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

Prüfung

Grundlagen der modernen Informationstechnik

Wintersemester 2020 / 2021

12.03.2021

MUSTERLÖSUNG
- ohne Gewähr -

Aufgabe 1: Zahlensysteme

Berechnen Sie das Ergebnis des folgenden Terms.

$(B)_{16} + (0101)_2 + (15)_8 - (20)_{10} = (\underline{\quad 21 \quad})_4$

11 + 5 + 13 – 20 = 9

9 / 4 = 2 Rest 1
2 / 4 = 0 Rest 2 → **21**

Aufgabe 2: IEEE 754 Gleitkommazahlen

Rechnen Sie die Dezimalzahl $(12{,}375)_{10}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) mit folgender Formatierung um und führen Sie die rechts im Lösungskasten dargestellte Berechnung durch:

V e (3 Bit) M (4 Bit)

Vorzeichen: **V = 0**

Mantisse (Binärzahl und normalisiert)

M_2= **$(1100, 011)_2 = (1{,}100011 * 2^3)_2$**

Exponent

E = **3**

Bias und biased Exponent

B= **$2^{(x-1)}-1 = 2^{(3-1)}-1 = 3$**

e = **$B + E = (3+3)_{10} = (6)_{10} = (110)_2$**

Vollständige Gleitkommazahl nach gegebener Formatierung

0 110 1000

Multiplizieren Sie folgende Gleitkommazahl mit $(2)_{10}$. Wie lautet das Ergebnis?

1	0	1	0	1	0	0	1

V e (3 Bit) M (4 Bit)

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

Aufgabe 3: Querparität

Die folgende Nachricht wurde mittels ungerader Parität gegen Übertragungsfehler geschützt. Rekonstruieren Sie die fehlenden Daten und beantworten Sie die Frage rechts.

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

Wie viele Bits könnten mit Querparität fehlerhaft übertragen worden sein, ohne erkannt zu werden? (1 Bit oder 2 Bits)

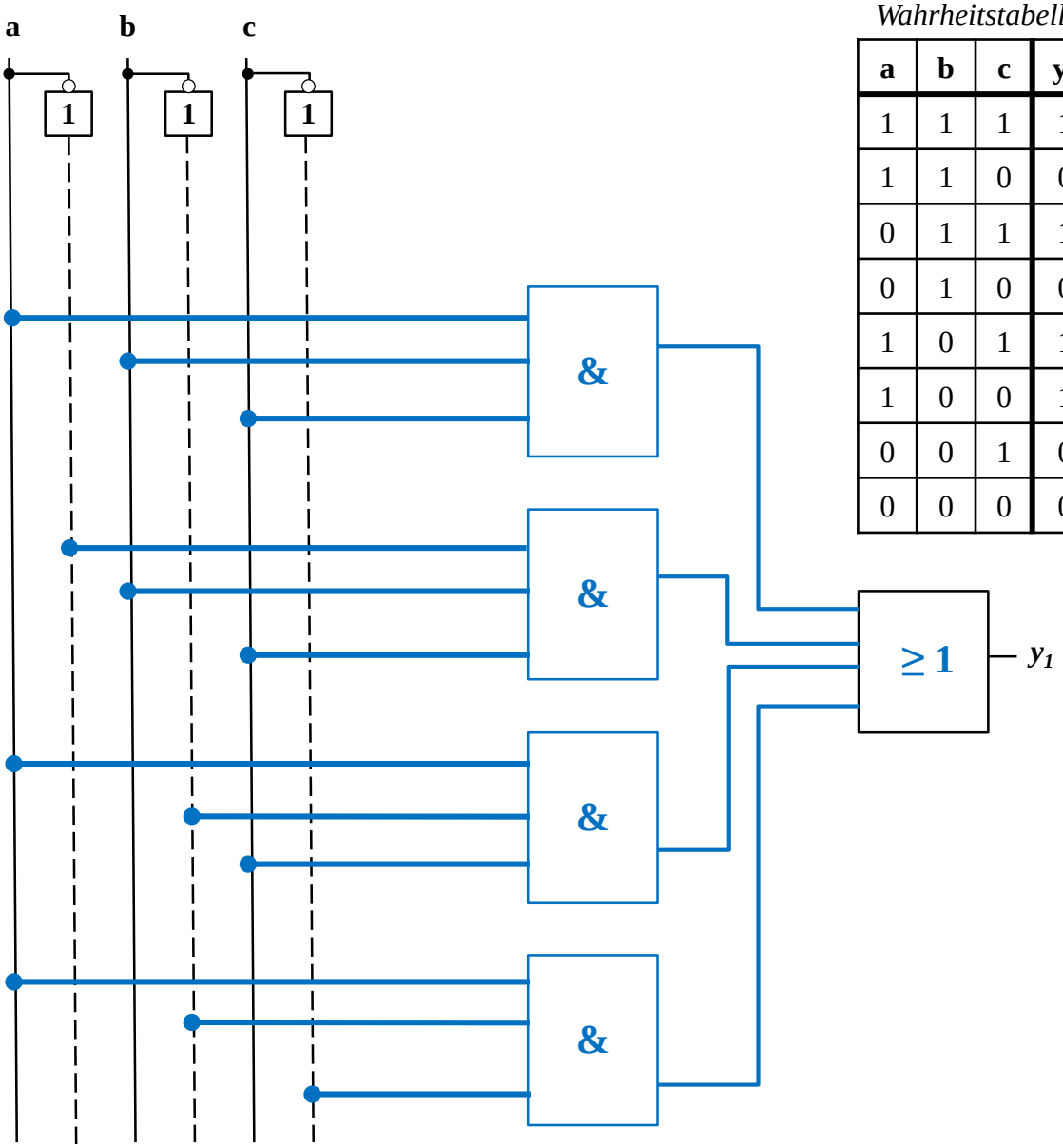
2 Bits.

Aufgabe 4: Logische Schaltungen und Schaltbilder

a) Gegeben sei die nebenstehende Wahrheitstabelle (Tabelle 4.1). Erstellen Sie das zugehörige Schaltbild der DNF. Statt ≥ können Sie auch >= schreiben.

Tabelle 4.1:
Wahrheitstabelle

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



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

( ) NAND-Gatter für 3 Eingänge

( ) Zweikanal-Demultiplexer mit Selektionseingang „c“ und Dateneingang y

(X) Zweikanal-Multiplexer mit Selektionseingang „b“

( ) Negiertes XOR-Gatter für 3 Eingänge

Aufgabe 5: Flip-Flops

Gegeben ist die folgende Master-Slave-Flip-Flop-Schaltung (Bild 5.1).

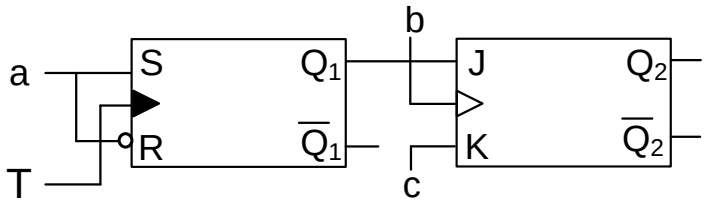
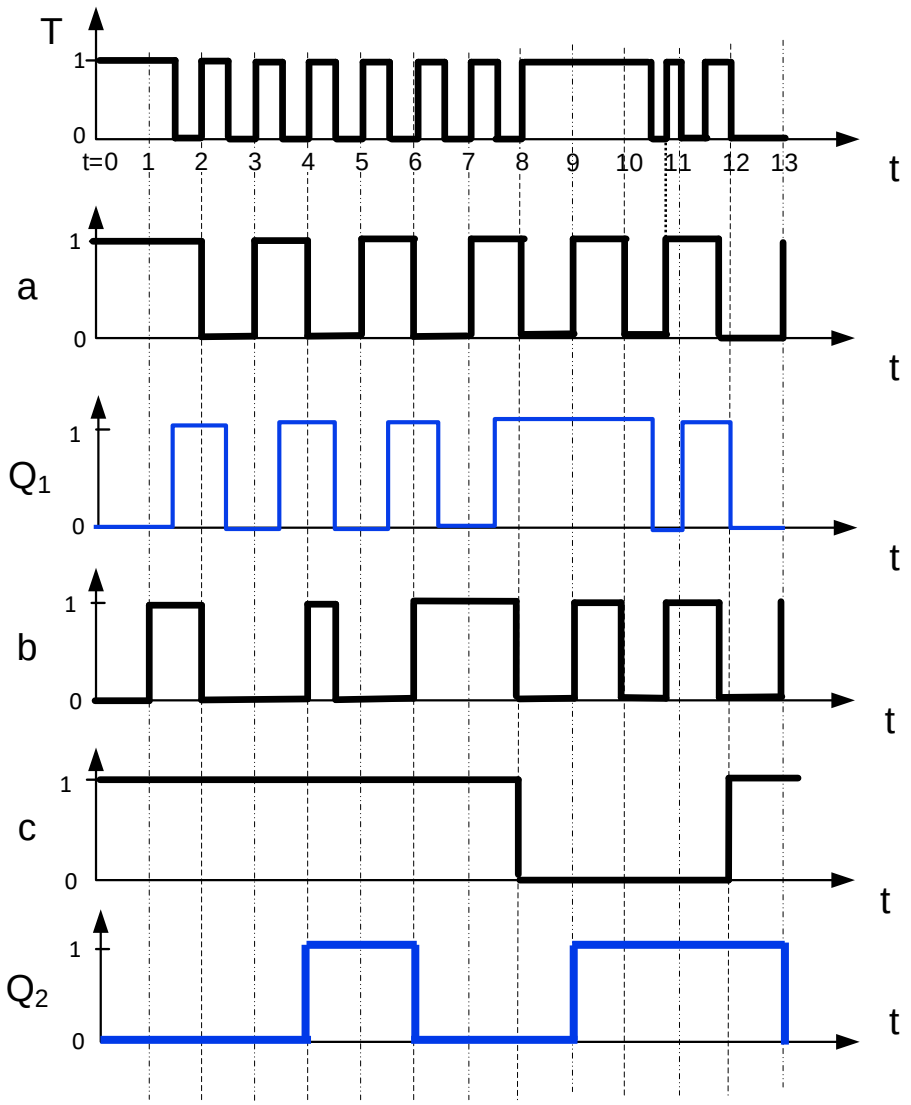


Bild 5.1: *MS-FF*

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.



