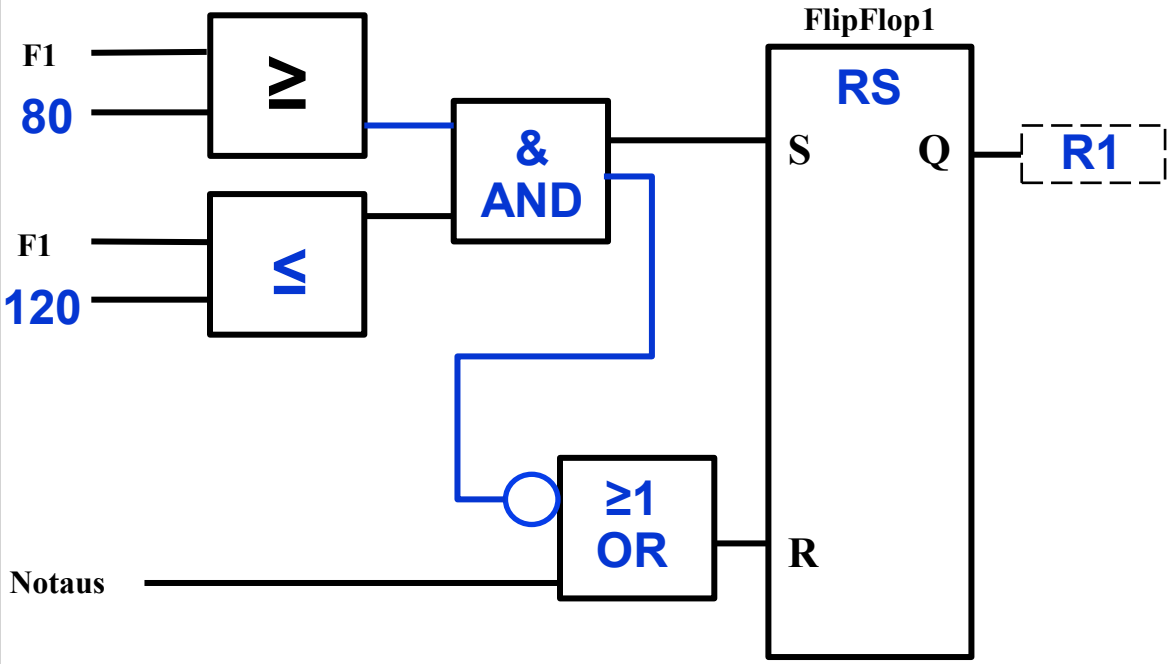
Aufgabe 8: IEC 61131-3 Funktionsbausteinsprache und Ablaufsprache

Ergänzen Sie das untenstehende Programm für die Steuerung eines Rührwerks in einem Produktionsprozess in IEC 61131-3 Funktionsbausteinsprache (FBS).

- Das Rührwerk ist aktiv (`R1=1`), wenn der Füllstand im Kessel (`F1;` gemessen in l) mindestens `80 l` und maximal `120 l` beträgt.
- Das Rührwerk stoppt (`R1=0`), falls der Notaus aktiv ist (`Notaus=1`) oder sobald der Füllstand im Kessel sich außerhalb des zulässigen Bereichs befindet. Im Zweifel stoppt das Rührwerk immer.

Hinweise:

- Signalverzögerungen im System sind zu vernachlässigen.
- Verwenden Sie **keine** Schaltglieder außer den in der Vorlage bereits vorhandenen.
- **Ergänzen Sie Negationen, Flankenerkennung und Flipflopart falls notwendig**.



Aufgabe 9: Echtzeitprogrammiersprache PEARL

Vervollständigen Sie den untenstehenden Codeausschnitt eines Rührwerks in PEARL gemäß den Kommentaren über den Lücken.

```
// Definieren Sie die Unterbrechung „N1“ für den Notaus.
SPC N1 INTERRUPT;
// Definieren Sie den Task „steuern“ mit Priorität 1.
steuern: TASK PRIORITY 1;
// Von nun an soll auf die Unterbrechung N1 reagiert werden.
ENABLE N1;
// Der Task „fuellstandmessen“ wird aktiviert.
ACTIVATE fuellstandmessen;
// Wenn der Füllstand (F1) größer 80 (l) ist, wird der Task
„ruehren“ aktiviert.
IF F1 > 80 THEN ACTIVATE ruehren;
FIN;
// Wenn Unterbrechung „N1“ auftritt, wird der Task „ruehren“
beendet.
WHEN N1 TERMINATE ruehren;
END;
```

Aufgabe 10: UML Use Case Diagramm

Mit dem im Bild 10.1 gezeigten **`Rührkessel`** können Orangensaft und Cola in einem festen Verhältnis gemischt werden, um das Getränk Cola-Mix herzustellen. Der Rührkessel besteht aus **`einem Behälter`**, **`einem Rührer`** und **`drei Ventilen`**.

Zur Gewährleistung der Produktqualität muss ein **`Arbeiter`** regelmäßig den **`Behälter spülen`**.