Aufgabe 6: MMIX Rechner

Gegeben sei der nachfolgende Algorithmus sowie ein Ausschnitt der MMIX-Code-Tabelle (Bild 6.1) und eines Registerspeichers (Bild 6.2):

	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	

Bild 6.1: *MMIX-Code-Tabelle*

Algorithmus (Dezimal): $\frac{a}{c-512*b+5}$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 01 00	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 05	Zwischenergebnis
$0x8B	0x00 00 00 00 00 00 61 0B	Nicht veränderbar
…	…	

Bild 6.2: *Registerspeicher*

a) Im Registerspeicher eines MMIX-Rechners befinden sich zu Beginn die in Bild 6.2 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 6.1 umrahmten Befehlsbereiche. Speichern Sie die Zwischenergebnisse nach jedem Befehl des Algorithmus in der Registerzelle mit dem Kommentar *Zwischenergebnis*.

1	MULI $0x8A $0x86 0x02
2	MUL $0x8A $0x88 $0x8A
3	SUB $0x8A $0x89 $0x8A
4	ADDI $0x8A $0x8A 0x05
5	DIV $0x8A $0x87 $0x8A

Alternative Lösungen für 1. Zeile:

a) ADD $0x8A $0x86 $0x86

b) SETL $0x8A 0x0200

Alternative Lösung für 2. Zeile:

MUL $0x8A $0x8A $0x88

b) Angenommen Ihr Zwischenergebnis sei 0x0...00 12 34 56. Sie speichern es mit dem Befehl STT $0x8A $0x8B $0x86 in den Datenspeicher (Bild 6.3). Die Byte-Reihenfolge ist mit big endian festgelegt. Geben Sie für die vier Eingangsbusse am Datenspeicher an, welcher Wert dort jeweils anliegt. Tragen Sie die durch den Speicherbefehl geänderten Werte in den Datenspeicher ein.

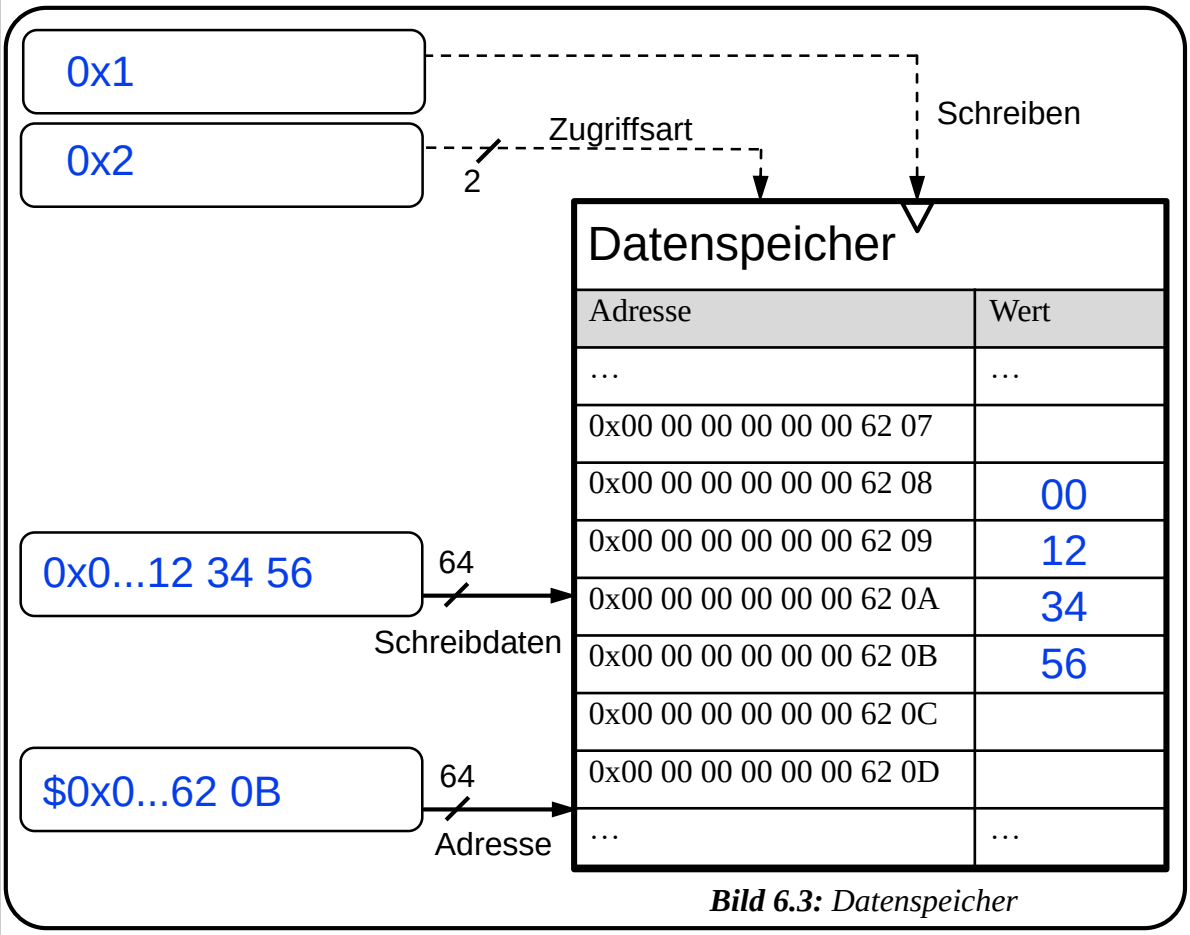


Bild 6.3: *Datenspeicher*

c) Geben Sie den Befehl LDOI $0x88 $0x8B 0xFF in Maschinensprache an.

0x8D 88 8B FF

Aufgabe 7: Scheduling

Auf einem Einprozessorsystem sollen die nachfolgend angegebenen Tasks zum Bohren von Löchern ablaufen. Beantworten Sie hierzu die folgenden Fragen.

Hinweis: Die Tabelle enthält alle wichtigen Angaben zur Planung der Tasks A,B,C. Die Ausführungsdauer ist eine relative Zeitangabe, wohingegen Bereit und Deadline als absolute Zeitangaben zu verstehen sind. Bei der Prioritätsangabe besitzt der Task mit dem niedrigsten nummerischen Wert die höchste Priorität. Geben Sie, wenn gefragt, die Reihenfolge der ablaufenden Tasks an. z.B. ABCABB

Bezeichnung	**Task**	Ausführungsdauer	Bereit	Deadline	Priorität
Loch_A	**A**	3s	4s	8s	3
Loch_B	**B**	5s	0s	9s	1
Loch_C	**C**	6s	5s	13s	2

1. Geben Sie die Reihenfolge der Tasks und den Zeitpunkt an, an dem die Ausführung des letzten Tasks startet. Als Scheduling-Verfahren ist das First-In-First-Out (FIFO) Scheduling einzusetzen.
2. Würde der Task B bei einem präemptiven Least-Laxity Scheduling (LL) von Task A zum Zeitpunkt t = 4s unterbrochen werden? Begründen Sie Ihre Antwort stichpunktartig.
3. Wird bei einem nicht-präemptiven Earliest Deadline First (EDF) Scheduling die Bedingung der Rechtzeitigkeit aller Tasks A,B,C erfüllt? Begründen Sie Ihre Antwort stichpunktartig.
4. Inwiefern unterscheidet sich die Reihenfolge der Tasks bei einem nicht-präemptiven Earliest Deadline First (EDF) von der des First-In-First-Out (FIFO) Scheduling?

1. BAC, 8s

2. Ja, da der Spielraum (Laxity) von Task A (1s) kleiner ist als der von Task B (4s).

3. Nein, da Task C erst eine Sekunde nach Ablauf der Deadline abgeschlossen ist

4. Sie sind identisch, kein Unterschied vorhanden.

8. Semaphoren

Gegeben seien die folgenden vier Tasks T1 bis T4 sowie die Semaphoren S1 bis S4 (Bild 8.1). Die Startwerte der Semaphoren entnehmen Sie der Antworttabelle (Bild 8.2).

Vervollständigen Sie die fünf Zeilen 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. Welchen Ablauf erhalten Sie? Welche Art Verklemmung tritt auf?