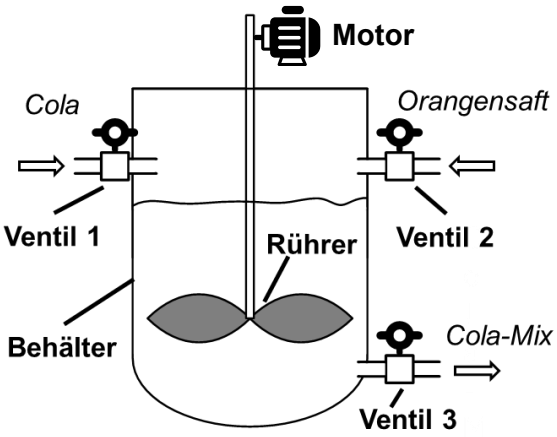
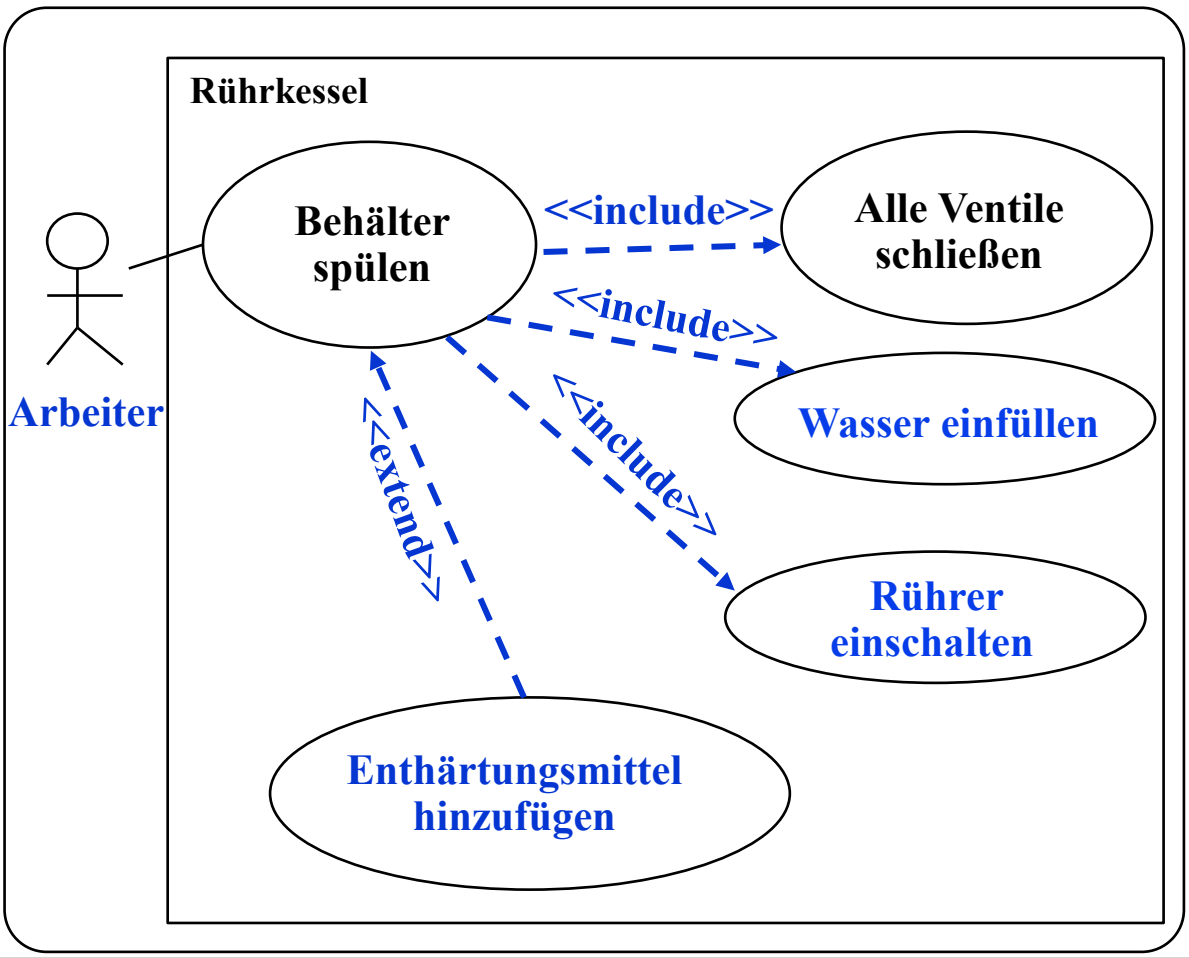


Bild 10.1: *Schematischer Aufbau des Rührkessels für Cola-Mix*

Für den Spülvorgang muss der Arbeiter zuerst **`alle Ventile schließen`**, um den Zu- und Abfluss von Flüssigkeit zu vermeiden. Danach muss der Arbeiter **`Wasser einfüllen`**. Am Ende muss der **`Rührer eingeschaltet`** werden.

Der Arbeiter kann bei besonders kalkhaltigem Wasser zusätzlich ein **`Enthärtungsmittel hinzufügen`**.

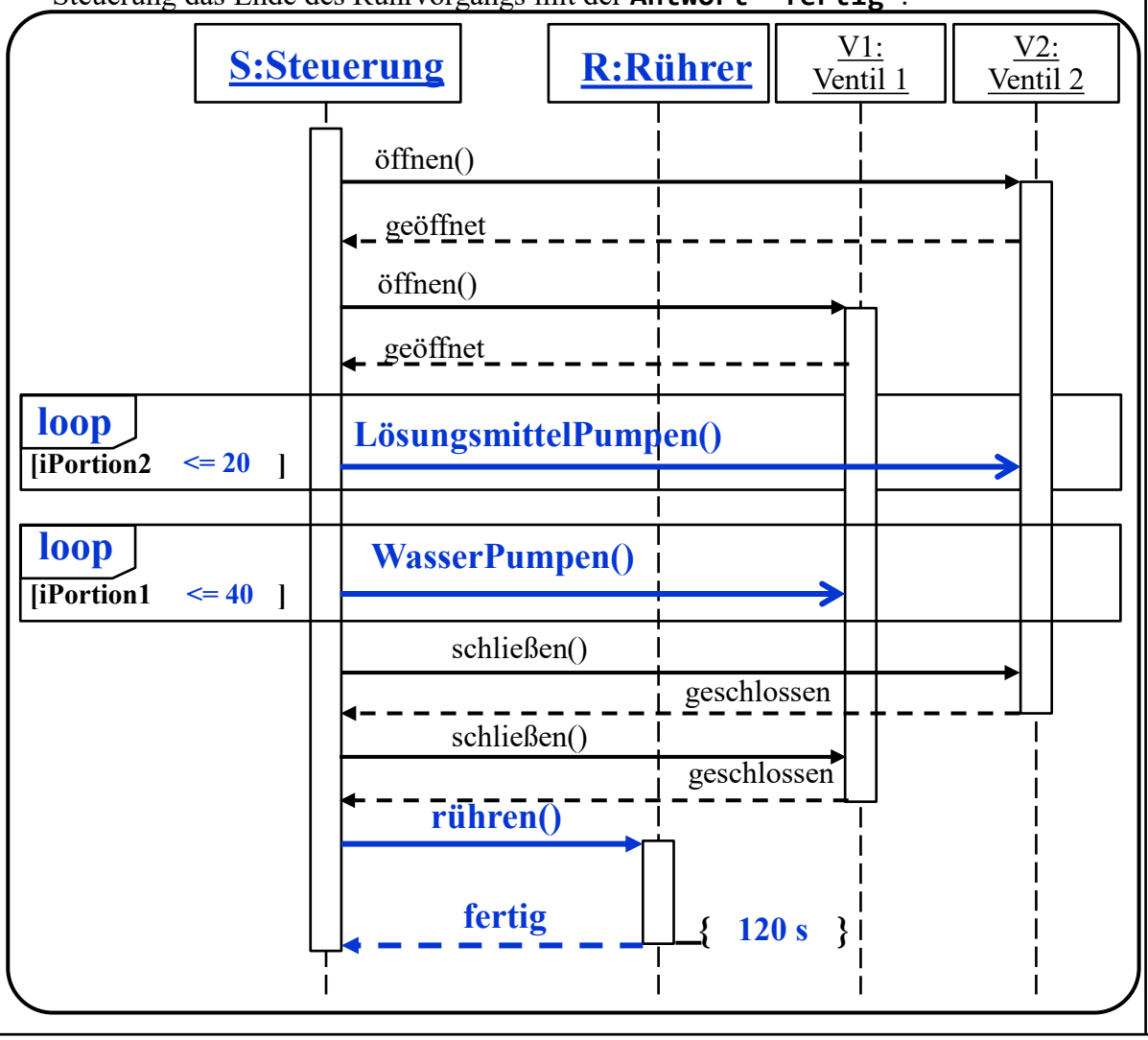
Vervollständigen Sie das untenstehende UML Use Case Diagramm gemäß der oben beschriebenen Anwendungsfälle.



Aufgabe 11: UML-Sequenzdiagramm

Der Rührkessel muss regelmäßig mit Lösungsmitteln gereinigt werden. Dieser Reinigungsvorgang soll in einem Sequenzdiagramm modelliert werden. Hierzu werden die Objekte **`Steuerung (Objekt S)`**, **`Ventil 1 (Objekt V1)`**, **`Ventil 2 (Objekt V2)`** und **`Rührer (Objekt R)`** benötigt. Vervollständigen Sie das folgende Sequenzdiagramm entsprechend der untenstehenden Beschreibung. Achten Sie auf die passenden Pfeilspitzen gemäß der erforderlichen Nachrichtentypen.

- Die **`Steuerung`** öffnet (**`öffnen()`**) zunächst nacheinander die **`beiden Ventile`** und empfängt deren **`Antworten "geöffnet"`**.
- Anschließend fordert die ***`Steuerung`*** das **`Ventil 2`** über eine **`asynchrone Nachricht`** auf, Lösungsmittel einzupumpen (**`LösungsmittelPumpen()`**). Dieser Vorgang wird wiederholt, bis die Menge des durch Ventil 2 gepumpten Lösungsmittels (**`iPortion2`**) **`20`** überschreitet.
- Danach fordert die **`Steuerung`** das **`Ventil 1`** über die **`asynchrone Nachricht`** auf, Wasser einzupumpen **`(WasserPumpen())`**, bis die Menge des durch Ventil 1 gepumpten Wassers (**`iPortion1`**) **`40`** überschreitet.
- Die **`Steuerung`** schließt (**`schließen()`**) **`beide Ventile`** vollständig und empfängt die entsprechenden **`Antworten "geschlossen"`**.
- Schließlich fordert die **`Steuerung`** den **`Rührer`** mittels einer **`synchronen Nachricht rühren()`** auf, **120 Sekunden** lang zu rühren. Der Rührer **meldet** der Steuerung das Ende des Rührvorgangs mit der **`Antwort "fertig"`**.