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

Bild 8.1: *Semaphorenzuweisung*

Task	S1	S2	S3	S4
-	0	3	0	0
T2	0	1	1	0
T3	2	0	0	0
T1	1	0	1	1
T1	0	0	2	2
T4	2	0	2	0

Bild 8.2: *Antworttabelle*

Ablauf der Tasks: T2, T3, T1, T1, T4

Art der Verklemmung: Partielle Verklemmung / Livelock

9. Echtzeitbetriebssysteme

Eine Bohrmaschine hat einen Task „Bohren_Mit_Metalldetektor“, welcher kontinuierlich ausgeführt wird. Erkennt der induktive Sensor in der Bohrspitze Metall, so wird der Task in den Zustand „ruhend“ (RTOS-UH-Taktzustandsdiagramm) versetzt. Ergänzen Sie den nachfolgenden Programmausschnitt mittels PEARL.

```
// Definiere Task „Bohren_Mit_Metalldetektor“ mit Priorität 2

Bohren_Mit_Metalldetektor : TASK PRIORITY 2;

  // Spezifiziere Unterbrechung „MetallErkannt“

  SPC MetallErkannt INTERRUPT ;

  // Aktiviere Unterbrechung „MetallErkannt“:

  ENABLE MetallErkannt;

  ALL 20s ACTIVATE Bohren_Mit_Metalldetektor;

  // Beende Task Bohren_mit_Metalldetektor

  WHEN MetallErkannt TERMINATE Bohren_Mit_Metalldetektor;

END;
```

10. Bussysteme und Prozessperipherie

Beantworten Sie die nachfolgenden Fragen zu Bussystemen und Prozessperipherie. Hinweis: Bei Multiple-Choice Fragen ist jeweils nur eine der Antworten korrekt.

Bei welcher Netzwerktopologie…

…bleibt die Kommunikation zwischen zwei Knoten trotz Ausfall eines **beliebigen** dritten Knotens unbeeinflusst?

Bus (**x**) Linie ( ) Ring ( ) Stern ( )

… kann das Bussignal zwischen zwei Knoten durch benachbarte Knoten verstärkt werden?

Bus ( ) Linie (**x**) Stern ( )

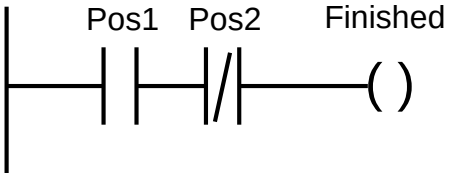
Nennen Sie ein Beispiel für ein Buszugriffsverfahren, das dezentral gesteuert ist und z.B. in den Technologien FlexRay, TTP oder TTCAN eingesetzt wird.

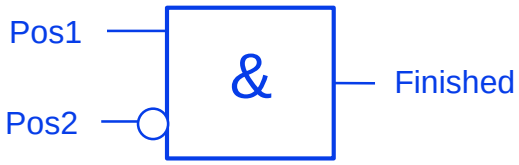
Token Passing / Token Ring / TDMA (Time division multiple access)

Aufgabe 11: IEC 61131-3

a) Gegeben ist nebenstehender Programmausschnitt in IEC 61131-3-Kontaktplan (KOP).

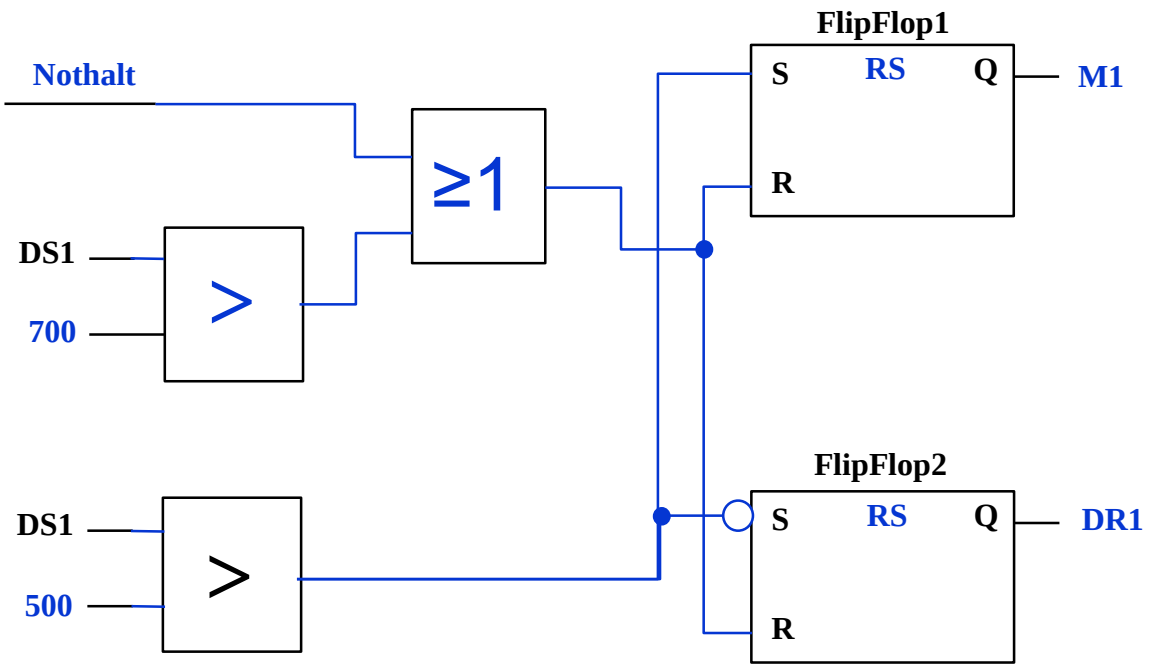
Überführen Sie den Codeausschnitt in IEC 61131-3-Funktionsbausteinsprache (FBS).



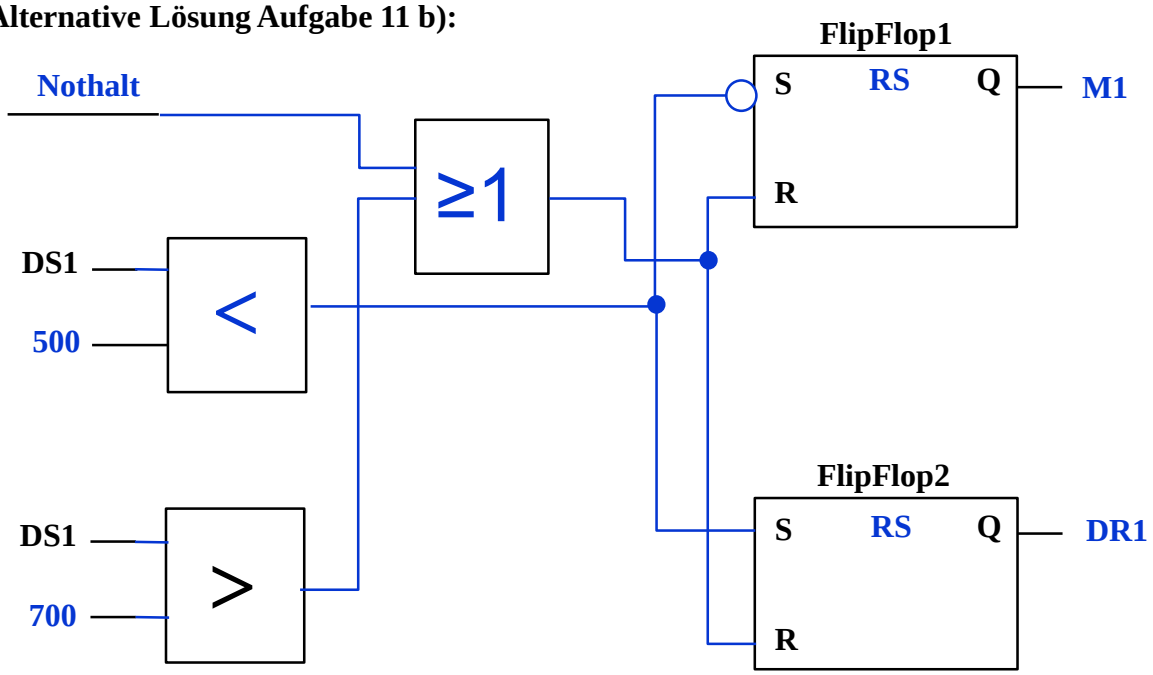


b) Gegeben ist eine Ansteuerung des Bohrmotors in Abhängigkeit des Bohrwiderstands gemessen mit einem Drucksensor (DS1) am Ende des Bohraufsatzes:
Der Bohrer besitzt zwei Aktoren: Einen Motor (M1) und eine Drosselung zur Reduktion der Geschwindigkeit für den sicheren Betrieb. Der ideale Gegendruck für die Bohrung liegt zwischen 500 und 700 bar. Liegt der Gegendruck unter 500, wird zunächst die Drosselung (DR1) eingeschaltet. Ab einem Druck von 500 startet der Motor. Ab einem Druck von 700 ist das Bohren nicht mehr zulässig und sowohl Motor als auch Drosselung müssen abgeschaltet werden.
Wird der Nothalt betätigt (Nothalt = 1), sollen ebenfalls sowohl der Motor als auch die Drosselung unverzüglich ausgeschaltet werden und aus bleiben.

Vervollständigen Sie im Folgenden die Ansteuerung des Bohrers in IEC 61131-3-Funktionsbausteinsprache (FBS).
Hinweis: Signalverzögerungen im System sind zu vernachlässigen. Verwenden Sie keine Schaltglieder außer den in der Vorlage bereits vorhandenen. Ergänzen Sie Negationen falls notwendig.