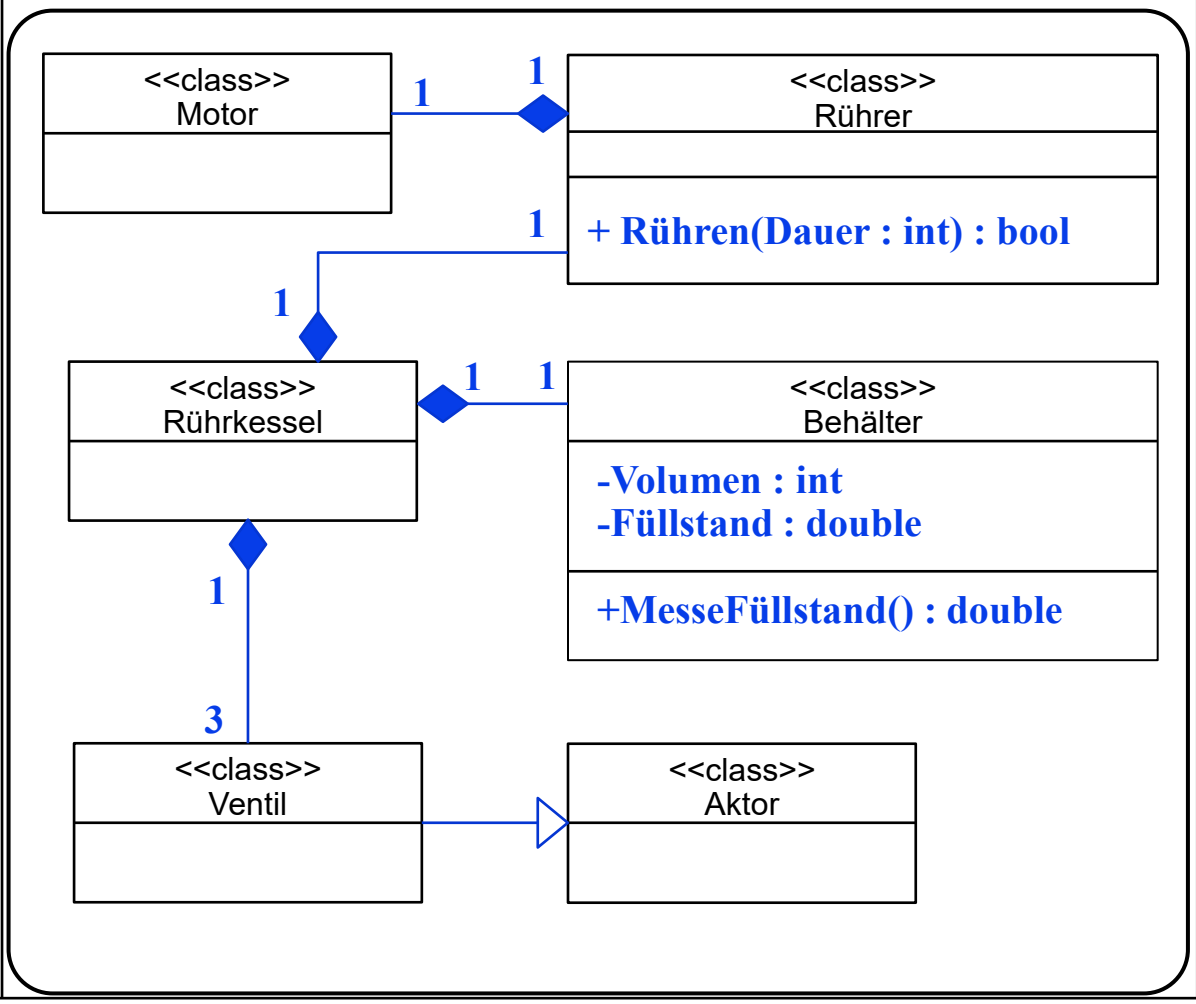


Aufgabe 12: UML-Klassendiagramm

Ein **`Rührkessel`** zur Herstellung des Cola-Mix Getränks muss laut Herstellungsnorm aus genau einem **`Behälter`**, genau drei **`Ventilen`**, sowie genau einem **`Rührer`** bestehen. Keine der aufgeführten Rührkesselkomponenten ist optional.

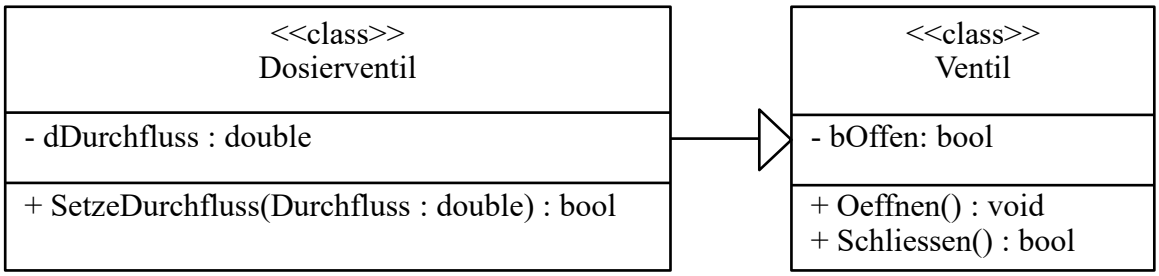
- Ein **`Rührer`** enthält genau einen **`Motor`**, der ihn antreibt. Ohne Motor ist der Rührer nicht funktionsfähig.
- Die Klasse **`Ventil`** ist eine Spezialisierung der Klasse **`Aktor`**.
- Ein Behälter hat die `privaten` Attribute **`Volumen`** (Typ `int`) zur Speicherung des Fassungsvermögens und **`Füllstand`** (Typ `double`) zur Speicherung des aktuellen Behälterfüllstands.
- Der aktuelle Füllstand des Behälters kann durch seine öffentliche Methode **`MesseFüllstand()`** erfasst werden. Der Rückgabewert der Methode ist vom Typ `double`. Der Methode werden keine Parameter übergeben.
- Ein **`Rührer`** lässt sich mittels einer `öffentlichen` Methode namens **`Rühren`** steuern. Der Methode namens **`Rühren`** wird der Parameter **`Dauer`** (Typ `int`) übergeben. Der Rückgabewert der Methode namens **`Rühren`** ist vom Typ `bool` (True = erfolgreicher Abschluss, False = nicht erfolgreich)

Vervollständigen Sie das untenstehende UML-Klassendiagramm um Attribute, Methoden und Beziehungen gemäß obiger Beschreibung. Achten Sie hierbei auf die Kardinalitäten und geben Sie **alle Kardinalitäten** explizit an.



Aufgabe 13: Überführung eines UML-Klassendiagramms in C++-Code

Folgendes Klassendiagramm ist gegeben:



Ergänzen Sie die Klassendeklarationen der zwei dargestellten Klassen **`Dosierventil`** und **`Ventil`** in der Programmiersprache C++.

Hinweise:

- Alle benötigten Header-Dateien sind bereits eingebunden.
- Deklarieren Sie Konstruktoren und Destruktoren.

```
class Ventil {
private:
    bool bOffen;
  public:
    void Oeffnen();
    bool Schliessen();
    Ventil();
    ~Ventil();
};

class Dosierventil : public Ventil {
private:
    double dDurchfluss;
  public:
    bool SetzeDurchfluss(double Durchfluss);
    Dosierventil();
    ~Dosierventil();


};
```

Aufgabe 14: Ergänzen Sie das UML-Zustandsdiagramm

Im Folgenden wird ein Prozessablauf (vgl. Bild 14.2) für das Herstellen eines Cola-Mix Getränks mit dem Rührkessel (Bild 14.1) beschrieben.

Der Rührkessel befindet sich nach dem Start im Zustand **`Wartend`**. Beim Eintritt in diesen Zustand werden aus Sicherheitsgründen **`Ventil 1, Ventil 2`** und **`Ventil 3`** geschlossen **(`iVentil1=0, iVentil2=0, iVentil3=0`)**. Der Rührkessel verbleibt so lange im Zustand **`Wartend`**, bis eine Steuerungsanweisung (**`command`**) mit Wert **`"start"`** erfolgt.

Sobald **`[command=="start"]`** erfolgt ist, geht der Rührkessel in den Zustand **`Cola-Fuellend`** über. Beim Eintritt in den Zustand **`Cola-Fuellend`** wird **`Ventil 1`** geöffnet (**`iVentil1=1`**). Ist der Füllstand des Behälters (**`dFuellstand`**) größer oder gleich einem Wert von **`0.5`**, so wird **`Ventil 1`** wieder geschlossen (**`iVentil1=0`**) und der Rührkessel geht in den Zustand **`Orangensaft-Fuellend`** über.

Bei Eintritt in den Zustand **`Orangensaft-Fuellend`** wird analog zunächst **`Ventil 2`** geöffnet (**`iVentil2=1`**). Erreicht der Füllstand (**`dFuellstand`**) nun einen Wert von größer oder gleich **`0.8`**, so wird Ventil 2 geschlossen (**`iVentil2=0`**) und der Rührkessel geht in den Zustand **`Ruehrend`** über.

In dem zeitgesteuerten Zustand **`Ruehrend`** wird die Methode **`Ruehren()`** dauerhaft ausgeführt. Der Zustand **`Ruehrend`** kann durch zwei Fälle verlassen werden:

Fall 1: Sollte der Füllstand (**`dFuellstand`**) unter einen Schwellwert von **`0.75`** fallen, so wechselt der Rührkessel wieder in den Zustand **`Orangensaft-Fuellend`** und es wird Orangensaft aufgefüllt, wie zuvor für den Zustand **`Orangensaft-Fuellend`** beschrieben.

Fall 2: Der Zustand **`Ruehrend`** wird nach **`7 Sekunden`** verlassen; hierzu wird die Timer-Variable **`iTimer`** verwendet, welche die **`Zeit in Millisekunden`** zählt.

Nachdem der Zustand **`Ruehrend`** durch die Timer-Bedingung (Fall 2) verlassen wurde, wechselt der Rührkessel in den Zustand **`Leerend.`** Im Zustand **`Leerend`** wird das **`Ventil 3`** geöffnet. Ist der Füllstand (**`dFuellstand`**) unter einen Wert von **`0.1`** gefallen, wechselt der Rührkessel wieder in den Zustand **`Wartend`**.

Hinweis: Beim Verlassen **aller** Zustände soll die Timervariable **`iTimer`** zurückgesetzt werden (**`iTimer=0`**).