Alternative Lösung Aufgabe 11 b):

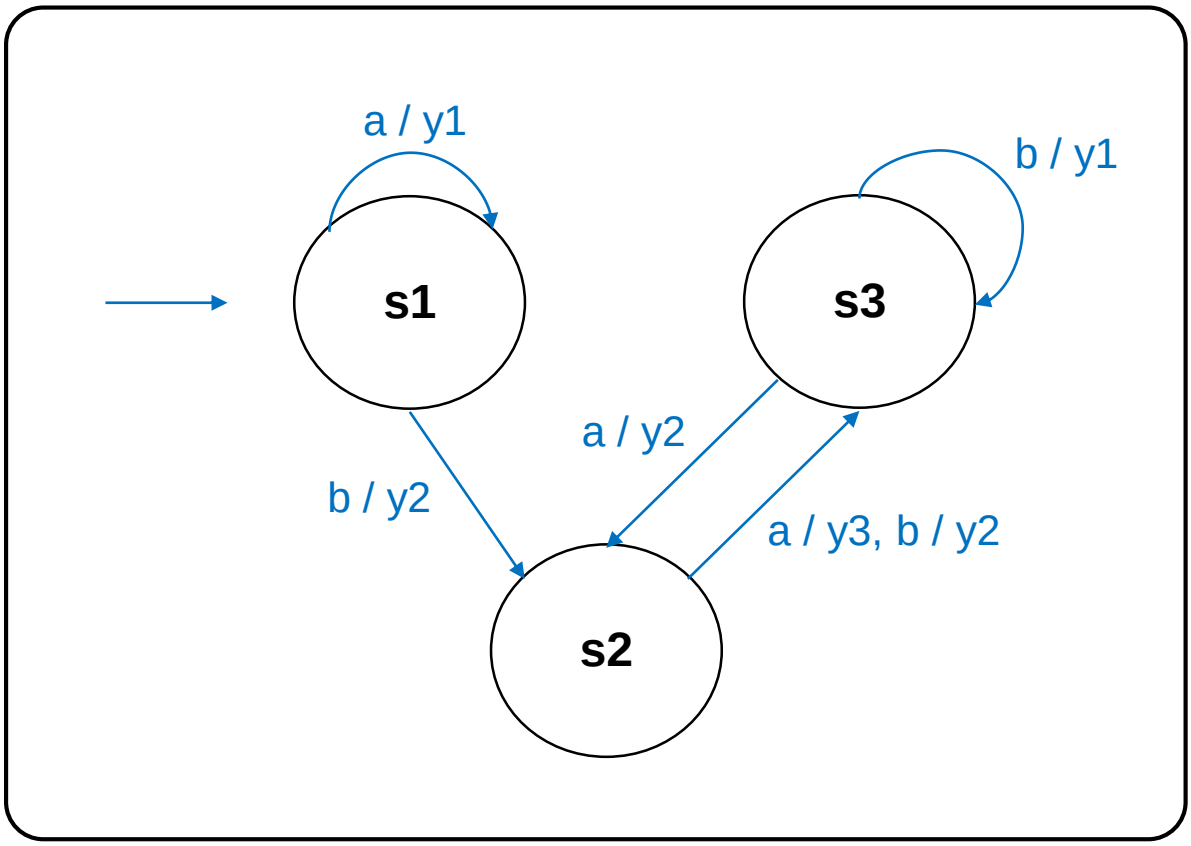


Aufgabe 12: Automaten

Gegeben ist folgende Übergangstabelle:

T \ S	s1	s2	s3
a	s1, y1	s3, y3	s2, y2
b	s2, y2	s3, y2	s3, y1

a) Zeichnen Sie den entsprechenden Übergangsgraphen. Der Startzustand ist s1.



b) Kreuzen Sie an, welche Ausgabe durch die folgende Eingabe erzeugt wird: a b a b a

☐ y1 y2 y2 y1 y2 ☒ y1 y2 y3 y1 y2 ☐ y1 y2 y3 y2 y2

Aufgabe 13: Grundlagen

a) Initialisieren Sie ein Array `Zahlen` vom Datentyp `float` mit Speicherplatz für zwei Elemente. Weisen Sie dem ersten Speicherplatz den Wert 2.3 zu und geben Sie diesen anschließend im Format `2.30` über das Terminal aus.

```
float Zahlen[2];
Zahlen[0] = 2.3;
printf("%.2f", Zahlen[0]);
```

b) Programmieren Sie eine Funktion `pow3`, welche die dritte Potenz eines Eingabeparameters vom Typ `int` berechnet und als `int` zurückgibt.

```
int pow3 (int in)
{
    return in * in * in;
}
```

c) Nachfolgend finden Sie eine Schleife, die für den Laufindex `i` die Werte 1 bis 1000 durchläuft. Ergänzen Sie an der richtigen Stelle Code innerhalb der Schleife, um die vorgegebene Funktion `writeToFile(i)` nur bei den Werten `i=3,13,23,33,43,…` aufzurufen.

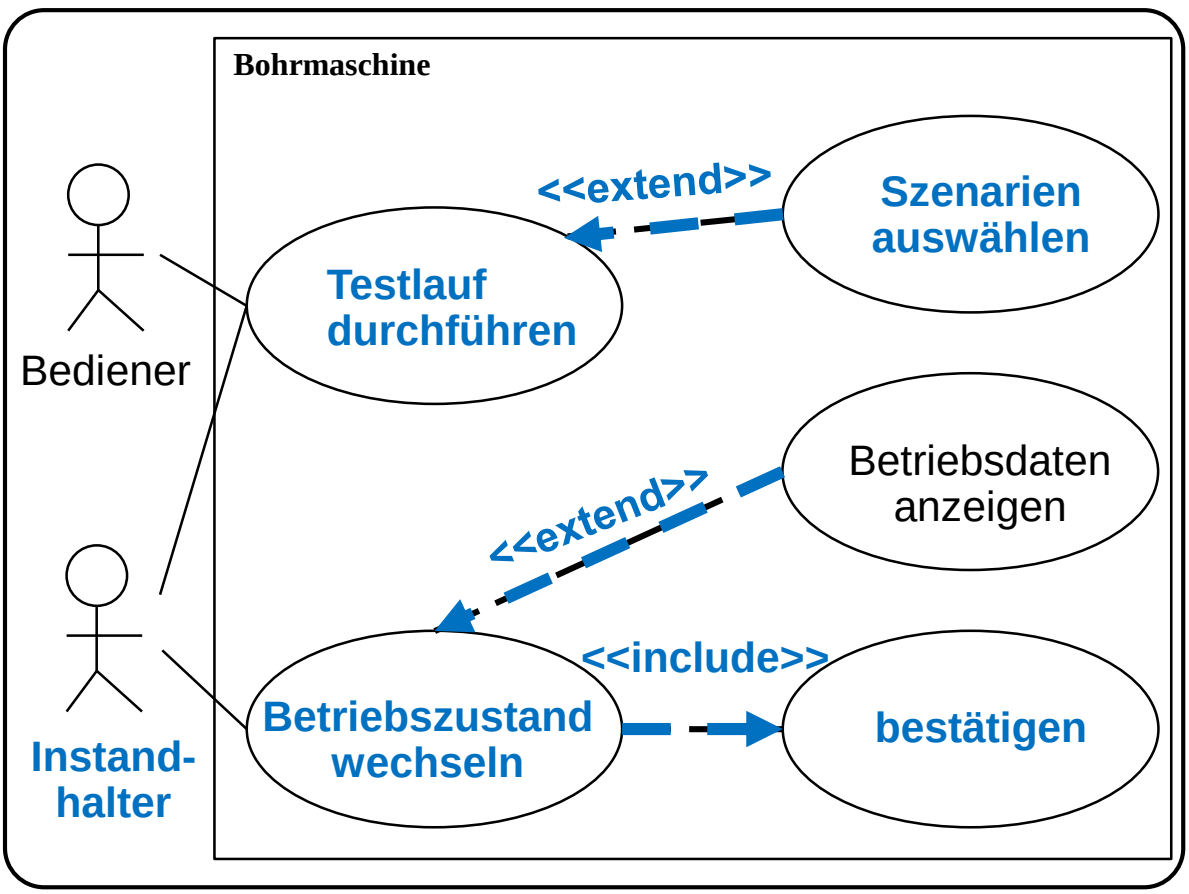
```
int i = 1;
while (i<=1000)
{
    if ( !((i-3) % 10))
    {
        writeToFile(i);
    }
    i++;
}
```

Aufgabe 14: Use Case Diagramm

Erstellen Sie mithilfe der folgenden Angaben ein Use-Case-Diagramm der UML. Benennen Sie die Akteure sowie die Anwendungsfälle und ergänzen Sie die Pfeile der Beziehungen.

Der *Instandhalter* eines Trockendrehbohrers (Bild 14.1) muss die Hydraulik der Maschine regelmäßig warten. Um die Klappe zur Hydraulik öffnen zu können, *wechselt* der Instandhalter den *Betriebszustand* der Maschine zu ‚Instandhaltung'. Hierzu muss er *bestätigen*, dass sich keine Personen im Gefahrenbereich der Maschine befinden. Zudem kann er sich die aktuellen *Betriebsdaten* (Betriebsstunden etc.) *anzeigen* lassen. Nach der Wartung *führt* der Instandhalter gemeinsam mit dem *Bediener* einen kurzen *Testlauf durch*. Hierzu kann aus vorprogrammierten *Szenarien ausgewählt* werden.