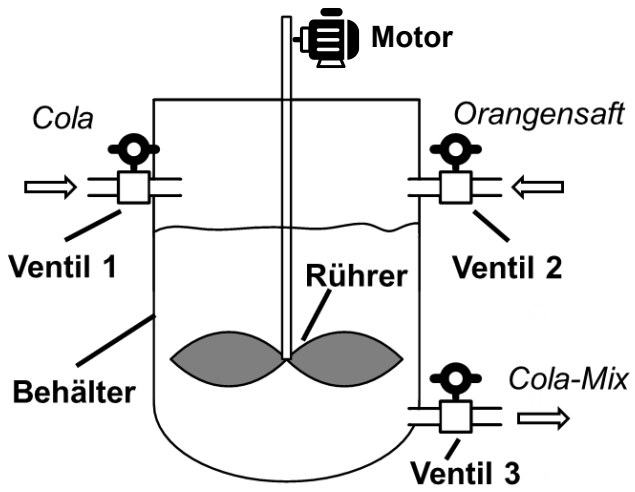


Bild 14.1: Aufbau des Rührkessels zur Herstellung des Cola-Mix Getränks

Bild 14.2 zeigt ein Zustandsdiagramm, welches den zuvor beschriebenen Prozessablauf modelliert.

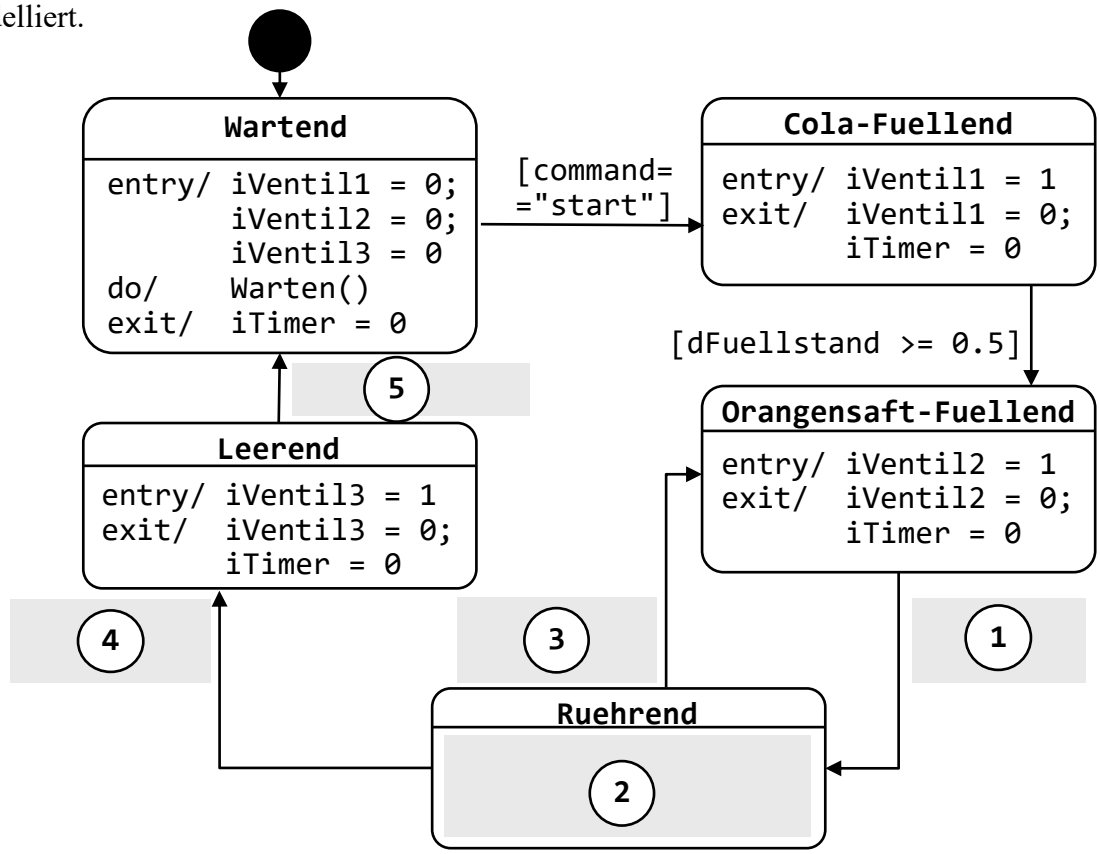


Bild 14.2: Zustandsdiagramm des Prozessablaufs mit zu ergänzenden Lücken ① bis ⑤.

Geben Sie unten an, wie die durch die Ziffern ① bis ⑤ gekennzeichnete Lücken des Zustandsdiagramms in Bild 14.2 ergänzt werden müssen, sodass dieses den Herstellungsprozess nach der gegebenen Beschreibung modellieren.

Geben Sie die korrekte Wächterbedigung für Lücke ① an:

[dFuellstand >= 0.8]

Modellieren Sie den Zustand **Rührend** für Lücke ② aus:

do/Ruehren()

exit/iTimer=0

Geben Sie die korrekte Wächterbedigung für Lücke ③ an:

[dFuellstand <= 0.75]

Geben Sie die korrekte Wächterbedigung für Lücke ④ an:

[iTimer >= 7000]

Geben Sie die korrekte Wächterbedigung für Lücke ⑤ an:

[dFuellstand < 0.1]

Aufgabe 15: UML-Zustandsdiagramm zu C-Code

Implementieren Sie Teile des in Aufgabe 14 beschriebenen Zustandsdiagramms in der Programmiersprache C. Nutzen Sie hierfür die in Tabelle 15.1 vorgegebenen Variablen und Funktionen.