Bild 14.1:
Trockendrehbohrer

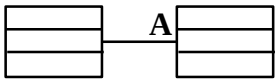
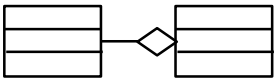


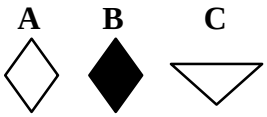
Aufgabe 15: Klassendiagramm

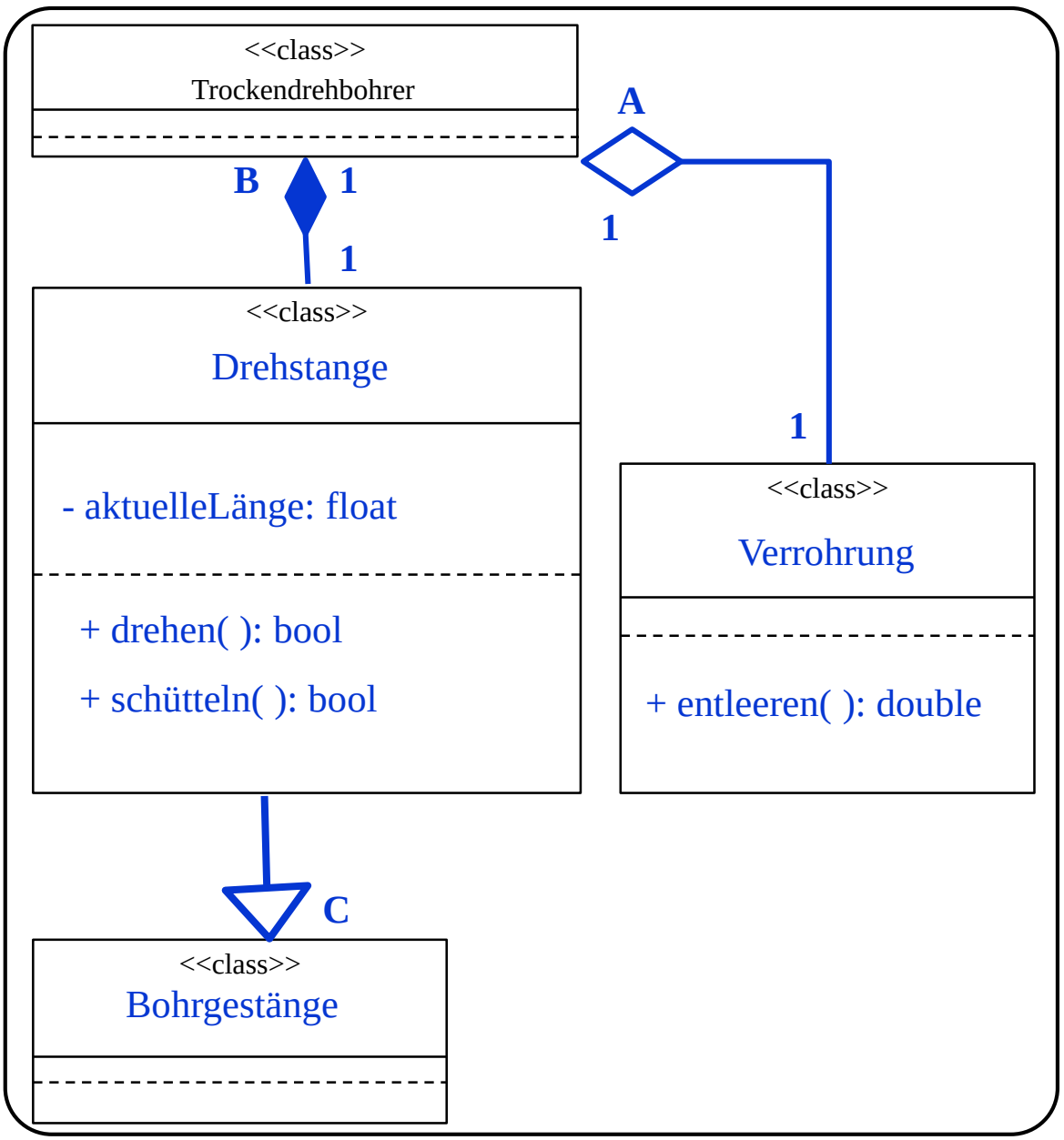
Ein *Trockendrehbohrer* besteht immer aus einer teleskopierbaren *Drehstange*. Eine Drehstange ist ein *Bohrgestänge*, das Bohren durch Drehung ermöglicht. Ihre *aktuelle Länge* wird als float erfasst. Die Drehstange kann *drehen* und *schütteln*, die jeweiligen Funktionen liefern als bool zurück, ob sie erfolgreich aktiviert wurden. Bei nicht ausreichend standfesten Böden ist eine verrohrte Bohrung notwendig. Der Trockendrehbohrer wird dafür bei Bedarf mit einer *Verrohrung* ausgestattet, die sich *entleeren* kann, wobei sie die Menge an entleertem Material als double ausgibt.

Vervollständigen Sie das Klassendiagramm mithilfe der obenstehenden Informationen. Achten Sie auf die Kardinalitäten und die Sichtbarkeiten von Attributen und Methoden.

Falls Zeichnen nicht möglich: Schreiben Sie den Buchstaben des Elements an die Position, an die Sie das Element einzeichnen würden.

Beispiel:  für 





Aufgabe 16: Klassendiagramm zu Code

Sie möchten eine Software zur Steuerung des Trockendrehbohrers schreiben. Bild 16.1 zeigt das vereinfachte UML-Klassendiagramm für die zu implementierende Software.

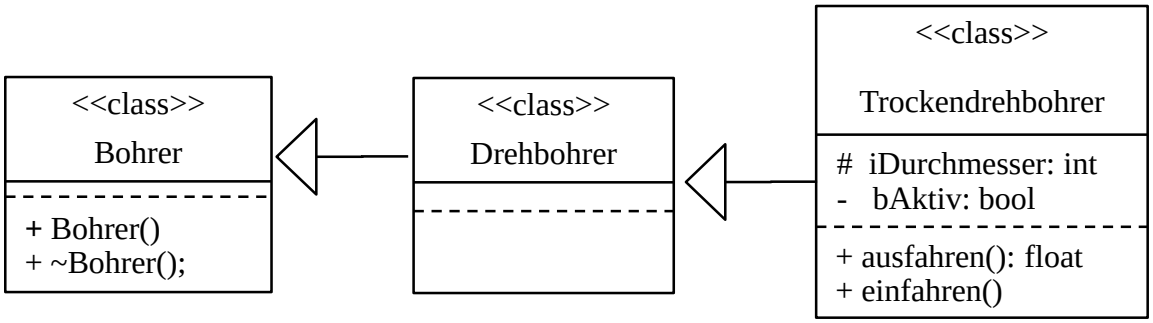


Bild 16.1: Klassendiagramm für Aufgabe 16

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

```
class Bohrer {
_____________________________________________________
public:
_____________________________________________________
  Bohrer();
_____________________________________________________
  ~Bohrer();
_____________________________________________________

_____________________________________________________
};

class Trockendrehbohrer ___: public Drehbohrer_____ {
_____________________________________________________
 protected:
_____________________________________________________
   int iDurchmesser;
_____________________________________________________
 private:
_____________________________________________________
   bool bAktiv;
_____________________________________________________
 public:
_____________________________________________________
   float ausfahren();
_____________________________________________________
   void einfahren();
_____________________________________________________

_____________________________________________________

};
```

Aufgabe 17: Zustandsdiagramm

Im Folgenden wird die Erstellung einer verrohrten Bohrung mittels eines Trockendrehbohrers betrachtet. In diesem Verfahren wird zunächst ein Rohr in den Boden gebohrt und anschließend die im Inneren liegende Erde in einer zweiten Bohrung abgetragen. Bild 17.1 zeigt den vereinfachten Aufbau des Trockendrehbohrers mit der für die Aufgabe relevanten Sensorik und Aktorik. Zur Locherstellung schwenkt der Ausleger zur Bohrposition und beginnt nach einer einzuhaltenden Abschwingzeit (5 Sekunden) mit dem ersten Bohrvorgang. Zwischen beiden Bohrvorgängen muss das Werkzeug am Ausleger gewechselt werden, um mit der zweiten Bohrung fortzufahren. Nach Beendigung des Vorgangs schwenkt der Ausleger in die Fahrposition, um eine sichere Fahrt zum nächsten Einsatzort zu gewährleisten. Die Löcher sollen 10 Meter tief gebohrt werden.

Der Trockendrehbohrer besitzt folgende Funktionen: Die Operation **`fahren(int Position)`** schwenkt den Ausleger zwischen den drei Positionen Rüsten mit dem Wert **`1`**, Bohren mit dem Wert `2` und Fahren mit dem Wert **`3`.** Der Bohrvorgang wird mittels der Funktion **`bohren()`** durchgeführt. Die Funktion **`ruesten()`** versetzt den Bohrer in einen sicheren Zustand (Bohrwelle gesichert, Werkzeug entriegelt), in dem das Werkzeug umgerüstet werden kann. Dies ist zwingend notwendig vor dem Austausch eines Werkzeugs.

Eine Funktion, die den Bohrer aus dem Loch fahren lässt, existiert noch nicht und muss mittels direktem Setzen von Aktorvariablen erfolgen. Hierbei kann die Aktorvariable **`a.Bohrwelle`** mit den Werten **`1`** (zum Hochfahren der Welle), **`2`** (zum Herunterfahren der Welle) und **`0`** (zum Stoppen der Welle) gesetzt werden.

Zur Erkennung des aktuell montierten Werkzeugs besitzt der Bohrer einen Sensor (**`s.Werkzeug`**), der zwischen Rohr (**`1`**) und Bohrer (**`2`**) unterscheiden kann. Ist kein Werkzeug montiert, so gibt der Sensor den Wert **`NULL`** zurück. Der Sensor **`s.Bohrwelle`** gibt in Metern (vorzeichenlosen Ganzzahlen) abgerundet an, wie tief der Bohrer sich in die Erde befindet. Der Sensor **`s.Ausleger`** erkennt die Position des Auslegers und liefert Ganzzahlen zurück, wie im Bild beschrieben. Der Aktor **`a.Ausleger`** schwenkt den Ausleger nach rechts (1) links (2) oder stoppt ihn (0).

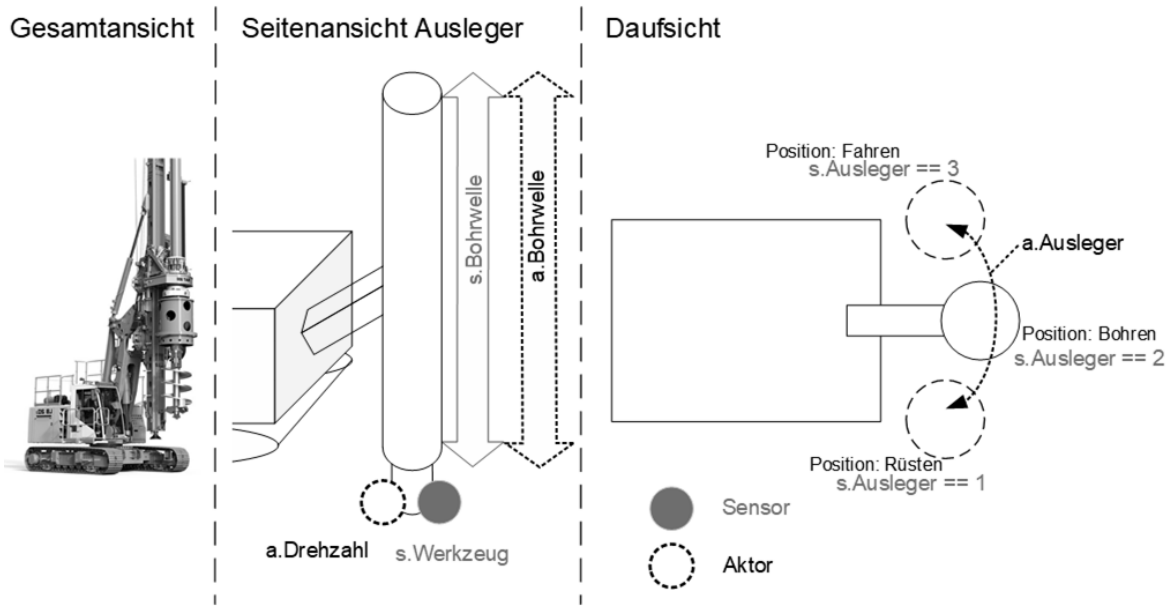


Bild 17.1: Skizze des Trockendrehbohrers

Vorgegeben ist das in Bild 17.2 gezeigte Zustandsdiagramm mit den vorgegebenen Zustandsnummern. Füllen Sie die durch römische Ziffern gekennzeichneten Lücken im Lösungsfeld bzw. beantworten Sie die Fragen. Für zeitbezogene Wächterbedingungen kann der Term [nach X Sekunden] verwendet werden, wobei X für die vergangene Zeit steht.

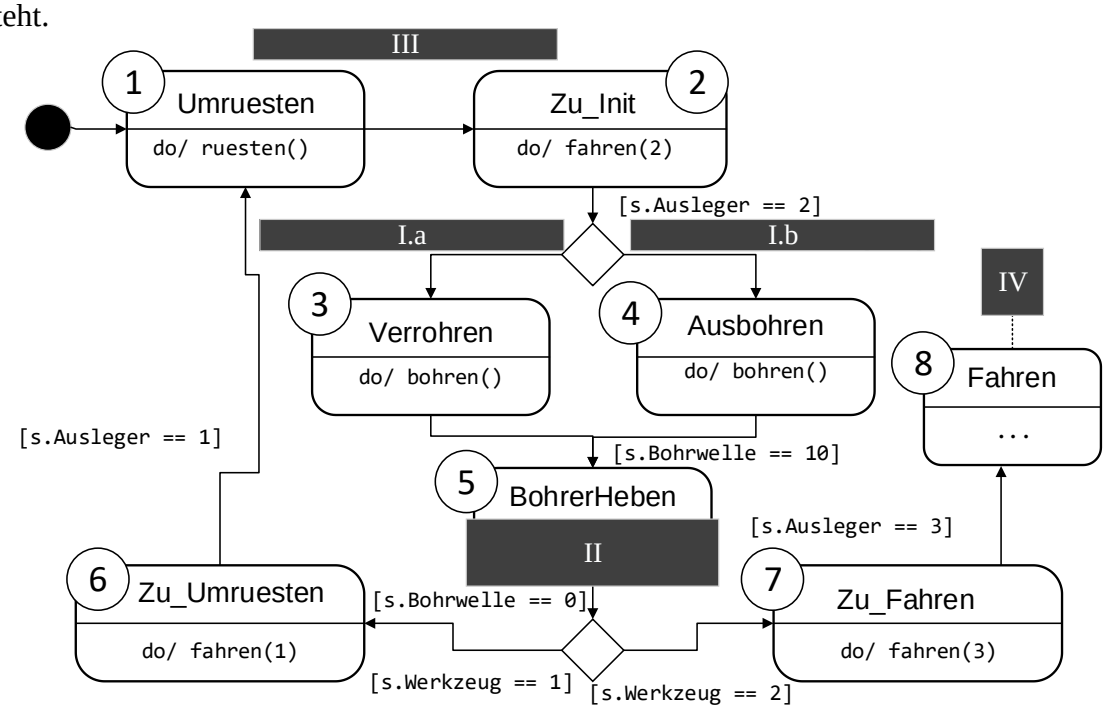


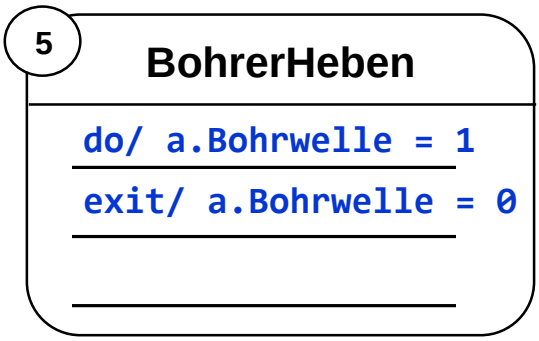
Bild 17.2: Unvollständiges Zustandsdiagramm des Betriebsablaufs

I. Geben Sie die korrekte Wächterbedingungen an:

I a. `[s.Werkzeug == 1 && nach 5 Sekunden]`

I b. `[s.Werkzeug == 2 && nach 5 Sekunden]`

II. Modellieren Sie den Zustand `BohrerHeben` korrekt aus:



III. Geben Sie die korrekte Wächterbedingung an:

`[s.Werkzeug == 1 | | s.Werkzeug == 2]`

IV. Welcher Zustand muss nach Ankunft am nächsten Einsatzort folgen (nach Zustand (8)), um mit dem nächsten Loch fortzufahren:

`6`

Aufgabe 18: Zustandsdiagramm zu C-Code

Sie haben nun die Aufgabe, Teile des in Aufgabe 17 modellierten Zustandsdiagramms in Form von C-Code zu implementieren. Hierfür stehen Ihnen die in Tabelle 18.1 dargestellten bereits implementierten Funktionen zur Interaktion mit dem Trockendrehbohrer, die Ein- und Ausgänge sowie erweiterten Variablen zur Verfügung.