Typ	Name	Beschreibung
VARIABLEN	`int iState`	Variable für den aktuellen Zustand des Rührkessels mit folgender Zuordnung: iState = 0 : Wartend iState = 1 : Cola-Fuellend iState = 2 : Orangensaft-Fuellend **iState = 3 : Ruehrend** *(im Folgenden betrachtet)* iState = 4 : Leerend
	`double dFuellstand`	Variable, die den Füllstand des Behälters angibt und als Bedingung für den Zustandswechsel verwendet wird.
	`unsigned int vplcZeit`	Zählvariable für die aktuelle Zeit der SPS in Millisekunden ab Programmstart.
	`unsigned int iTimer`	Zählvariable für Zeitmessung in Millisekunden.
FUNKTIONEN	`void Ruehren()`	Funktion zum Steuern des Rührers im Zustand Ruehrend, zur Durchmischung des Cola-Orangesaft-Mixgetränks

Tabelle 15.1 Vorgegebene Variablen und vorimplementierte Funktionen

Vervollständigen Sie das folgende Programmgerüst in der Programmiersprache C gemäß den Kommentaren im Lösungskästchen und der Beschreibung des Herstellungsprozesses in Aufgabe 14. Verwenden Sie hierfür die in Tabelle 15.1 angegebenen Variablennamen und Funktionen, welche im Header namens **`Ruehrkessel.h`** bereits deklariert und implementiert sind.

Hinweis: Die Platzhalter /*ZUSTAENDE*/ enthalten spezifischen Code für Zustände des Zustandsautomaten und müssen nicht implementiert werden. Sie können für diese Aufgabe davon ausgehen, dass **`iTimer`** beim Verlassen des Zustands **`Orangensaft-Fuellend`** (**`iState=2`**) auf **`iTimer = 0`** gesetzt wurde.

```
// C-Standard-Bibliothek stdio.h einbinden
#include <stdio.h>
// Benutzerdefinierte Bibliothek Ruehrkessel.h einbinden
#include "Ruehrkessel.h"
// Zustandsvariable iState
iState = 0;
int main () {
// Zyklische Endlosausfuehrung
while(1){          alt.: while(TRUE)
// Zustandsautomat
   switch(iState)
    {
     case 0: /* WARTEND */
     case 1: /* COLA-FUELLEND */
     case 2: /* ORANGENSAFT-FUELLEND */
     case 3:
             Ruehren();          // Methode Ruehren ausfuehren
             // iTimer auf aktuelle Zeit setzen, wenn 0
             if(iTimer == 0) {
                 iTimer = vplcZeit;
             }
             // Abfragen, ob 7 Sekunden vergangen
             else if (vplcZeit-iTimer >= 7000){
                 iTimer = 0;     // iTimer zuruecksetzen
                 iState = 4;     // Wechsel in Zustand 4
              }
             // Abfragen, ob Fuellstand kleiner gleich 0.75
             if(dFuellstand <= 0.75){
                 iTimer = 0;          //iTimer zuruecksetzen
                 iState = 2;          //Wechsel in Zustand 2
             }
             break;
     case 4: /* LEEREND */
     }
 }
 return 0;
}
```

Aufgabe 16: Algorithmen

Es soll für einen Regler eine Rundungsfunktion `fkt` implementiert werden, welche eine Gleitkommazahl `rzahl` (`rzahl ≠ 0`) in Abhängigkeit eines gesetzten Grenzwertes `gwt` auf- oder abrundet. Der Grenzwert ist eine Dezimalzahl mit genau einer Nachkommastelle aus der Menge {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}.

Rundungsgesetz für `rzahl > 0`: Falls der Nachkommaanteil von `rzahl` größer gleich dem Grenzwert `gwt`, soll auf die nächstgrößere Ganzzahl gerundet werden. Andernfalls soll auf die nächstkleine Ganzzahl gerundet werden.

Rundungsgesetz für `rzahl < 0`: Falls der Nachkommaanteil von `rzahl` größer gleich dem Grenzwert `gwt`, soll auf die nächstkleinere Ganzzahl gerundet werden. Andernfalls soll auf die nächstgrößere Ganzzahl gerundet werden.

Beispiel zum Nachkommaanteil:	+23,654244	→ +23	(Vorkommaanteil)
		→ 0,654244	(Nachkommaanteil)
	-23,654244	→ -23	(Vorkommaanteil)
		→ 0,654244	(Nachkommaanteil)

Die Rundungsfunktion `fkt` gibt einen gerundeten Wert vom Typ `int` zurück und bekommt als Übergabeparameter übergeben:

- eine Zahl `rzahl` vom Typ `float,` welche gerundet werden soll.
- den Grenzwert `gwt` vom Typ `float.`

Implementieren Sie die Funktion `fkt` in der Programmiersprache C. Sie dürfen ausschließlich die Bibliothek `<stdlib.h>` verwenden.

Hinweis: Der Fall `rzahl = 0` muss nicht betrachtet werden.

Lösungskasten für Aufgabe 16:

```
int fkt(float rzahl, float gwt)
{
    if(rzahl > 0)
    {
        return (int)((1.0 – gwt) + rzahl);
    }
    else
    {
        return (int)(rzahl - (1.0 - gwt));

    }

}
```

Alternative Lösung:

```
int fkt(float rzahl, float gwt)
{
    if(rzahl > 0)
    {
        if(rzahl – (int) rzahl >= gwt)
        {
                return (int)(rzahl + 1);
        }
        else
        {
                return (int)(rzahl - 1);
        }
    } else {
        if((int) rzahl - rZahl >= gwt)
        {
                return (int)(rzahl - 1);
        }
        else {
                return (int)(rzahl + 1);
        }
    }
}
```

Aufgabe 17: Datenstrukturen

Für eine Getränkemischanlage soll zur Qualitätssicherung eine einfache Datenbankfunktionalität in Form einer **doppelt verketteten Liste mit Listenkopf** implementiert werden, um die hergestellten Produktionschargen (Mengen an Cola-Mix-Mischungen) zu speichern.