Tabelle 18.1: Sensor- und Aktorvariablen der Antriebsstation, sowie erweiterte Variablen und bereits implementierte Funktionen

Typ	Name	Beschreibung
FUNKTIONEN	`fahren(int Position)`	Fährt den Ausleger zur gewünschten Position. Eingabewerte sind 1 (Rüsten), 2 (Bohren), 3 (Fahren).
	`bohrerHeben()`	Hebt den Bohrer vertikal aus dem Bohrloch heraus.
	`bohren()`	Startet den Bohrvorgang (senken des Auslegers und Regelung der Drehzahl)
	`ruesten()`	Versetzt den Bohrer in einen sicheren Zustand, um das Werkzeug zu tauschen.
	`lSensoren (unsigned int *zeit, SENSOREN *s)`	Einlesen der aktuellen Sensorwerte und Zuweisung dieser auf die Variable s. Aktualisiert die Laufzeit des Programms über einen Zeiger auf die Zeitvariable.
	`sAktoren (AKTOREN *a)`	Überträgt die Werte der Aktoren in Variable a an die gesteuerten Ausgänge.
SENSOREN / AKTOREN	`s.Werkzeug`	Besitzt je nach montiertem Werkzeug die Werte: 1 (Rohr), 2 (Bohrer). Ist kein Werkzeug montiert besitzt es den Wert NULL.
	`s.Bohrwelle`	Gibt die aktuelle Tiefe der Bohrwelle in Metern (m) an. Werte in vorzeichenlosen Ganzzahlen.
	`s.Ausleger`	Gibt die Position des Auslegers an. Kann folgende Werte annehmen: 1 (Rüsten), 2 (Bohren), 3 (Fahren).
	`a.Bohrwelle`	Fährt die Bohrwelle hoch (1), runter (2) oder stoppt sie (0)
	`a.Ausleger`	Fährt den Ausleger nach rechts (1), links (2) oder stoppt diesen (0)
	`a.Drehzahl`	Steuert die Drehzahl der Bohrspitze mittels vorzeichenloser Ganzzahlen im Bereich 0 bis 1000
VARIABLEN	`vplcZeit`	Aktuelle Laufzeit des Programms in ms (positive Ganzzahl).
	`t`	Variable für timer-Programmierung (positive Ganzzahl).
	`zustand`	Aktueller Zustand des Zustandsautomaten (Ganzzahl), welcher ausgeführt wird.

a) Programmgrundgerüst

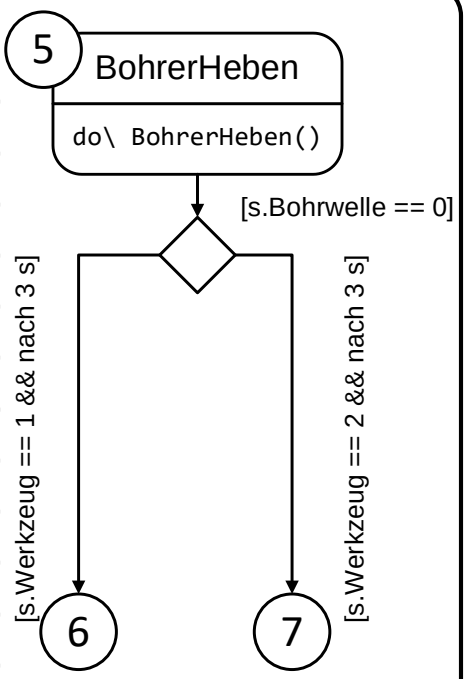
Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst, um eine zyklische Ausführung des Zustandsautomaten zu ermöglichen. Verwenden Sie hierfür die in Tabelle 18.1 angegebenen Variablennamen, da diese in anderen Programmteilen so verwendet werden sollen. Zur Interaktion mit der Hardware verwenden Sie die ebenfalls in Tabelle 18.1 gegebenen Funktionen, welche im Header *Trockendrehbohrer.h* bereits definiert und implementiert sind. Der Platzhalter `/* ZUSTAENDE */` soll später durch den spezifischen Code der Zustände in Form eines Zustandsautomaten ersetzt werden.

```
#include „Trockendrehbohrer.h" // Header einbinden

SENSOREN s;                   // Sensorik
AKTOREN a;                    // Aktorik
unsigned int t = 0;           // Timer-Programmierung

int main()
{  // Deklarationen
   unsigned int vplcZeit;// Zeitvariable
   int zustand;          // Schrittvariable

    while(1)             // Zyklische Ausführung
   {
         // Einlesen von Hardware
         lSensoren(&vplcZeit, &s)      ;
         switch(zustand)
         // Zustandsautomat
         {
                  /* ZUSTAENDE */
         }
          // Schreiben auf Hardware
         sAktoren(&a);

   }
   return 1;      // Wird nicht erreicht
}
```

b) Implementierung des Zustands „`BohrerHeben`“

Der Zustand „`BohrerHeben`“ wurde im Folgenden um eine Wartezeit erweitert und soll nun implementiert werden. Der Rumpf des Falls (`case 5`) ist bereits vorgegeben. Ergänzen Sie den Code, so dass der Zustand wie im Zustandsdiagramm korrekt umgesetzt wird. Verwenden Sie hierfür die Variablen, sowie die Ein- und Ausgänge aus Tabelle 18.1. Beachten Sie, dass mehr Leerzeilen zur Verfügung stehen, als für die korrekte Antwort erforderlich sind.

```
case 5:   // BohrerHeben
  bohrerHeben();
  _____________________________________
  _____________________________________
  if (s.Bohrwelle == 0)
  _____________________________________
  {
  _____________________________________
    if (t==0)
  _____________________________________
    {
  _____________________________________
      t = vplcZeit + 3000;
  _____________________________________
    }
  _____________________________________
    else if ((vplcZeit >= t)
  _____________________________________
    && (s.Werkzeug == 1))
  _____________________________________
    {
  _____________________________________
      t = 0;
  _____________________________________
      schritt = 6;
  _____________________________________
    }
  _____________________________________
    else if ((vplcZeit >= t)
  _____________________________________
    && (s.Werkzeug == 2))
  _____________________________________
    {
  _____________________________________
      t = 0;
  _____________________________________
      schritt = 7;
  _____________________________________
    }
  _____________________________________
  }
  _____________________________________
  _____________________________________
  _____________________________________
  _____________________________________
  _____________________________________
  _____________________________________
  break;
```