Im Listenkopf vom Typ **`ANLAGE`** sollen gespeichert werden:

- Die Anzahl an produzierten Produktionschargen (**`menge`**) als vorzeichenlose Ganzzahl vom Typ `unsigned int`.
- Ein Zeiger (**`pfirst`**) auf das erste Listenelement vom Typ **`CHARGE`**.

Im Listenelement vom Typ **`CHARGE`** sollen gespeichert werden:

- Das Volumen (**`volumen`**) einer Cola-Mix-Mischung im Kessel als Gleitkommazahl vom Typ `double`.
- Ein Qualitätsparameter (**`quality`**), der einen Ganzzahligen Wert zwischen 1 und 200 annehmen kann. Achten Sie auf Speichereffizienz.
- Ein Zeiger (**`pnext`**) auf das nachfolgende Listenelement.
- Ein Zeiger (**`pprev`**) auf das vorherige Listenelement.

a) Implementieren Sie ein Listenelement vom Typ **`CHARGE`** sowie den Listenkopf vom Typ **`ANLAGE`**, indem Sie die Lösungskästchen in der Programmiersprache C ausfüllen.

```
typedef struct charge {
    struct charge* pnext;
    struct charge* pprev;
    double volumen;
    unsigned char quality;

} CHARGE;
```

```
typedef struct {
    unsigned int menge;
    CHARGE* pfirst;

 Alt: struct charge* pfirst

} ANLAGE;
```

b) Implementieren Sie die Funktion **anhaengen** in der Programmiersprache C, welche ein weiteres Listenelement vom Typ **CHARGE** an das Ende der doppelt verketteten Liste anhängt und im Listenkopf vom Typ **ANLAGE** die Anzahl der produzierten Produktionschargen (**menge**) aktualisiert (um eine Produktionseinheit erhöht)**.**

Die Funktion **anhaengen** hat keinen Rückgabewert und bekommt als Übergabeparameter einen Zeiger (**start**) auf den Listenkopf vom Typ **ANLAGE** und das Volumen (**vol**) vom Typ `double` übergeben.

Gehen Sie davon aus, dass die doppelt verkettete Liste nicht leer ist und der Zeiger (**pnext**) des letzten Listenelements auf **NULL** zeigt. Nach dem Anhängen eines weiteren Listenelements soll der Zeiger (**pnext**) des letzten Listenelements auf **NULL** zeigen.

Zu berücksichtigen: Reservieren Sie Speicherplatz für das neue Listenelement vom Typ **CHARGE**, hängen Sie das neue Listenelement korrekt in die doppelt verkettete Liste ein und initialisieren die das Volumen im neuen Listenelement mit dem entsprechenden Übergabeparameter **vol**. Der Qualitätsparameter wird später erst berechnet und muss hier nicht initialisiert werden.

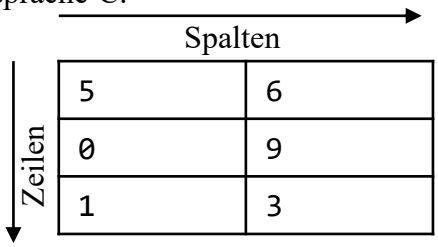
```
void anhaengen (ANLAGE* start, double vol)
{

    CHARGE* tmp = start->pfirst;
    while(tmp->pnext!=NULL)
    {
       tmp = tmp->pnext;
    }
    CHARGE* neu = (CHARGE*)malloc(sizeof(CHARGE));

    neu->volumen = vol;

    tmp->pnext = neu;
    neu->pprev = tmp;
    neu->pnext = NULL;

    start->menge= start->menge+1;
}
```

Korrekturhinweis: Auch struct *NAME* statt typedef zulässig

Aufgabe 18: Grundlagen in C

Lösen Sie die folgenden Aufgaben in der Programmiersprache C.

a) Deklarieren Sie ein Array `arr vom Typ int` mit 2 Spalten und 3 Zeilen und weisen Sie dem Array die Werte in Tabelle 18.1 zu.



5	6
0	9
1	3

Tabelle 18.1 Werte für Array

```
int arr[3][2] = {{5,6}, {0,9}, {1,3}};
```

Alt.: `arr[][2]`

b) Deklarieren und initialisieren Sie einen Pointer `ptr`, der auf das letzte Element eines eindimensionalen Arrays `brr` der Dimension `10` (`brr` enthält 10 Elemente) und vom Typ `float` zeigt.

```
float *ptr = &brr[9];
```

c) Deklarieren und initialisieren Sie einen Funktionspointer `fptr`, der auf die Funktion `scanf` aus der Bibliothek `<stdio.h>` zeigt. Sie müssen `<stdio.h>` nicht einbinden.

Hinweis zu `scanf` aus `<stdio.h>`: `int scanf(const char *format, ...);`

```
int (*fptr)(const char *format, …) = scanf;
```

Alternative:

```
int (*fptr)(const char *format, …) = &scanf;
```

Lesen Sie ausschließlich unter Verwendung des bereits deklarierten und initialisierten Funktionspointers `fptr` einen Wert ein und speichern Sie diesen in der Variablen `var` vom Typ `float`. Sie müssen die Variable `var` zunächst deklarieren.

```
float var;

(*fptr)("%f", &var);
```

Alternative: `fptr("%f", &var);`