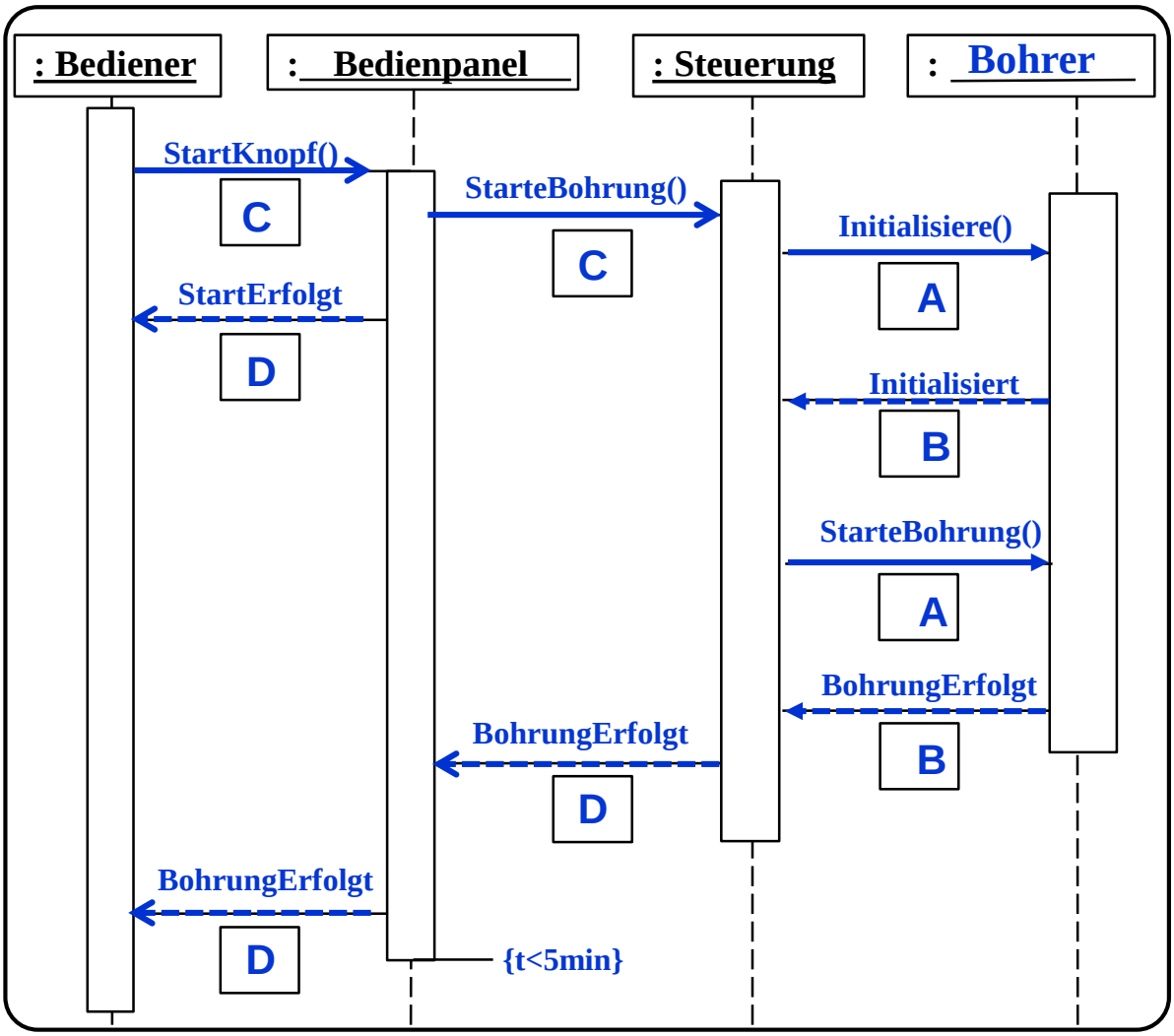


Aufgabe 19: Sequenzdiagramm zum Use-Case *Bohren*

Der Bediener betätigt das Bedienpanel durch den *StartKnopf()* und kann anschließend weitere Prozesse ausführen. Über das Bedienpanel wird daraufhin die Anweisung *StarteBohrung()* an die Steuerung übermittelt und anschließend ein Signal an den Bediener gesendet, dass die Bohrung gestartet wurde (*StartErfolgt*). Nachdem die Steuerung die Anweisung vom Bedienpanel erhalten hat, startet die Steuerung die Initialisierung des Bohrers (*Initialisiere()*). Nachdem der Bohrer die erfolgreiche Initalisierung (*Initialisiert*) an die Steuerung zurückgemeldet hat, übermittelt die Steuerung dem Bohrer das Signal zum Start der Bohrung (*StarteBohrung()*). Der Bohrer sendet der Steuerung ein Signal zurück, sobald die Bohrung abgeschlossen ist (*BohrungErfolgt*). Während der Initalisierung und während des Bohrvorgangs kann die Steuerung keine weiteren Prozesse auslösen. Nachdem die Steuerung vom Bohrer die Information über die erfolgte Bohrung erhalten hat, meldet sie das Signal (*BohrungErfolgt*) an das Bedienpanel, das die Information (*BohrungErfolgt*) anschließend dem Bediener anzeigt. Spätestens fünf Minuten nach Drücken des Startknopfs wechselt das Bedienpanel in den Ruhezustand. Die Bohrung ist bis dahin bereits erfolgt und vollständig abgeschlossen.

Ergänzen Sie das Sequenzdiagramm entsprechend der Beschreibung. Geben Sie im Kasten unter den Linien zwischen den Objekten den jeweiligen Nachrichtentyp an:

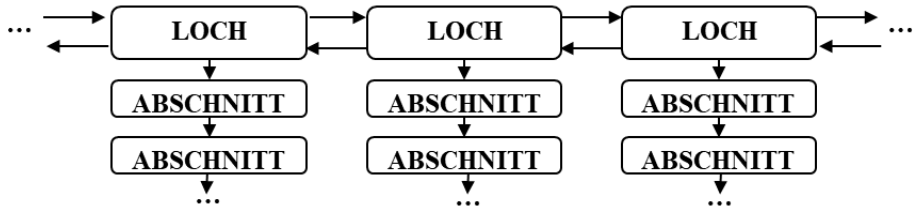
A: ——→ **B:** ←- - - - **C:** ——→ **D:** ←- - - -
E: ←—— **F:** - - - -→ **G:** ←—— **H:** - - - -→



Aufgabe 20: Algorithmen und Datenstrukturen

In dieser Aufgabe soll ein digitaler Zwilling für die in Aufgabe 17 gebohrten Löcher realisiert werden. Dabei handelt es sich um ein Datenelement, in welchem Eigenschaften des gesamten Bohrloches gespeichert werden. Jedes einzelne Loch wird schrittweise in mehreren Abschnitten gebohrt. Für jeden Bohrabschnitt wird die Bodenbeschaffenheit sensorisch erfasst und gespeichert.

Sie entscheiden sich dazu, die Bohrlöcher in einer **doppelt verketteten Liste** zu speichern, wobei jedes dieser Listenelemente gleichzeitig als Listenkopf für eine angehängte **einfach verkettete Liste** mit den jeweiligen Bohrabschnitten dient. Die Zeiger **`prev`**, **`pnext`** verketten die doppelt verketteten Listenelemente; der Zeiger **`down`** die einfach verketteten Elemente.



In einem Listenelement vom **Typ LOCH** soll gespeichert werden: Durchschnittliche Dichte (`Dichte`) der Erde im Loch als vorzeichenlose 32-bit Ganzzahl

In jedem Listenelement vom **Typ ABSCHNITT** sollen gespeichert werden: Masse (`Masse`) und Volumen (`Volumen`) der im jeweiligen Bohrabschnitt enthaltenen Erde als vorzeichenlose 16-bit Ganzzahlen

Der **Typ BEGINN** soll einen Zeiger auf den Anfang der doppelt verketteten Liste enthalten.

a) Definition der Listenelemente: Vervollständigen Sie die folgenden Lösungskästchen gemäß der obigen Angaben und der obigen Abbildung.

```
typedef struct ABSCHNITT {
     struct__ABSCHNITT*_____ down;
      unsigned short Masse;
      unsigned short Volumen;
}ABSCHNITT;
```

```
typedef struct LOCH {
     __struct LOCH*_____ prev;
     __struct LOCH*_____ pnext;
     struct__ABSCHNITT*_____ down;
      unsigned long/int Dichte;
}LOCH;
```

```
typedef struct BEGINN{
     struct__LOCH*_____ first;
}BEGINN;
```

b) Schreiben Sie eine Funktion zur Berechnung der durchschnittlichen Dichte der Erde in einem Bohrloch:

In jedem Listenelement vom **Typ LOCH** soll die durchschnittliche Dichte der Erde entlang der zugehörigen Bohrabschnitte gespeichert werden. Schreiben Sie hierzu eine Funktion `(Dichteberechnung)`, welche die durchschnittliche Dichte entlang dieser Bohrabschnitte von **Typ ABSCHNITT** berechnet. Hierzu müssen Sie die Massen und Volumina aller zugehörigen Bohrabschnitte addieren und final dividieren.

Beachten Sie dabei, dass die Funktion `(Dichteberechnung)` die berechnete Dichte als vorzeichenlose 32-bit Ganzzahl in der **Einheit Kg/m^3** ausgeben soll. In jedem Listenelement vom **Typ Abschnitt** ist das **Volumen in der Einheit m^3** und die **Masse in der Einheit t (Tonnen)** gespeichert**.**

Sie müssen die berechnete Dichte in der Einheit Kg/m^3 nicht runden: Schneiden Sie mögliche Nachkommastellen des Rückgabewertes einfach ab. Vervollständigen Sie die Funktion gemäß der obigen Angaben.

```
unsigned long/int Dichteberechnung (LOCH *data)
{
      // Initialisierung
      float Dichte;
      unsigned int Masse = 0;
      unsigned int Volumen = 0;

      // Volumina und Massen im Bohrloch addieren
      while(  data->down != NULL          ){
            Volumen += data->down->Volumen;
            Masse += data->down->Masse;
            data->down = data->down->down;
      }

      // Dichte berechnen
      Dichte = Masse/(float) Volumen;

      // Rückgabewert: Dichte in Kg/m^3
      Dichte = Dichte * 1000;

      // Rückgabewert: Dichte in Kg/m^3
      return (unsigned long/int) Dichte;
}
```

c) Erstellen Sie eine Funktion zum Löschen eines Bohrloches mit Bohrabschnitten aus der Datenstruktur.

Sie sollen eine Funktion (`loeschen`) implementieren, welche ein Bohrloch vom **Typ LOCH** mit allen zugehörigen Bohrabschnitten vom **Typ ABSCHNITT** aus der Datenstruktur entfernt und den verwendeten Speicherplatz frei gibt. Der Funktion (`loeschen`) soll als Funktionsparameter das Element vom **Typ LOCH** übergeben werden, welches sich nachfolgend (rechts) des zu löschenden Bohrloches befindet. Bei dem zu löschendem Bohrloch handelt es sich nicht um das erste Listenelement (Prüfung nicht erforderlich).

Vervollständigen Sie die Funktion (`loeschen`).

```
void loeschen (LOCH *data)
{
  //Hilfszeiger zum Zwischenspeichern
  LOCH *loch = data->prev;

  //Loeschen der Datenelemente von Typ ABSCHNITT
  while(        loch->down != NULL       ){
      ABSCHNITT *abschnitt = loch->down;
      loch->down = loch->down->down;
      free(abschnitt);
      
      
  }

  //Loeschen des Datenelements vom Typ LOCH

    data->prev = loch->prev;
    data->prev->next = data;
    free(loch);

    return;
}
```

Auch:
`data->prev = data->prev->prev;`
`loch->prev->next = data;`

d) Erstellen Sie eine Funktion zum Schreiben von Werten in eine Datei.

Schreiben Sie eine Funktion (`schreiben`), welche als Funktionsparameter ein Element vom **Typ BEGINN** übergeben bekommt. Wie in Aufgabe 20 a) beschrieben, wird durch das Element vom **Typ BEGINN** auf das erste Element vom Typ LOCH in der doppelt verketteten Liste referenziert. In der Funktion (`schreiben`) soll der im ersten Listenelement vom **Typ LOCH** gespeicherte Dichtewert in eine Datei (`inhalt.txt`) geschrieben werden.

Dazu soll die Datei geöffnet, der Dichtewert in die Datei geschrieben und die Datei anschließend gespeichert sowie geschlossen werden.

Prüfen Sie nach dem Öffnen der Datei zunächst, ob diese geöffnet werden kann. Sofern diese nicht geöffnet werden kann, geben Sie eine Warnung (`Fehler`) am Bildschirm aus.

Vervollständigen Sie die Funktion (`schreiben`) gemäß der obigen Ausführungen.

```
void schreiben(BEGINN *root)        Auch a,r+,w+,a+
  // Oeffnen der Datei
   FILE* pDB = fopen("inhalt.txt", "w");

  // Schreiben in Datei
   if (pDB == NULL){
       printf("Fehler");
   }
   else {
       fprintf(pDB, "%i", root->first->Dichte);
   }
   fclose(pDB);

  return;
}
```